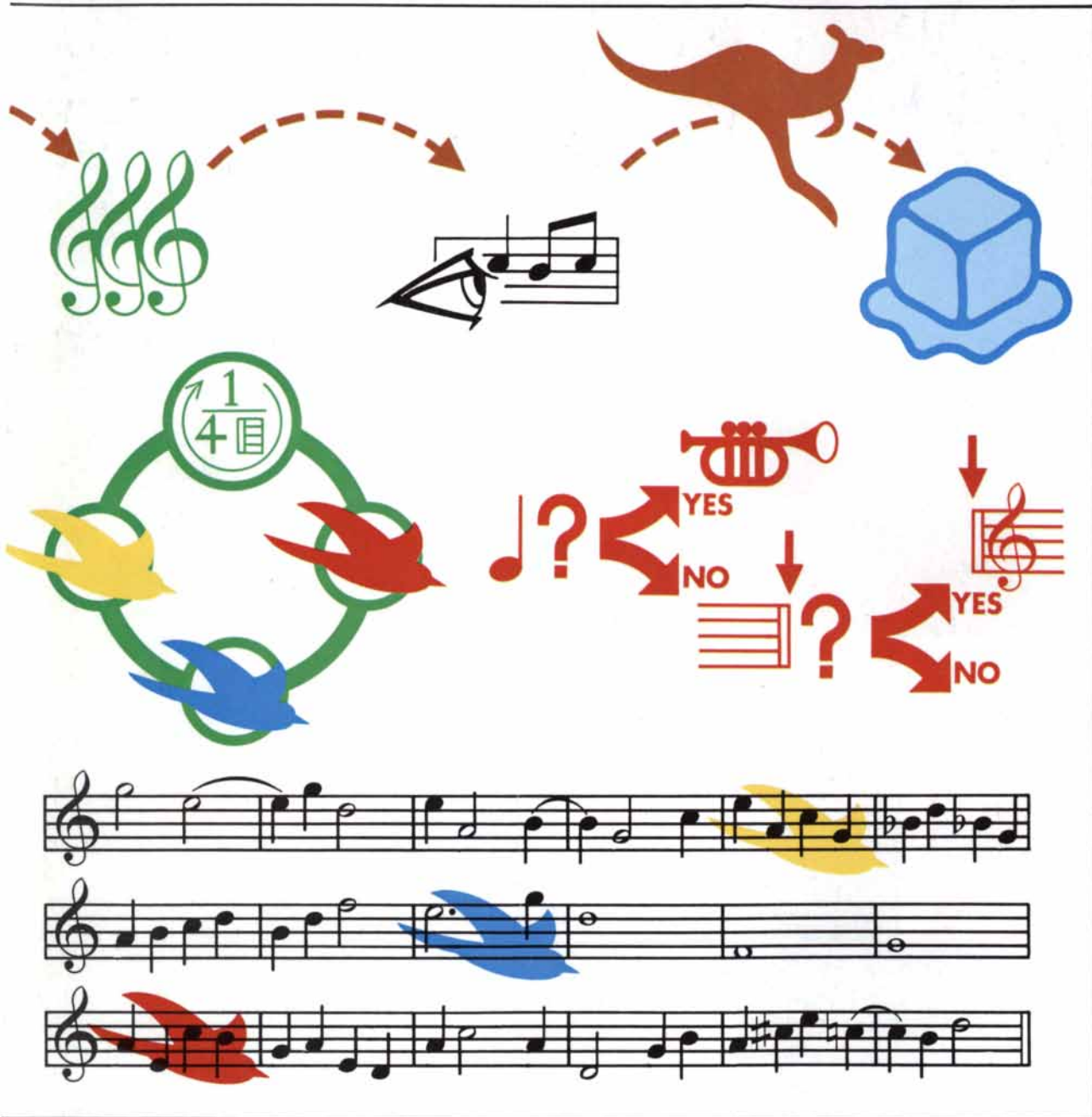


SCIENTIFIC AMERICAN

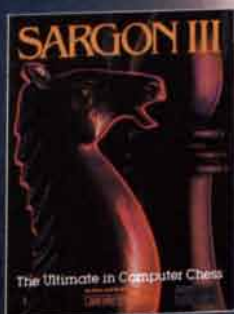


COMPUTER SOFTWARE

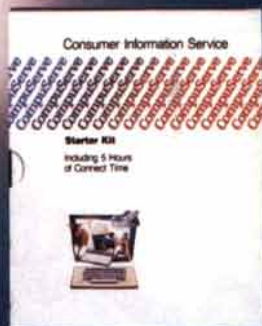
\$2.50

September 1984

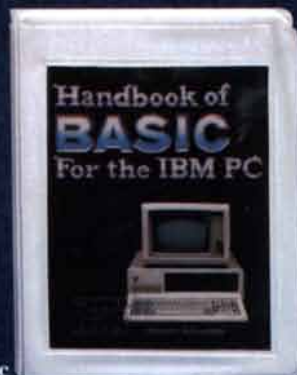
Discover



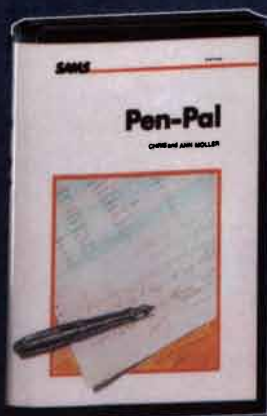
a



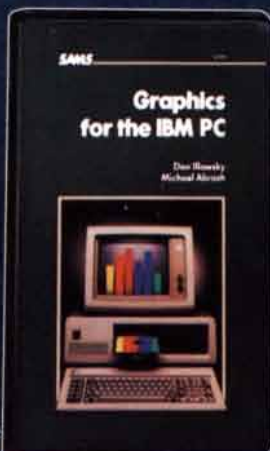
b



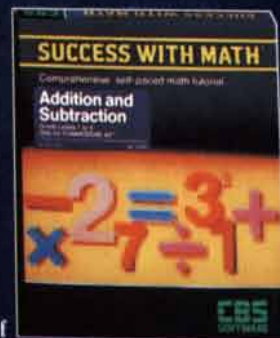
c



d



e

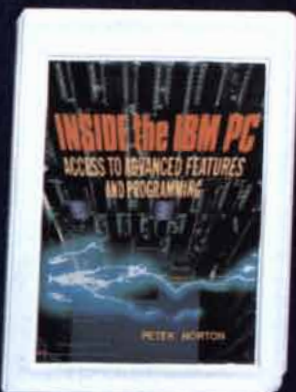


f

a) The ultimate in computer chess for the novice or expert/Apple, IBM. b) An easy to use interactive videotex service for all computers. c) A complete reference package designed specifically for the beginning IBM PC user. d) An easy to learn, easy to master and very powerful word processor/Apple. e) Easy to create graphics for

home or business for the IBM PC. f) A set of comprehensive, self-paced math tutorials/Commodore 64, Apple. g) An inside look at the IBM's microprocessor, operating system, PC-DOS and ROM. h) An enhanced BASIC to create amazing graphics with sound/Commodore 64. i) A special hands-on tutorial that makes assembly language.

Software



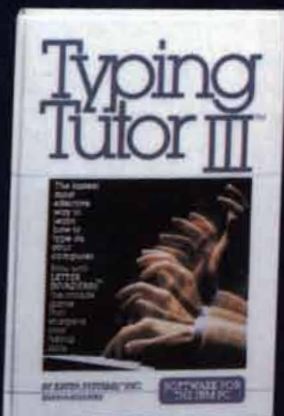
f



h



i



j



k



l

programming and interfacing easy to learn/
Apple. j) A simple, fast, effective way to
improve computer typing skills/IBM,
Commodore 64, Apple. k) A book and

software package of over 32 basic programs for
the IBM Personal Computer. l) An easy to
master program to teach your computer
new tricks/Apple.

at
Waldenbooks
Your new source for software.

J&B. It whispers.

RARE

BLENDED SCOTCH WHISKY
100% SCOTCH WHISKIES

BLENDED AND BOTTLED IN SCOTLAND BY

JUSTERINI & BROOKS LTD
St. James's Street, London, England

WINE MERCHANTS TO THEIR LATE MAJESTIES

KING GEORGE IV
KING WILLIAM IV
QUEEN VICTORIA
KING GEORGE III
KING EDWARD VII
KING GEORGE V
KING GEORGE VI

ARTICLES

- 52** **COMPUTER SOFTWARE, by Alan Kay**
Presenting an issue on the concepts and techniques that give form to the programmable machine.
- 60** **DATA STRUCTURES AND ALGORITHMS, by Niklaus Wirth**
They are the essential elements of a computer program and the key to verifying its correctness.
- 70** **PROGRAMMING LANGUAGES, by Lawrence G. Tesler**
A language transforms a computer into a "virtual machine" with features determined by software.
- 94** **OPERATING SYSTEMS, by Peter J. Denning and Robert L. Brown**
They span layers of complexity from keyboard commands to the details of electronic switching.
- 130** **COMPUTER SOFTWARE FOR WORKING WITH LANGUAGE, by Terry Winograd**
Programs readily manipulate linguistic symbols, but understanding is hampered by ambiguity.
- 146** **COMPUTER SOFTWARE FOR GRAPHICS, by Andries van Dam**
Interactive graphics is rapidly becoming the standard medium of communication with computers.
- 162** **COMPUTER SOFTWARE FOR INFORMATION MANAGEMENT, by Michael Lesk**
Stored data are of use only if information can be retrieved quickly in an understandable form.
- 174** **COMPUTER SOFTWARE FOR PROCESS CONTROL, by Alfred Z. Spector**
A process-control program cannot set its own pace but must respond to events in the real world.
- 188** **COMPUTER SOFTWARE IN SCIENCE AND MATHEMATICS, by Stephen Wolfram**
Simulation programs offer a new way to study natural phenomena and mathematical concepts.
- 204** **COMPUTER SOFTWARE FOR INTELLIGENT SYSTEMS, by Douglas B. Lenat**
Intelligence in problem solving is primarily a matter of constraining the search for solutions.

DEPARTMENTS

- 11** LETTERS
- 14** 50 AND 100 YEARS AGO
- 18** THE AUTHORS
- 22** COMPUTER RECREATIONS
- 39** BOOKS
- 82** SCIENCE AND THE CITIZEN
- 215** THE AMATEUR SCIENTIST
- 228** BIBLIOGRAPHY

BOARD OF EDITORS Gerard Piel (Publisher), Dennis Flanagan (Editor), Brian P. Hayes (Associate Editor), Philip Morrison (Book Editor),
Tim Appenzeller, John M. Benditt, Peter G. Brown, Ari W. Epstein, Michael Feirtag, Robert Kunzig,
Jonathan B. Piel, James T. Rogers, Armand Schwab, Jr., Joseph Wisnovsky

ART DEPARTMENT Samuel L. Howard (Art Director), Steven R. Black (Assistant Art Director), Ilil Arbel, Edward Bell

PRODUCTION DEPARTMENT Richard Sasso (Production Manager), Carol Eisler and Leo J. Petruzzi (Assistants to the Production Manager),
Carol Hansen (Electronic Composition Manager), Carol Albert, Karen Friedman, Karen O'Connor, Julio E.
Xavier

COPY DEPARTMENT Sally Porter Jenks (Copy Chief), Debra Q. Bennett, Mary Knight, Dorothy R. Patterson

GENERAL MANAGER George S. Conn

ADVERTISING DIRECTOR C. John Kirby

CIRCULATION MANAGER William H. Yokel

SECRETARY Arlene Wright

“We asked for an encore.”



*Ronald J. Cook,
Senior Vice President,
E.F. Hutton & Company Inc.
He helped pioneer the investment
industry's first direct client com-
munication service.*

Lotus gave us a Symphony."



"We've been using 1-2-3™ from Lotus™ almost from the day it was introduced.

"It's the one software program we've tried that meets the analytical needs of almost every department. From Equity Research to Corporate Finance. Right now, we're using 1-2-3 for planning, forecasting, and decision support.

"With the success we're having with 1-2-3, we were naturally excited about new Symphony™. Symphony's word processing simplifies our preparation of in-house reports and with Symphony's communication capability we can easily access a wide range of information sources.

"We also recommend Symphony to our clients because it's the ideal complement to

Huttonline™—our online service that gives clients direct access to their portfolios, stock quotes and critical investment research. All this right on their own PC's.

"With Symphony, our clients can now get this information in really useable terms, and in exactly the format they want: spreadsheet, graphics, database, or words. And like 1-2-3, Symphony comes with everything in one package.

"In a business where timely information is everything, Symphony clearly meets our needs."

To find out which Lotus product is best for you, visit your authorized Lotus dealer.

 **Lotus**
One great idea after another.™

SCIENTIFIC AMERICAN

CORRESPONDENCE

Offprints of more than 1,000 selected articles from earlier issues of this magazine, listed in an annual catalogue, are available at \$1.25 each. Correspondence, orders and requests for the catalogue should be addressed to W. H. Freeman and Company, 4419 West 1980 South, Salt Lake City, UT 84121. Offprints adopted for classroom use may be ordered direct or through a college bookstore. Sets of 10 or more Offprints are collated by the publisher and are delivered as sets to bookstores.

Photocopying rights are hereby granted by Scientific American, Inc., to libraries and others registered with the Copyright Clearance Center (CCC) to photocopy articles in this issue of SCIENTIFIC AMERICAN for the flat fee of \$1.25 per copy of each article or any part thereof. Such clearance does not extend to the photocopying of articles for promotion or other commercial purposes. Correspondence and payment should be addressed to Copyright Clearance Center, Inc., 21 Congress Street, Salem, MA 01970. Specify CCC Reference Number ISSN 0036-8733/84. \$1.25 + 0.00.

Editorial correspondence should be addressed to The Editors, SCIENTIFIC AMERICAN, 415 Madison Avenue, New York, NY 10017. Manuscripts are submitted at the authors' risk and will not be returned unless they are accompanied by postage.

Advertising correspondence should be addressed to C. John Kirby, Advertising Director, SCIENTIFIC AMERICAN, 415 Madison Avenue, New York, NY 10017.

Subscription correspondence should be addressed to Subscription Manager, SCIENTIFIC AMERICAN, P.O. Box 5969, New York, NY 10017. The date of the last issue on your subscription is shown in the upper right-hand corner of each month's mailing label. For change of address notify us at least four weeks in advance. Please send your old address (if convenient, on a mailing label of a recent issue) as well as the new one.

Name _____

New Address _____

Street _____

City _____

State and ZIP _____

Old Address _____

Street _____

City _____

State and ZIP _____



THE COVER

The illustration on the cover symbolizes the theme of this issue of SCIENTIFIC AMERICAN: computer software. The illustration is itself software: it is a program in a pictorial language called Mandala, under development by Jaron Z. Lanier and his colleagues at VPL Research in Palo Alto, Calif. Instructions are given to the computer by arranging icons, or small graphic symbols, on a display screen and setting them in motion. At the top a kangaroo hops from a triple-clef icon, which activates a program for playing three-part canons, to an icon that allows musical data to be viewed in traditional music notation and then to an ice cube, where the sequence of hops is "frozen" so that it can be referred to by a single symbol. An icon can represent a hierarchy of programming structures. The triple clef "expands" into the loop shown below it; the loop is executed once, launching (at intervals of four measures) the three birds that perform the canon. The sequence of instructions embodied in each bird is shown to the right of the loop. If a bird flying along the score is at a note, it sounds the note; otherwise, if it is at the end of the score, it returns to the beginning. The canon itself was composed by Lanier with the aid of the illustrated Mandala program.

THE ILLUSTRATIONS

Cover illustration by Jerome Kuhl

Page	Source	Page	Source
27-34	Alan D. Iselin	154	Evans & Sutherland
52	Timothy C. May, Intel Corporation	155	Lee Westover and Turner Whitted, University of North Carolina and Numerical Design Ltd.
54-56	Jerome Kuhl		
58-59	Alan Kay		
60-69	Alan D. Iselin	158-159	Alvy Ray Smith, Lucasfilm Ltd.
71	Steven P. Reiss, Brown University	160	Huseyin Kocak, Brown University
72-78	Alan D. Iselin		
82D	Rudolph Turner, Indiana University	161	Bryce Flynn, The Picture Group, Inc.
87	Ilil Arbel	162	Michael Lesk
88	Frank Schottler and Kevin S. Lee, University of California at Irvine	164-170	Edward Bell
		172	Michael Lesk
		174	Aydin Controls
95-106	Gabor Kiss	176-186	Hank Iken, Walken Graphics
130-144	Hank Iken, Walken Graphics	189	Quesada/Burke
147	Ned Greene, New York Institute of Technology	190-197	Ilil Arbel
		198-200	Quesada/Burke
148	Ian Worpole	203	Ilil Arbel
150	James K. Rinzler, Brown University	205	Douglas B. Lenat, Stanford University
151	Ian Worpole		
152	James K. Rinzler, Brown University	206-212	Ilil Arbel
		215-227	Michael Goodman



A year's worth of reports, plans, schedules, charts, graphs, files, facts and figures and it could all be lost in the blink of an eye.



The most important part of your computer may be the part you've considered least—the floppy disk. After all, there doesn't seem to be much difference between one disk and another. But now Fuji introduces a floppy disk that's worth a second look.

We designed our disk with the understanding that one microscopic imperfection can erase pages of crucial data. That's why every Fuji Film Floppy Disk is rigidly inspected after each production process. And that's why each one is backed with a lifetime warranty.

We've even considered how carefully a disk has to be handled, so we designed user-friendly packaging that makes it easier to get the disk out of the box. And we provided plenty of labeling space, so you won't have any trouble telling which disk is which.

So think twice before buying a floppy disk. And then buy the one you won't have any second thoughts about.

Fuji Film Floppy Disks.

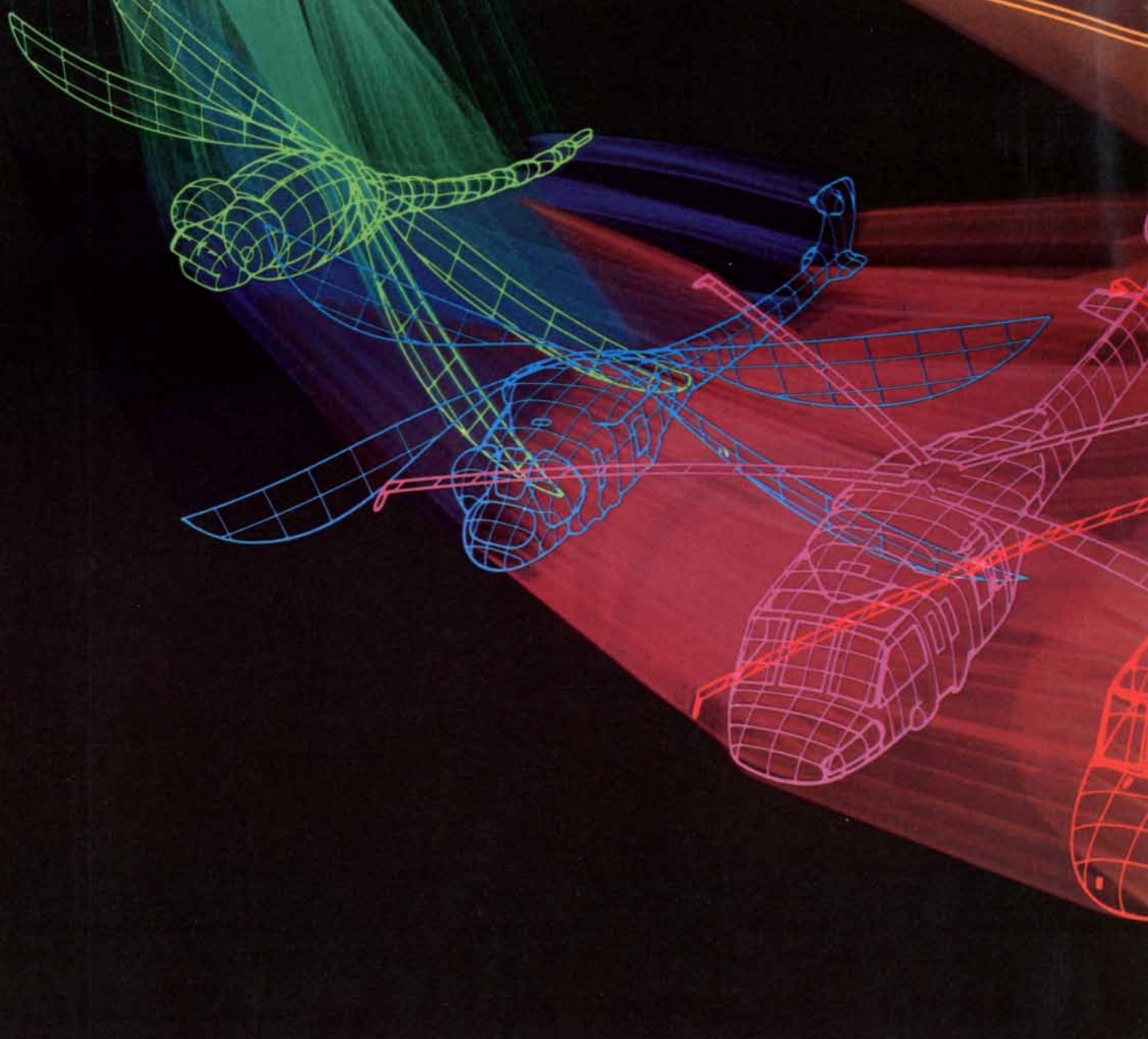


FUJI.

Nobody gives you better performance.

© 1984 Fuji Photo Film U.S.A., Inc., Magnetic Products Div., 350 Fifth Avenue, NY, NY 10118

© 1984 SCIENTIFIC AMERICAN, INC



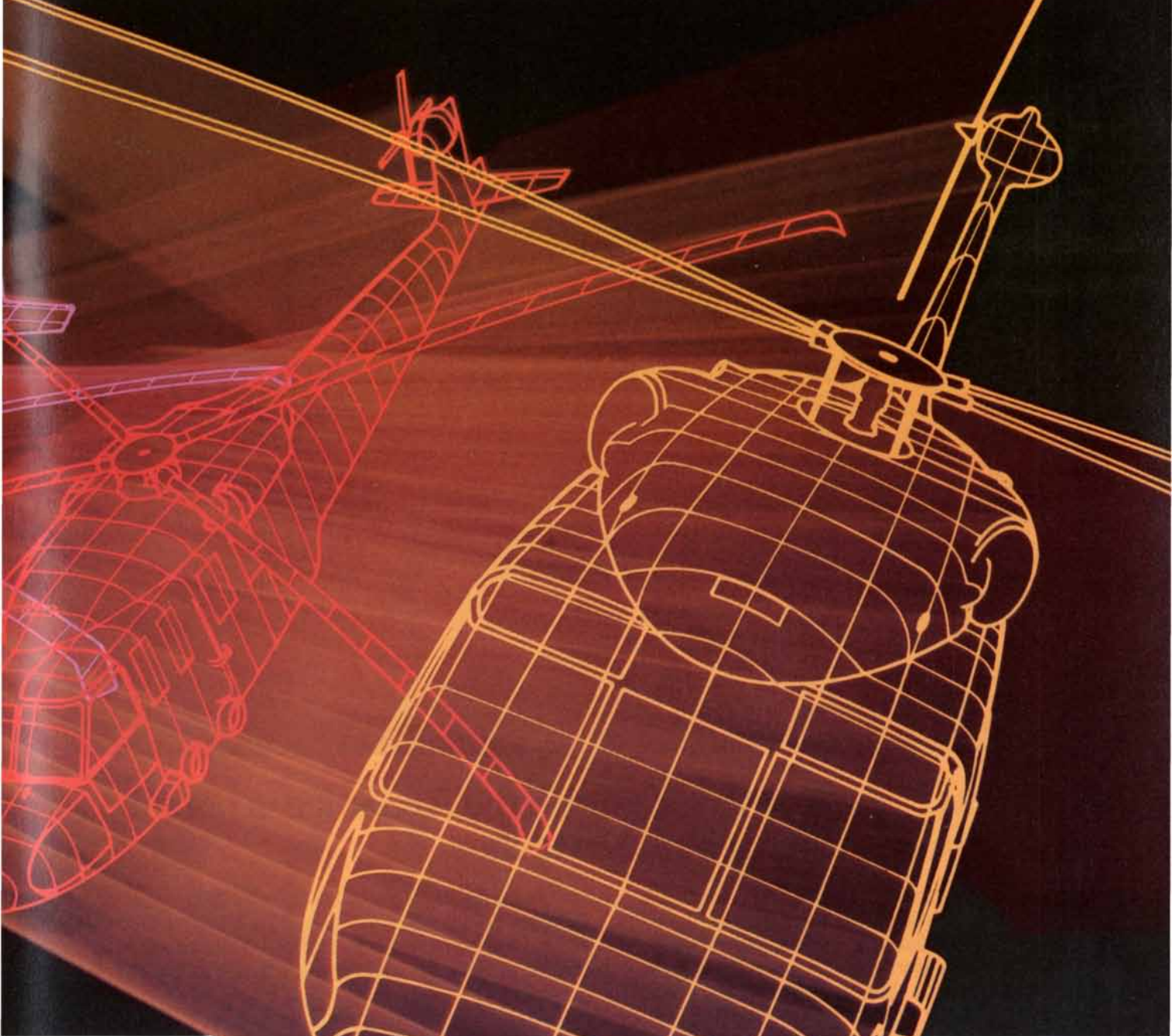
Artificial Intelligence at Sikorsky Aircraft

Exploring the evolution of vertical flight—at the speed of light.

The dragonfly perfected the fine art of vertical flight about 250 million years ago. Nature's organic design system gave us a perfect example of form following function.

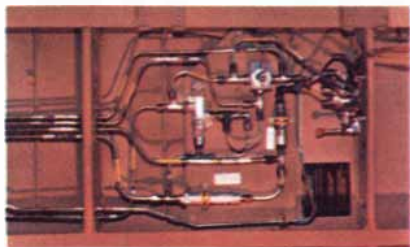
Today, United Technologies Sikorsky Aircraft is building an "organic" design system for advanced vertical flight aircraft. And it's an information challenge that staggers the imagination.

Says Joe Piteo, Sikorsky Chief of CAD/CAM: "Our goal is to gather in one central system *everything* our engineers know. We want to extract and store the very process of their decision-making. Once we've added this 'creative logic component,' our system will



make a quantum leap from being merely a repository for data, to being an *information* base that is truly 'intelligent.'"

Using CAD/CAM software and two high-speed IBM 3081 computer systems, the Sikorsky team has tackled the design and manufacture



With the help of an "intelligent" system, fluid line systems like these can be designed in minutes instead of weeks.

of fluid lines. More than 2,000 tube drawings and their related design and manufacturing logic have been fed into the data system.

"Down the line," Michael Adami, project coordinator, says, "as more and more parameters are included in the software, the system will be able to 'see' more and more of the overall design. Ultimately, for example, the aircraft's tubing could be considered as a structural

element and, as such, used in achieving the ideal blend of function, weight and strength."

"Our goal," says Piteo, "is to weld together these islands of knowledge and the automation of industrial design into a powerfully productive 'organism' which understands the relationship of its parts and works toward a common end."

For more information on IBM's programs for engineers and scientists write IBM Engineering and Scientific Marketing, 1133 Westchester Ave., White Plains, NY 10604.



600 miles north of the Arctic Circle, a Canadian ice breaker cuts a trail in the search for the HMS Breadalbane.



Searching for a shipwreck under the Arctic ice is not for ordinary divers. Or ordinary watches.

Temperatures reach 50° below zero. The wind chill factor: -100°F. Ice floes are a constant hazard.

For Dr. Joe MacInnis, the conditions were perfect.

In this hostile world lay a unique treasure: the HMS Breadalbane, a three-masted British barque lost in 1853 during the search for the Northwest Passage. MacInnis was determined to find her.

Using delicate sonar, he was able to pinpoint the location of the vessel 340 feet below the surface, near Beechey Island. Divers discovered one of the most perfectly preserved shipwrecks found in any ocean.

In the Arctic, ordinary divers find it difficult to function. So do ordinary watches. For the past 15 years, Dr. MacInnis, the first to dive and film under the North Pole, has chosen one watch: the Rolex Submariner.

"I've worn it everywhere. From the North Pole ... to the Red Sea."

As for the future, there are still more explorer ships resting on the Arctic floor.

A fitting challenge for Dr. Joe MacInnis and his Rolex.

© 1984 Rolex Watch, U.S.A., Inc.


ROLEX



*Pictured: The Rolex® Submariner Date Chronometer. Pressure-proof to 1,000 feet.
Write for brochure. Rolex Watch, U.S.A., Inc., Dept. 542, Rolex Building, 665 Fifth Avenue, New York, New York 10022-5383.
World headquarters in Geneva. Other offices in Canada and major countries around the world.*

LETTERS

Sirs:

I cannot resist tweaking the toes of Alan H. Guth and Paul J. Steinhardt for propagating, in their otherwise excellent article ["The Inflationary Universe," *SCIENTIFIC AMERICAN*, May], an old myth, namely that the universe, according to Einstein's theory of gravity, must be infinite if its energy density is less than the critical density for recollapse.

When the energy density is too low for recollapse to occur, the average curvature of three-dimensional space must be either negative or zero (flat). It has long been known to mathematicians that a world of negative or zero curvature need not have infinite volume. The mental block that appears to afflict many cosmologists is their inability to visualize anything but 3-spheres (positively curved spaces) and infinite hyperboloids (negatively curved spaces). Even if the energy density is supercritical, the universe need not be a 3-sphere topologically. It could be what mathematicians call a quotient space, the divisor being any discrete subgroup of the symmetry group of the 3-sphere. Hyperboloids too can be factored into finite quotient spaces.

Although the possibility that an ever expanding universe might be finite cannot do much to help the horizon problem that besets the original big-bang model, it is by no means inconsistent with either the isotropy of the three-degree background radiation or the inflationary model.

BRYCE S. DEWITT

University of Texas at Austin

Sirs:

We agree with Professor DeWitt that a universe with an energy density less than the critical density need not be infinite, as quotient spaces exemplify. Instead these more exotic spaces are characterized by a periodicity. An observer who can see to distances greater than or equal to the periodicity length would see images of himself.

In the context of standard cosmology these possibilities are frequently neglected because they violate the assumption of uniformity, which states that the universe should appear the same in all directions to observers at all points in space. A quotient space is locally uniform in that the universe appears the same to all "nearsighted" observers: those who cannot see their own images a periodicity length away. It is globally nonuniform in that "farsighted" observers would see different patterns.

Thus if the energy density is less than

the critical density and one assumes the universe is globally uniform, it must be infinite. On the other hand, one might assume only local uniformity, in which case quotient spaces of finite volume would be allowed. Because the inflationary model can explain the creation of huge regions of homogeneity even in an inhomogeneous universe, however, it seems to us that one might just as well give up the assumption of local homogeneity as well. One can simply assume that the universe might do almost anything at distances far beyond those we observe. In that case it would be possible for the universe to have a finite volume even without the fancy topology of a quotient space.

ALAN H. GUTH

Massachusetts Institute of Technology
Cambridge

PAUL J. STEINHARDT

University of Pennsylvania
Philadelphia

Sirs:

We appreciate the general accuracy of your summary of our paper on the low-fertility zone in sub-Saharan Africa ["Science and the Citizen," *SCIENTIFIC AMERICAN*, April]. An essential point was missed, however.

The low-fertility zone developed after the establishment of the Congo Free State and the French Congo, two areas of unusually ruthless colonization: Even those societies that were fairly permissive about premarital sexual relations did not allow 10-year-old girls to have intercourse until the social order fell apart as a result of massive terror and the breaking up of families. Such early sexual relations are probably the only explanation for sterility on such a scale by 15 or 16 years of age.

Since we first discussed these findings a dozen years ago, David Voas, for a Ph.D. dissertation at the University of Cambridge, has checked out the thesis in great detail and has shown the extremely close geographical identification between the areas of low fertility and those of maximum social disruption. Where an area has maintained high fertility in spite of being included in a colonial concession, it turns out the company involved failed to exploit that part of the concession.

JOHN C. CALDWELL

PAT CALDWELL

Department of Demography
Research School of Social Sciences
Australian National University
Canberra

NWA STATPAK
MULTI-FUNCTION STATISTICS LIBRARY

Since 1978, Northwest Analytical, Inc. (NWA) has supplied personal computer users with the software tools for quantitative analysis. NWA STATPAK is currently used by thousands of professionals as an effective alternative to time-share computing systems. NWA STATPAK is designed so that beginning users can rapidly learn to use the system while providing experienced analysts with the power to handle major projects.

- Extensive statistics library
- Data management and manipulation
- Streamlined workflow
- On-line choice of menu or command line operation
- ASCII format files for data interchange with other software
- Excellent user support
- Supplied in MS-BASIC or Personal BASIC source code form
- Versions available for
 - CP/M-80
 - CP/M-86
 - MS-DOS(PC-DOS)
 - CTOS(BTOS)
 - Macintosh
- \$495 Retail
 - Volume, Academic, and Site Licenses Available

Northwest Analytical, Inc.

520 N.W. Davis Street
Portland Oregon 97209
503-224-7727
RCA Telex 296565 NWA STAT

CP/M and Personal BASIC are trademarks of Digital Research Inc. MS-DOS and MS-BASIC are trademarks of Microsoft Corporation. PC-DOS is a trademark of IBM Corporation. CTOS is a trademark of Convergent Technologies, Inc. BTOS is a trademark of Burroughs Corporation. Macintosh is a trademark of Apple Computer, Inc.

Software that

Summary:

GTE computer scientists are producing software to help develop a variety of things, from telephone networks to integrated circuits—software that writes other software; software that designs microcircuits; even software that has its own intelligence.

Modern telephone systems are essentially special-purpose computers. Their software represents more than half of their cost, and this figure is rising.

GTE research is aiming to improve both productivity in software design and its quality. Two different compiling systems are helping us (compilers aren't new, of course, but these are special).

New-generation software compiler.

Our General-Purpose Compiling System (GPCS) does not work with

only one language and computer as ordinary compilers do. GPCS was designed to be independent of host and target machine, and support Pascal, Ada and CHILL. It was also designed to permit automatic language translation between these supported languages.

Our researchers have completed work on GPCS, and it is now in our software development facilities.

There, it will write software for new switching systems at increased productivity levels.

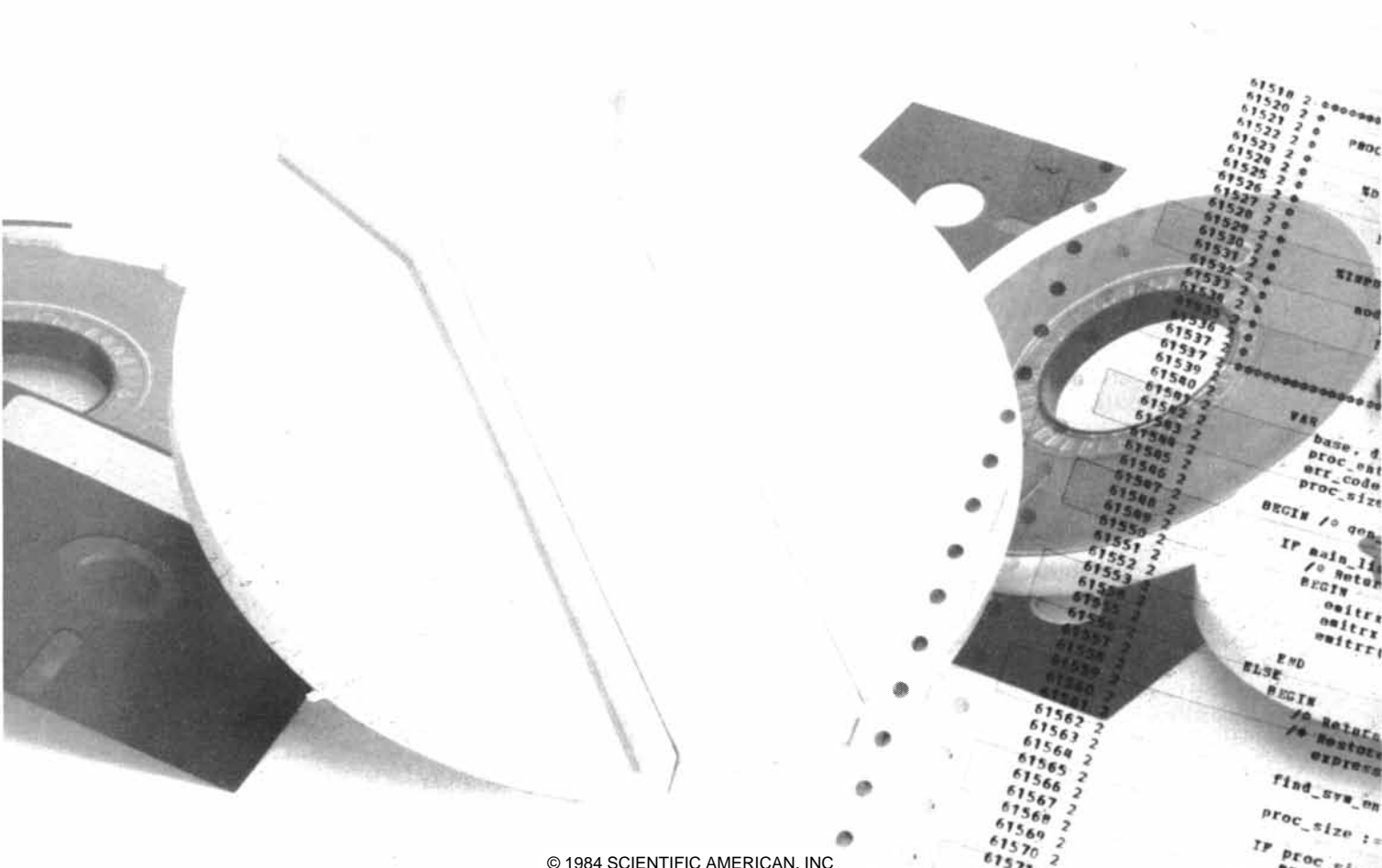
Designing hardware with software.

The demand for more and more function has inevitably led to higher costs for designing very large-scale integrated circuits. Designing a 32-bit microprocessor with today's tools, for instance, can take over 100 man-

years. Our silicon compiler project promises to reduce this time by 90 to 95 percent.

Using this design-automation tool, designers will describe what the circuit should do functionally, rather than graphically. The tool will translate (compile) requests into appropriate geometry without the laborious circuit layout and routing formerly required.

It is interesting to note that this project may result in the use of custom-logic circuits where microprocessors are used today. After all, when VLSI circuits become cheap and easy enough to produce, it may be preferable to integrate them into systems rather than write software for a microprocessor.



creates software.

Simplifying database access.

With the advent of communications networks like GTE Telenet, many databases have become accessible at relatively low cost. Each, however, may have its own access control mechanism, command language, and output format; and users may want to switch from one database to another with a transparent interface.

We have been working on a way for data-network subscribers to use natural language, which is translated into the various formats required by the different database services. It is called FRED—Front End for Databases. This technology incorporates natural-language processing and the expert systems areas of Artificial Intelligence.

FRED acts like a librarian. It recognizes the meaning of natural-language input and sends a request to the appropriate database.

Not all our computer science projects have the drama of artificial intelligence research, of course. However, our goal in all these investigations is to create software to help improve quality and productivity for advanced communications products and services.

The box at the right lists some of the pertinent papers GTE people have published on software and related subjects. For any of these you are invited to write GTE Marketing Services Center, Department TPIIIA, 70 Empire Drive, West Seneca, NY 14224 or call 1-800-828-7280 (in N.Y. State 1-800-462-1075).

Pertinent Papers.

The System Compiler, 1983 IEEE International Symposium on Circuits.
The MacPitts Silicon Compiler: A View from the Telecommunications Industry, VLSI Design, May-June, 1983.

An Intelligent Communication Assistant for Databases, Proceedings of the IEEE COMPSAC 83.

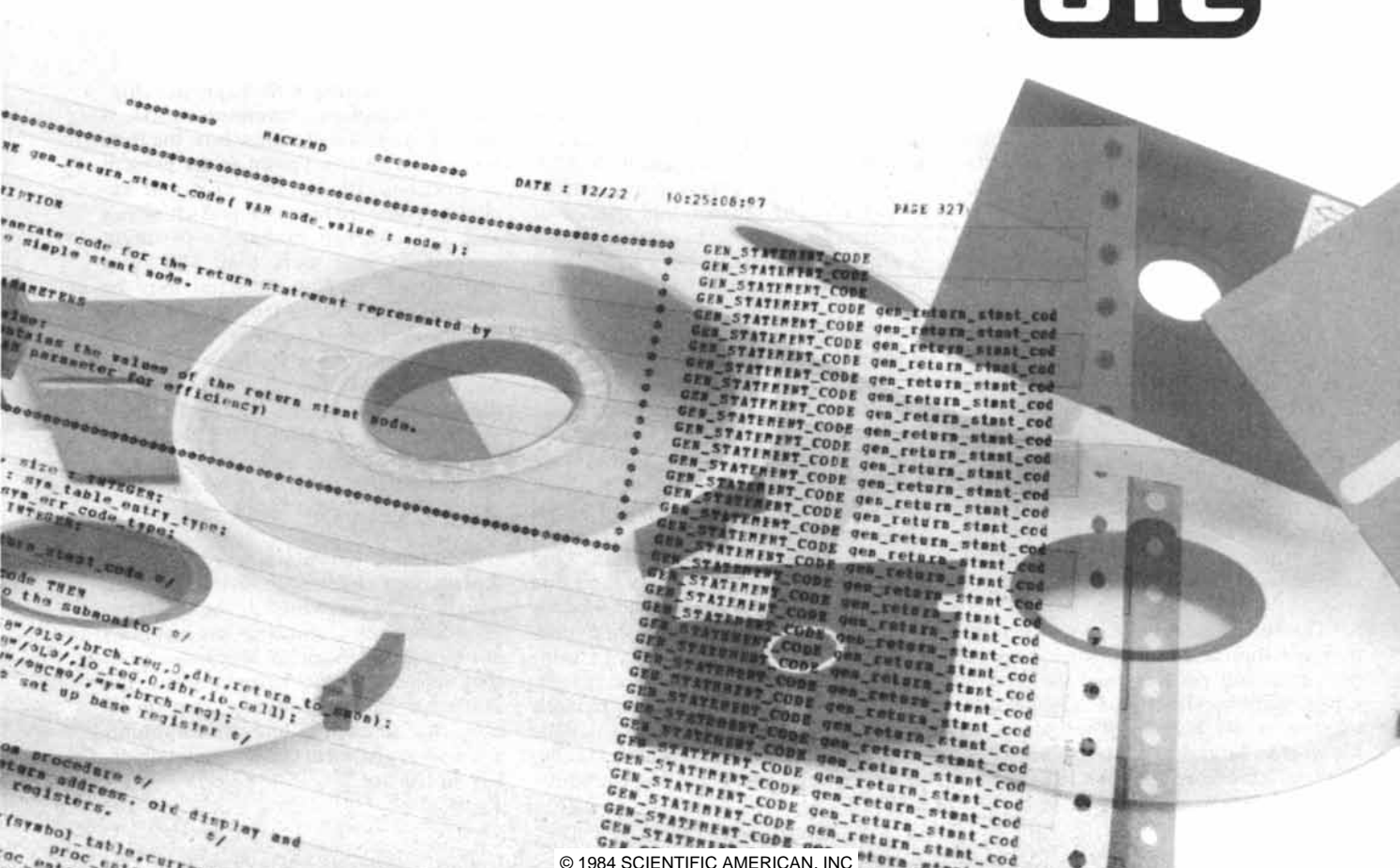
A Natural Language Interface for Medical Information Retrieval, AAMSI Congress, Computer Applications in Medicine.

Separate Compilation for Block Structured Languages: A Comparative Study of CHILL and Ada Compiling Systems, Proceedings of the IEEE Symposium on Application and Assessment of Automated Tools for Software Development.

An Efficient Compilation Strategy for Very Large Programs, ACM SIGPLAN 82 Symposium on Compiler Construction, June, 1982.



GTE



50 AND 100 YEARS AGO

SCIENTIFIC AMERICAN

SEPTEMBER, 1934: "A seasoned army of hard-rock miners is pressing forward, out on the California desert, on the largest tunnel-driving program ever undertaken in the history of engineering. They are excavating 29 18-foot bores, totaling 91 miles in length, through the bleak mountain ranges between the Colorado River and the Coastal Plain of southern California. Through these tunnels eventually will be turned a billion gallons of water daily to serve the 13 municipalities that comprise the Metropolitan Water District of Southern California. Longest of the aqueduct tunnels is the East Coachella bore, 18 miles from end to end. The aqueduct depends upon Boulder Dam for proper regulation of the Colorado River and for cheap electric energy needed to pump the aqueduct water over the mountain ranges. Thirty-six per cent of the power generated at Boulder Dam will be used for this purpose. Few realize that the aqueduct overshadows the dam in size. Cost of the dam proper is about 100,000,000 dollars, whereas the aqueduct bond issue, voted in 1931, was for 220,000,000 dollars."

"The America's Cup has become the symbol of supremacy under sail. It is not to be wondered, therefore, that in a mechanical age the boats built to challenge or defend should have become mechanized to an extent that would have caused sailors of the old school of beef and brawn to blush with shame. The chief interest centers in the new yacht, *Rainbow*. Perhaps the greatest engineering feat in the construction of the new boat is her mast of duralumin, some 165 feet in length. It is pear-shaped in section, being about 30 inches in diameter fore and aft by 18 inches the other way. It was made by the Glenn L. Martin Company, airplane manufacturers. The British challenger, again calling upon the knowledge of aerodynamics acquired in airplane development, has done much experimenting with model sails in wind tunnels."

"The newspapers have given the public more than adequate information on the plans and prospects of the stratosphere flight by Major W. E. Kepner and Captain A. W. Stevens. The balloon is the world's largest and has a capacity of 3,000,000 cubic feet. When leaving the

ground the balloon will be less than one-tenth filled with hydrogen, which will gradually expand as the balloon rises to the thinner air. At the top of its flight, nearly 15 miles above sea level, the balloon will have become a sphere 180 feet in diameter."

"The 'synthetic rubber' tire is now an accomplished fact! These relatively insignificant words tell a story of tremendous economic significance. They indicate the successful solution of a long fight to ensure for the United States a source of rubber goods and in particular tires which would make us independent of foreign producers of rubber in time of war. Tires made entirely of Du Prene, the so-called synthetic rubber developed by the Du Pont company, have been built by the Dayton Rubber Manufacturing Company, and severe tests have proved these as tough and durable as tires made of natural rubber."

SCIENTIFIC AMERICAN

SEPTEMBER, 1884: "Rufus Porter, the original founder of the SCIENTIFIC AMERICAN, died recently at New Haven, Conn., in the 93rd year of his age. He was a remarkable natural genius. He was in school learning Noah Webster's spelling book at the age of four; spent six months at Fryburg Academy when 12 years old; beyond this he had no educational advantages. He had become quite an adept in the making of all sorts of mechanism. He was also something of a musician; he played the fife and the violin, and wrote poetry. In 1807 his family concluded it would be best for him not to fiddle any longer with life, but to settle down to something solid and useful, in short, become a shoemaker, like his elder brother. But it was soon seen that he was not cut out for this species of industry. In 1810 he was apprenticed to a house painter; in 1813 he painted sleighs, beat the drum for the soldiers, taught others to do the same and wrote a book on the art of drumming. In 1814 he was enrolled in the militia; after this he taught school at Baldwin, married at Portland, taught at Waterford, made wind grist mills at Portland, painted in Boston, the same on through New York to Alexandria, Va. A peculiarity which he developed about this time, and which continued through life, was a frequent change of place and occupation. One of his most profitable businesses at this time was portrait painting. He made a camera obscura and was enabled to produce a satisfactory portrait in 15 minutes, for which his customers readily paid a dollar. He adorned his camera box with bright colors, bought a light handcart for locomotion, planted a flag on his vehicle, and with this attractive

establishment was welcomed in every town and village. He invented a revolving almanac and suddenly stopped painting to make and introduce it. In 1824 he adopted the profession of landscape painter. In 1825 he invented a successful cord-making machine. From this time on he figures very often as an inventor, producing among other things a wonderful clock, a steam carriage, washing machine, signal telegraph and fire alarm. In 1840 he was offered an interest in a newspaper called the New York *Mechanic* and at once decided to become an editor. He made it ostensibly a scientific newspaper, the first of its kind in the country. The paper prospered, but his attention was as usual diverted to something else, and in a few months' time the publication was stopped. He next learned the art of electroplating, and did profitable work. About this time the religious mania of the Millerite people struck him, and he was among the most ardent believers who hourly expected the second advent of the Messiah. In 1845 he was again in New York, doing electroplating. Here he wrote a prospectus for a new paper, which he entitled the SCIENTIFIC AMERICAN, and began its issue weekly, with a cash capital of one hundred dollars. He did not, however, continue long in charge of the publication. After running it for six months the desire and necessity for a change once again came over him. During the remaining half-century, nearly, of his life he was chiefly occupied with his inventions and regularly moved from place to place, but did not so often recur to his old profession of portrait painting."

"To give an idea of the rapid development of telephonic communication, it may be interesting to show how the Boston exchange has grown to its present proportions. Besides the principal exchange there are now two branch offices and 17 suburban exchanges in direct connection, and more than 200 cities and villages in New England may be reached by telephone from Boston. The new 'central office' together with the two branch offices gives Boston a capacity for 6,000 lines. Taking the increase in the United States, the figures are even more startling. In May, 1877, there was but one exchange with five subscribers; in January, 1884, there were 906 exchanges with 123,625 subscribers."

"Madame Kowalevski, a native of Russia, is a celebrated mathematician, who lectured last winter at the University of Stockholm, and who has just been appointed Professor of Mathematics at that university. We believe this is the first time, since the middle ages (in Italy), that a woman has been appointed to an academical chair at any university in Europe."

Without us, the home of free speech could lose the power of speech.



In the communications industry, a leading buzzword for the future is fiber optics. At NYNEX, it's a word for yesterday.

Because NYNEX has 30,000 miles of these hair-thin fibers in place now. In the most information-hungry communities in the country: Boston, New York and their surrounding areas.

It's one of the world's largest lightwave installations. It transmits calls, computer data, and saves space for the future in congested tunnels. Water can't harm it. And neither can extreme temperatures.

In short, it's a network tough enough to make sure the home of free speech never loses the power of speech. It's what you get when you team New England determination with New York foresight.

NYNEX is the parent company of New York Telephone and New England Telephone plus other subsidiaries that offer mobile services, directory publishing and business communication equipment.

For information and a copy of our new Profile, write Tony Parra, Director of Investor Relations, NYNEX, P.O. Box 2945, NY, NY 10185.

Tough demands breed tough minds.

NYNEX

WHAT SHOULD YOU WORKING "PARTNERSHIP"



"At Container Corporation of America, an HP computer network helps us respond to customers faster, and saves us \$600,000 a year.

"It used to take us two days to estimate the cost of a job for a customer. With the HP network we can come up with an accurate figure in minutes. We now have the information we need to provide better quality packaging."

Container Corporation of America is one of the nation's leading paper-board packaging manufacturers. The company has a network of 47 HP 3000 computers operating in 78 manufacturing plants.

At company headquarters in Chicago, Jeff Norkin, Vice President, says, "Using HP's Productivity Tools, our staff developed a wood costing and payables system in half the time it took to develop a similar system in the past. The flexibility of the HP 3000 network enables us to design and implement additional systems as we need them, without costly conversions.

"We've seen that we can count on HP's technical expertise to provide solutions to new challenges as they arise. Based on our success with HP systems, we plan to expand our network with 10 additional HP 3000s."



EXPECT FROM A WITH HEWLETT-PACKARD?

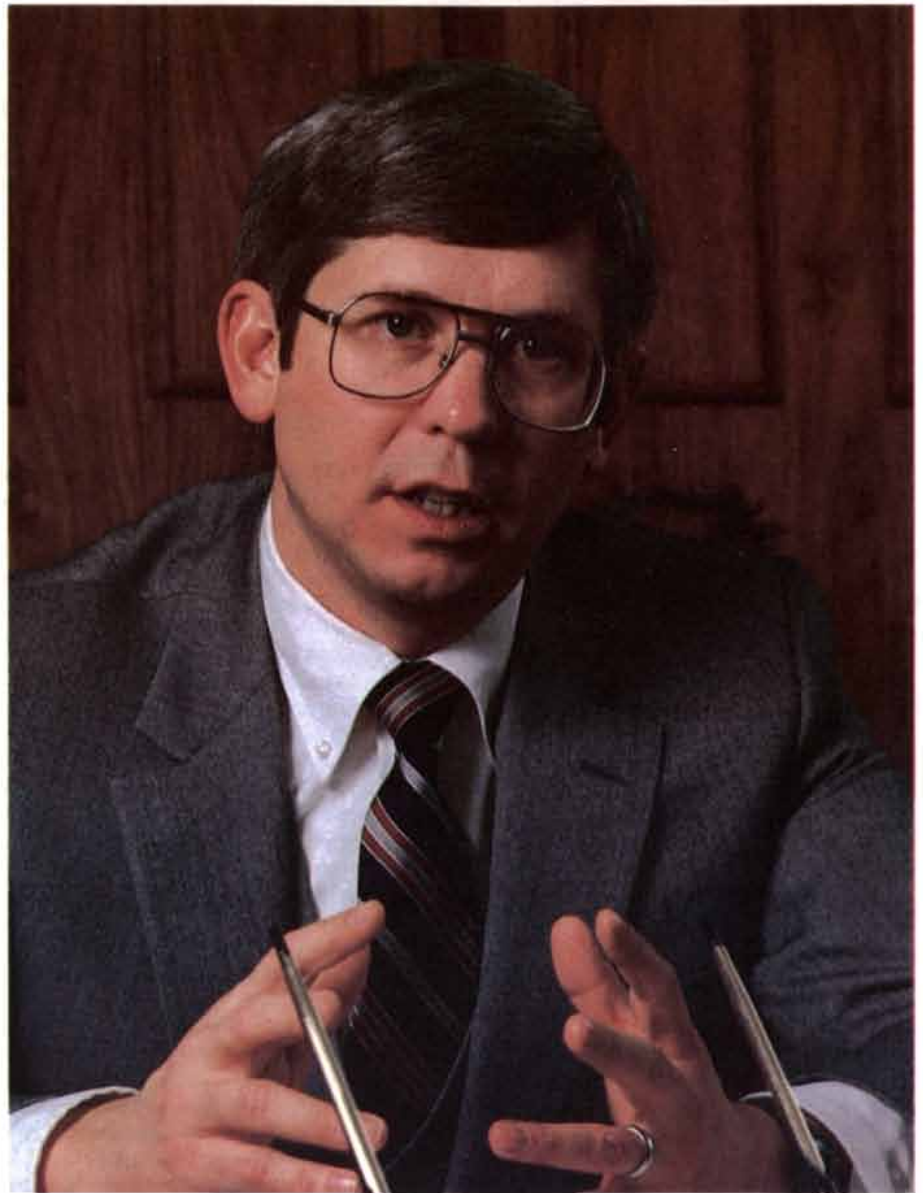
“At Fairview Hospital, an HP CareNet system facilitates critical care and saves \$54,000 a year in monitoring costs.

“Because the HP computer-based system delivers extremely accurate vital sign data, we no longer have to repeat procedures or revert to manual methods. On-line monitoring eliminates the need for outside hemodynamic monitoring services, so our staff accesses critical data immediately.”

At Fairview Hospital in Minneapolis, Bill Maxwell, Hospital Administrator, says: “HP offered us a total solution to intensive care monitoring—a means to deliver cost-effective quality care. HP CareNet gives us control over our monitoring strategy. And we can continuously monitor all patients from any location in the network.

“The flexibility of HP hardware will enable us to expand our network threefold without significantly increasing our hardware costs. And through HP’s upgrade program, we can put tomorrow’s technology on today’s system without increasing our capital investment.

“Our HP system is maintaining close to 100% uptime. The reliability of our HP network and HP’s support team has convinced us to equip another Fairview intensive care unit with an HP CareNet system.”



Results with assurance.



**HEWLETT
PACKARD**

0002407A

THE AUTHORS

ALAN KAY ("Computer Software") is an Apple Fellow at Apple Computer, Inc. While pursuing a vocation as a jazz musician, he earned B.A. degrees in pure mathematics and in molecular biology at the University of Colorado at Boulder and then did doctoral work at the University of Utah. He joined the Stanford Artificial Intelligence Laboratory and became a founding member of the Palo Alto Research Center (PARC) set up by the Xerox Corporation in 1971. There Kay and others developed a prototype of the first personal computer. Kay was instrumental in two personal-computer developments: "windows," separate areas of the computer display in which different tasks can be accomplished side by side, and the "mouse," a desktop controller that allows rapid movement of a screen cursor. In 1981 Kay joined Atari, Inc., where until this May he was chief scientist.

NIKLAUS WIRTH ("Data Structures and Algorithms") heads the division of computer science at the Swiss Federal Institute of Technology (ETH) in Zurich. A boyhood enthusiasm for radio-controlled model airplanes led him to study electronics, and he earned degrees in electrical engineering from ETH (1959) and Laval University in Quebec (1960). He went on to graduate studies in the U.S. at the University of California at Berkeley, where he received his doctorate in 1963. There he became interested in computer languages; as assistant professor in Stanford University's newly formed computer-science department from 1963 to 1967 he took part in the development of the computer language Algol W. After returning to Switzerland he created the structured programming language Pascal. Recently Wirth has explored the concept of tailoring hardware to software.

LAWRENCE G. TESLER ("Programming Languages") manages a software-development group in the Macintosh Division of Apple Computer, Inc. While studying at Stanford University for his bachelor's degree, which he got in 1965, he founded a small software company. He remained in business for five years, then joined the Stanford Artificial Intelligence Laboratory, where he did research in cognitive simulation and document formatting. In 1973 he moved to the Xerox Corporation's Palo Alto Research Center, where his work focused on software for personal computers. He began his career at Apple in 1980 as manager of applications-software development for the Lisa computer. Tesler writes: "Although I have worked in many areas of software engi-

neering and computer science during the past 24 years, a recurring theme has been the user interface: making it easier for people to get computers to do their bidding."

PETER J. DENNING and ROBERT L. BROWN ("Operating Systems") are specialists in computer systems organization at the Research Institute for Advanced Computer Science (RIACS), part of the National Aeronautics and Space Administration's Ames Research Center in Mountain View, Calif. Denning is director of RIACS and Brown is a staff scientist. Denning's degrees are in electrical engineering: a B.E.E. (1964) from Manhattan College and an M.S. (1965) and a Ph.D. (1968) from the Massachusetts Institute of Technology. He has taught electrical engineering at Princeton University and computer science at Purdue University. Brown was graduated from Ohio Wesleyan University in 1975 with a bachelor's degree in mathematics and is working toward a Ph.D. in computer science at Purdue.

TERRY WINOGRAD ("Computer Software for Working with Language") is associate professor of computer science and linguistics at Stanford University. He is a graduate of Colorado College and the Massachusetts Institute of Technology, where he received his doctorate in applied mathematics in 1970. He taught at M.I.T. until 1973 and then joined the Stanford faculty. Since 1973 he has served concurrently as a consultant at the Xerox Corporation's Palo Alto Research Center. Winograd pursues his research interests—artificial intelligence, computational linguistics and cognitive modeling—at Stanford's Center for the Study of Language and Information, and in addition he is a member of the National Executive Committee of Computer Professionals for Social Responsibility.

ANDRIES VAN DAM ("Computer Software for Graphics") is chairman of the department of computer science at Brown University. A native of Holland, he studied at Swarthmore College and the University of Pennsylvania, where in 1966 he obtained a Ph.D. in computer science—only the second to have been awarded in the U.S. At Brown he was a founder of the computer-science department, and he has been instrumental in initiating the campus-wide installation of computer work stations. Van Dam is coauthor of *Fundamentals of Interactive Computer Graphics*.

MICHAEL LESK ("Computer Software for Information Management") is

Division Manager of Computer Science Research at Bell Communications Research, Inc., in Murray Hill, N.J. After earning a Ph.D. in chemical physics at Harvard University in 1969, he joined the technical staff of Bell Laboratories. His interest in data bases has led him to develop a program for automobile route finding and to experiment with a computerized library-cataloguing system. Recently Lesk has taught at Columbia University as an adjunct lecturer in the department of computer science.

ALFRED Z. SPECTOR ("Computer Software for Process Control") teaches computer science at Carnegie-Mellon University. He went to Harvard College, where he got an A.B. in applied mathematics, and Stanford University, where he was awarded a Ph.D. in computer science in 1981. While studying for his doctorate, he worked at IBM's San Jose Research Laboratory. He took up his current job in 1981. Over the past two years he has helped to design an integrated file system for a campus computerization project undertaken jointly by Carnegie-Mellon and IBM. Spector writes: "I like designing and programming complex computer systems because of the challenge of integrating diverse techniques into cohesive systems that solve practical problems."

STEPHEN WOLFRAM ("Computer Software in Science and Mathematics") has been a member of the Institute for Advanced Study at Princeton since 1982. Born in London, he was educated at Eton College and the University of Oxford and then came to the U.S. to study at the California Institute of Technology, where he received a Ph.D. in theoretical physics in 1979. In 1980 he joined the faculty of Cal Tech, and he remained there until he accepted his present position. Wolfram has done work in high-energy physics, cosmology and statistical mechanics, and in 1981 he was awarded a MacArthur Foundation Prize Fellowship.

DOUGLAS B. LENAT ("Computer Software for Intelligent Systems") is a specialist in artificial intelligence who holds an assistant professorship in computer science at Stanford University. His B.A. and M.S. degrees are from the University of Pennsylvania; in 1976 he obtained his Ph.D. in computer science from Stanford. In his doctoral thesis he showed that a computer can be programmed to propose original mathematical theorems. Since then he has pursued an inquiry into the nature of heuristic reasoning. He taught for a year at Carnegie-Mellon University before taking up his present post. Lenat is associated with a number of corporations, organizations and journals specializing in artificial intelligence.

TAKE THE 3-VOLUME HANDBOOK OF ARTIFICIAL INTELLIGENCE (A \$141.95 VALUE) FOR ONLY \$4.95

when you join the Library of Computer and Information Sciences.

You simply agree to buy 3 more books—at handsome discounts—within the next 12 months.

Just completed, the massive, 3-volume HANDBOOK OF ARTIFICIAL INTELLIGENCE promises to become the standard reference work in the growing AI field.

Conceived and produced by leading scientists and researchers at Stanford University, with contributions from universities and laboratories across the nation, the *Handbook* makes available to scientists, engineers, students, and hobbyists who are encountering AI for the first time the techniques and concepts in this rapidly expanding computer universe.

The 200 articles cover the emerging issues, technical problems, and design strategies which have been developed during the past 25 years of research. The *Handbook* has been written for people with no background in AI; jargon has been eliminated; and, the hierarchical organization of the book allows the reader to delve deeply into a particular subject or browse the articles which serve as overviews of the various subfields.

The 15 chapters (5 per volume) include: the history, goals, and current areas of research activity; the key concept of “search”; research on “natural languages”; the design of programs that understand spoken language; applications-oriented AI

The comprehensive HANDBOOK OF ARTIFICIAL INTELLIGENCE answers questions like:

- What is a “heuristic problem-solving program?”
- How do computers understand English?
- Can computer programs outperform human experts?

AND INCLUDES:

- Over 1,450 pages.
- More than 200 articles in 15 chapters.
- With numerous charts, tables, and schematics.
- Edited by Avron Barr, Paul Cohen and Edward Feigenbaum.

research in science, medicine, and education; automatic programming; models of cognition; automatic deduction; vision and learning research; and, planning and problem solving.

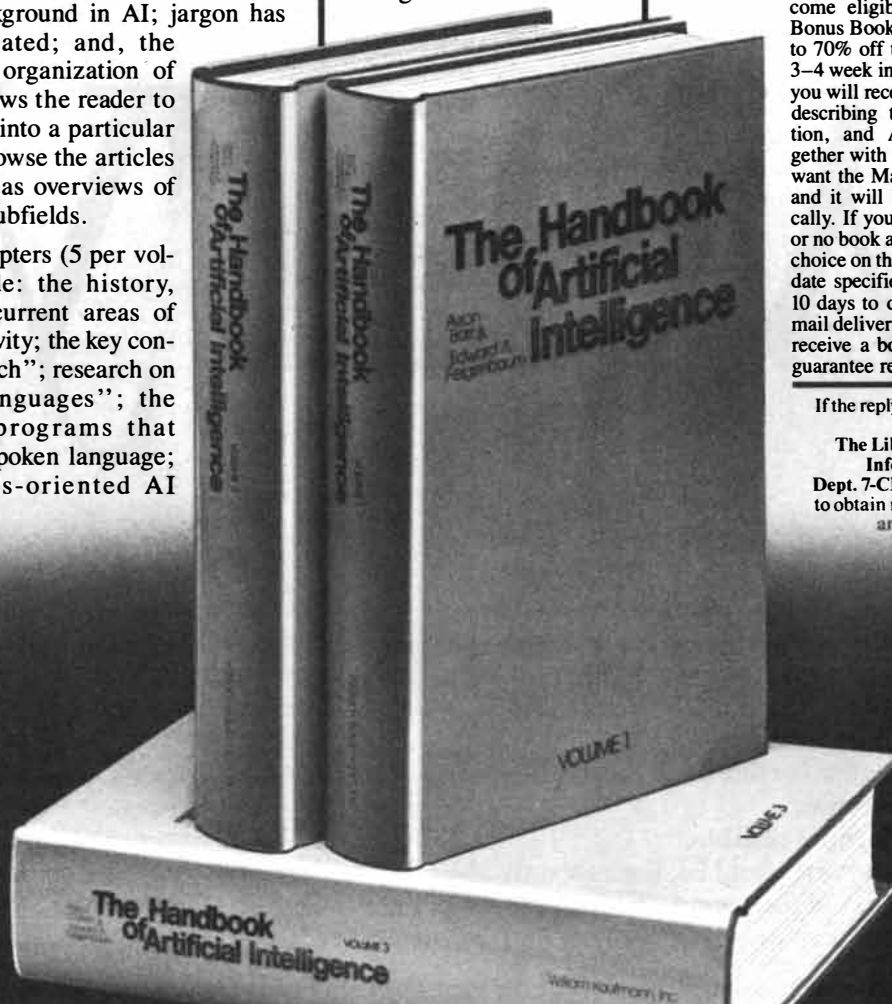
The Library of Computer and Information Sciences is the oldest and largest book club especially designed for the computer professional. In the incredibly fast-moving world of data processing, where up-to-date knowledge is essential, we make it easy for you to keep totally informed on all areas of the information sciences.

Begin enjoying the club's benefits today!

MEMBERSHIP BENEFITS: In addition to getting the 3-volume Handbook of Artificial Intelligence for only \$4.95, when you join, you keep saving substantially on the books you buy. Also, you will immediately become eligible to participate in our Bonus Book Plan, with savings of up to 70% off the publishers' prices. At 3–4 week intervals (16 times per year) you will receive the Book Club News, describing the coming Main Selection, and Alternate Selections, together with a dated reply card. If you want the Main Selection, do nothing and it will be sent to you automatically. If you prefer another selection, or no book at all, simply indicate your choice on the card, and return it by the date specified. You will have at least 10 days to decide. If, because of late mail delivery of the News, you should receive a book you do not want, we guarantee return postage.

If the reply card has been removed, please write to:

The Library of Computer and Information Sciences
Dept. 7-CE2, Riverside, N.J. 08075
to obtain membership information and an application.



Scientific American 9/84

3M introduces one less worry a

We can't do anything about your taxes. Or the rush-hour traffic. Or the person who keeps stealing lunches out of the office refrigerator.

But we can take a big load off your mind when it comes to diskettes.

3M diskettes are certified 100% error-free. And guaranteed for life.

No floppy is more reliable.

There's no way one could be. Because only 3M controls every aspect of the manufacturing process.

We make our own magnetic oxides. And

the binders that attach them to the dimensionally stable substrate. Which we make ourselves from liquid polyester. Which we make ourselves.

We also test our floppies. At least 327 ways. And not just on exotic lab equipment with perfectly aligned, spotless heads. But also on office equipment like yours.

We even reject a floppy if its label is crooked.

Some people think we're a little crazy to go to all that trouble. After all, do you really need a diskette that can make one read/write

roduces
thing to
bout.



3M
diskettes

pass on every track, every hour, every day for
the next 200 years?

Not really.

But now that you know a 3M floppy can
do it, you can relax.

And worry about other things.

Like who stole your lunch from the office
refrigerator.

One less thing to
worry about.

COMPUTER RECREATIONS

The failings of a digital eye suggest there can be no sight without insight

by A. K. Dewdney

Imagine a black box rather like a camera. At the front is a lens and on one side is a dial with various settings such as "Tree," "House," "Cat" and so on. With the dial set to "Cat" we go for a walk and presently encounter a cat sitting on a neighbor's porch. When the box is aimed at the cat, a red light goes on. When the box is aimed at anything else, the light remains dark.

Inside the box is a digital retina sending impulses to a two-layer logical network: an instance of the device called a perceptron. At one time it was hoped that perceptrons would ultimately be capable of real-world recognition tasks like the one described in the fantasy above. But something went wrong.

The 1950's and 1960's were years of tremendous creativity and experimentation in the newly developing field of computer science. Romantic paradigms such as self-organizing systems, learning machines and intelligent computers influenced many scientists, and I am tempted to call the period the Cybernetic Age. Incredible machines that could see or think or even reproduce themselves seemed just around the corner. The simplest of these machines was the perceptron.

A perceptron consists of a finite grid-like retina subdivided into cells that receive light. Like certain cells in the human retina, each perceptron cell turns on if it receives enough light; otherwise it stays off. It is therefore reasonable to think of the image a perceptron analyzes as a grid of light and dark squares, as in the illustration on page 27.

Besides the retina, a perceptron consists of a great many primitive decision-making elements I shall call local demons. Each local demon examines a fixed subset of the retinal cells and reports on conditions there to a more complex decision maker I may as well call the head demon. Specifically, each local demon is equipped with a notebook listing certain patterns it must watch for in its locale, the subset of retinal cells under its jurisdiction. If any of the listed

patterns appears, the local demon sends a signal to the head demon; otherwise it remains silent. The head demon's job is more complicated in that it must do some arithmetic. Each signal from a local demon is multiplied by a specific positive or negative integer (the local demon's assigned "weight") and the resulting numbers are added. If the sum is at least as great as a fixed threshold, the head demon says yes; otherwise it says no. To avoid making any assumptions about what the various demons look like I have shown them in the illustration on page 27 as boxes.

Demons are often given dangerous or even impossible jobs such as opening tiny doors in the wall of a container to let molecules pass through. In comparison, the demons of the perceptron have quite easy jobs. Indeed, the local demons could be replaced by simple logic circuits, and the head demon's job could easily be done by a few registers, an adder and a comparator (elements of the central-processing unit of any computer). Demons, however, have a romantic charm that electronic devices cannot match.

A perceptron's job is to say yes when certain patterns are presented to it and to say no to all others. The former patterns are said to be recognized by the perceptron. Although it is highly doubtful that a cat-recognizing perceptron could ever be built, other recognition tasks are attainable.

A perceptron can be programmed, after a fashion, to recognize a given class of patterns by adjusting the weights and the threshold. Local demons supplying evidence in favor of the class are weighted positively and those providing evidence against it are weighted negatively. The magnitude of each weight reflects the value or importance of the evidence. Although the perceptrons discussed here operate with a fixed set of weights, the notion of programming plays a central role in the theory of perceptrons developed in the 1950's.

The following perceptron recognizes a dark rectangle of any size or shape

placed anywhere on its retina. In fact, it recognizes any number of such rectangles (including zero), provided no two of them touch along a side or at a corner. There are three steps in the construction of the perceptron. First, install a local demon at each 2×2 locale in the retina. Then put all the subpatterns in list P [see top illustration on page 29] on each local demon's list. Third, set all the head demon's weights to $+1$ and set the threshold to d , the number of local demons.

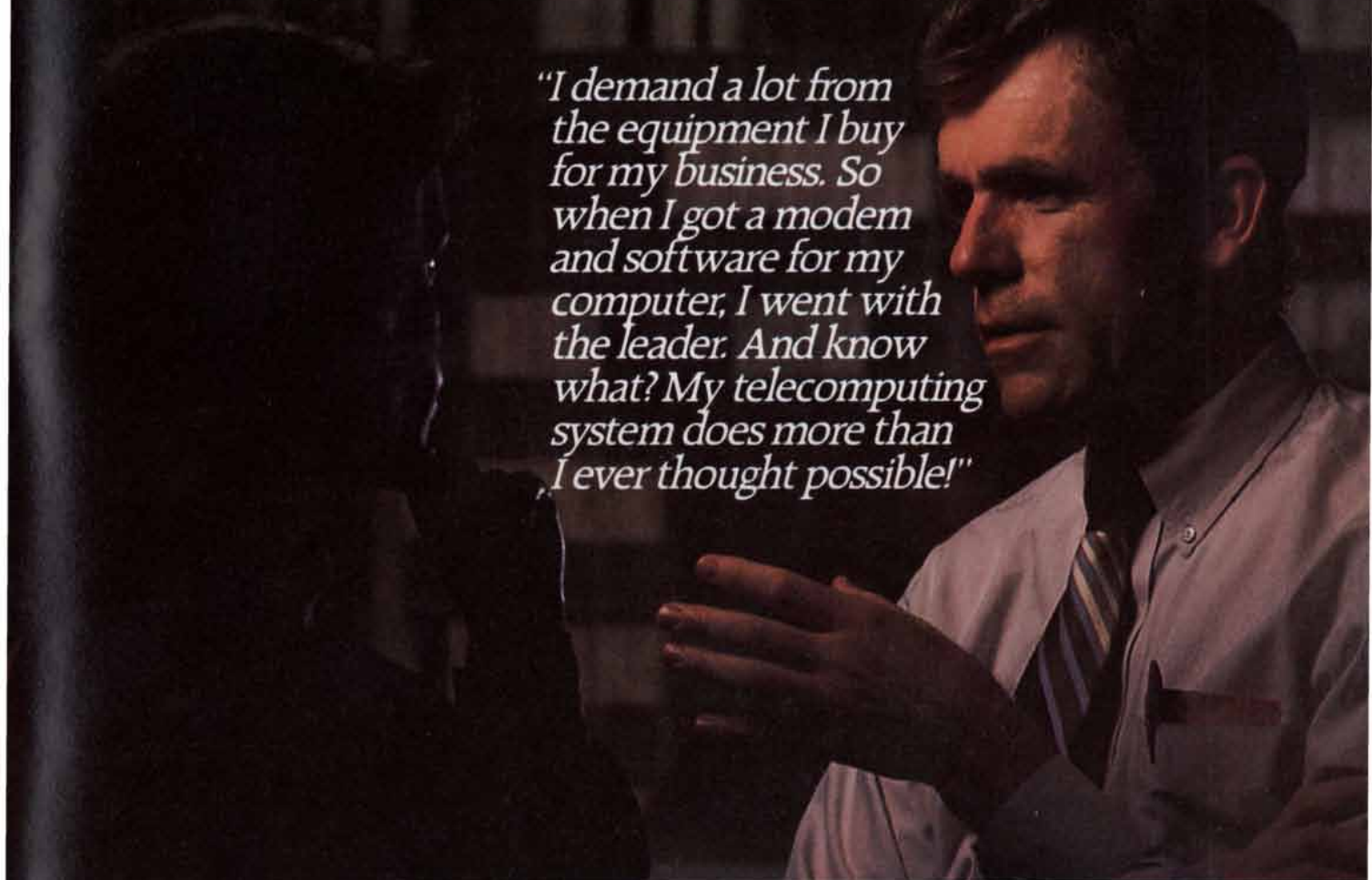
This design calls for quite a few demons: if the perceptron has an $n \times n$ retina, there will be $(n-1)^2$ local demons. They are all given positive weights, indicating they all supply positive evidence toward the recognition of rectangles. For example, it is not hard to see that when a single rectangle is projected onto the perceptron's retina, each 2×2 set of cells must contain one of the subpatterns in list P . It follows that every local demon sends a signal to the head demon and the weighted sum of the signals is, of course, d . The head demon says yes. On the other hand, if one of the dark shapes is not a rectangle or if two rectangles touch, then at least one of the 2×2 sets contains a subpattern from list N in the top illustration on page 29. Hence at least one of the local demons fails to report and the head demon develops a sum no greater than $d-1$. It says no.

An equivalent perceptron could be designed in which each local demon uses the smaller list N . In this case all the weights would be -1 and the threshold would be zero. Each local demon would supply negative evidence toward a pattern of rectangles and the head demon would say yes only if none of the local demons sent it a signal.

The style of perceptron defined above has many interesting properties, and it seems worthwhile to give it a name. Without specifying what list of subpatterns all the local demons use, a device of this kind will be called a window perceptron because each local demon looks at the input pattern through a 2×2 window. For an $n \times n$ retina there are $(n-1)^2$ demons, and the threshold is equal to this number.

Generally speaking, perceptrons seem to be best at recognizing geometric figures. Window perceptrons can recognize not only rectangles but also "black holes" (isolated dark cells), vertical and horizontal lines, stairways, checkerboards and many other patterns. It all depends on what set of 2×2 subpatterns is chosen for the local demon lists [see illustration on page 34]. Indeed, each subset of the 16 possible 2×2 subpatterns defines a different window perceptron, and each of the resulting 65,536 window perceptrons recognizes a certain class of patterns. Or does it?

The window perceptron based on the



"I demand a lot from the equipment I buy for my business. So when I got a modem and software for my computer, I went with the leader. And know what? My telecomputing system does more than I ever thought possible!"

Hayes. Leading the way with quality telecomputing systems for the personal computers that businesses use most.

When it comes to communicating—computer to computer—Hayes says it best. Let a Hayes telecomputing system handle your communications. Instantly. Accurately. Economically.

All you need is a Hayes Smartmodem (it's like a telephone for your computer) and Smartcom II* software. In no time at all you can create, send and store files, and automatically log on to information services. The communication possibilities are endless!

Introducing our new Smartcom II. More connection capabilities. More convenience.

Compatibility. Now Smartcom II is available for more than 16 personal computers (with more to come). That means you can communicate, Smartcom to Smartcom, with an IBM PC, DEC Rainbow 100, HP 150, TI Professional Computer* and many others.

Two popular protocols. In addition to the Hayes Verification protocol, our new Smartcom II includes the XMODEM protocol, for error-free transmission to a wide range of personal computers and mainframes at information services.

Terminal emulation. Smartcom II also allows your computer to "emulate" the DEC VT100 and VT52 terminals, opening the door to a vast number of DEC minicomputers.

Voice to data communications. With Smartcom II you can easily switch from voice to data transmission (and back again), all in the same phone call. This saves you time and money, since you don't have to hang up and dial again.

Unattended operation. Smartcom II makes telecomputing simple, even when you're not there. It can take a message when you're out, and leave it on your disk or printer. And you can tell Smartcom II to "save" the messages you've created during the day, and automatically send them at night, when phone rates are lowest.

Get your hands on the leader. Hayes Smartmodem.

With an unsurpassed record of reliability, it's a small wonder Smartmodem is such a smart buy! Smartmodem 300™

(the first of the Smartmodem series), transmits and receives data at up to 300 bps. For longer distances and greater volume, Smartmodem 1200™ and Smartmodem 1200B™ (it plugs into an expansion slot inside an IBM PC or compatible) provide high-speed, high-performance superiority.

Visit your computer dealer for a hands-on demonstration of Smartmodem and Smartcom II. A complete, reliable telecomputing system for your personal computer.



Hayes

Hayes Microcomputer Products, Inc.,
5923 Peachtree Industrial Blvd.,
Norcross, Georgia 30092. 404/441-1617.



Nile dial: A few years back, faced with explosive urban growth and an inadequate telephone system, the Egyptian government engaged Continental Telecom Inc. to redesign its communications system. A massive research and planning program to achieve this

metamorphosis was initiated and carried out by Contel Page, Contel's engineering and construction subsidiary. Today, one of the world's most up-to-the-minute communications systems is becoming a reality in one of the world's most ancient hubs. From telephony to satellites. Architects of telecommunication.

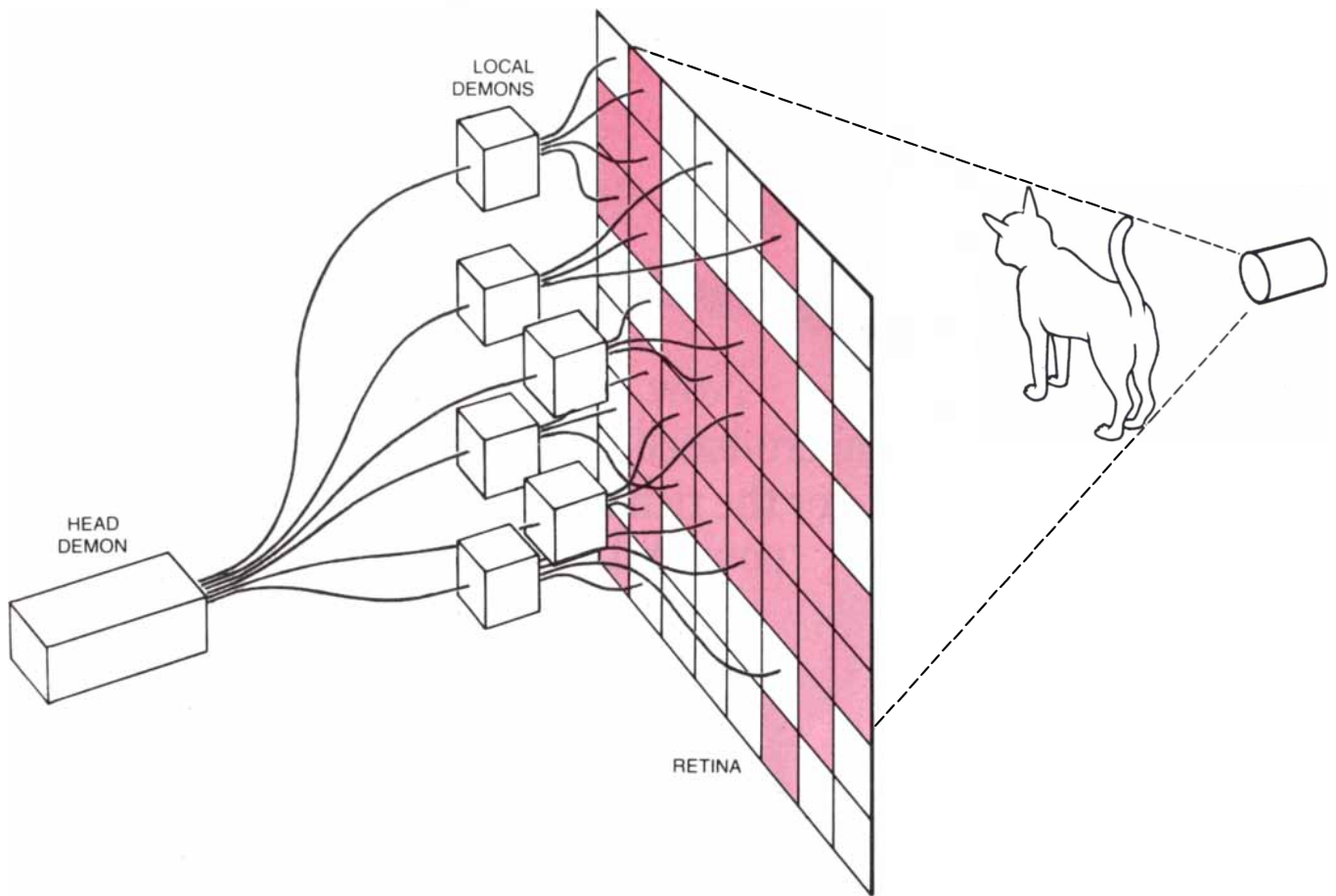
WRITE CONTEL, DEPT. 504, 245 PERIMETER CENTER PKWY, ATLANTA GA 30346 © 1983 CONTINENTAL TELECOM, INC.

CONTEL

In 1640, moments before Scotland's
Caerlaverock castle fell, the Lady made her escape.
And every twelvemonth since,
on the very eve,
she returns.
Or so they say.
The good things in
life stay that way.

DEWAR'S
"White Label"
never varies.
Authentic
The Dewar Signature





A perceptron attempts to recognize a cat

two subpatterns below does not recognize anything.



The reason is simple. Assuming a fairly large retina, select a 2×2 window somewhere in the middle of it. If the window contains the first of the two subpatterns above, examine the window one cell to the right: it will have a dark cell in its upper left corner, and so the demon in charge of that locale will send no signal to the head demon. Remember that in a window perceptron *all* the local demons must report in for a pattern to be recognized. If the second subpattern is present, shifting the window one cell to the left yields a similar contradiction.

Which subsets of the 16 subpatterns give rise to window perceptrons that actually recognize something? The question is probably hard to answer, but it illustrates very well the kind of question an interested computer scientist or mathematician might ask when confronted by the phenomenon of a perceptron that recognizes nothing. Because of the large number of such perceptrons, the answer would best take the form of some easily applied criterion or test: giv-

ing a subset of 2×2 subpatterns, one applies the test and obtains an answer for that particular subset.

The point of these remarks is that professional standing as a computer theorist is not always needed to answer such questions. Although they go somewhat beyond the kind of puzzle commonly given in recreational-mathematics columns, they call for the same kind of thinking. Readers who have solved at least one of Martin Gardner's puzzles in the "Mathematical Games" department of *Scientific American* should be able to make some progress with the question above. In theoretical research, as in experimental science, a partial answer is better than no answer at all.

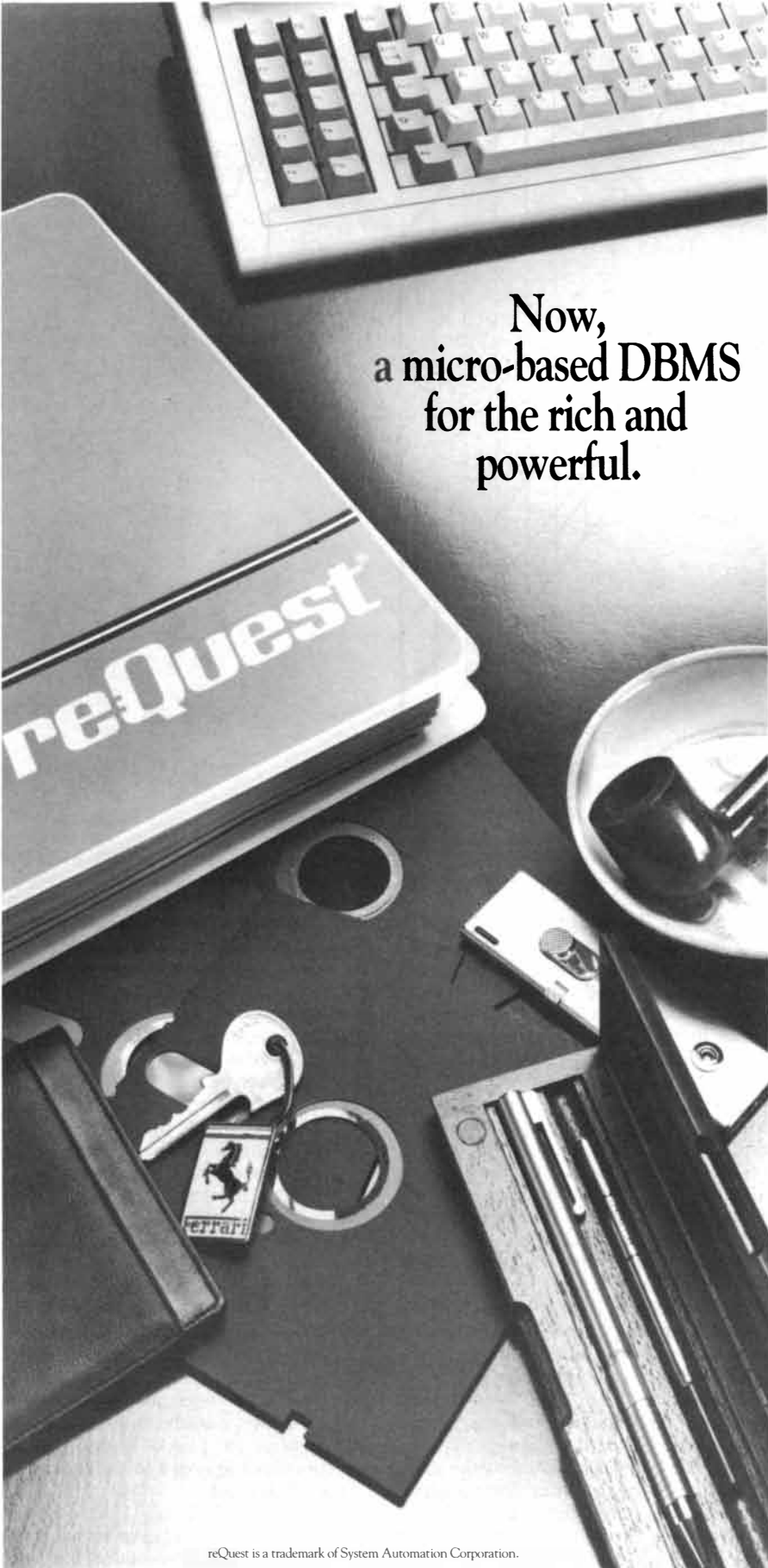
Work in perceptrons was pioneered by Frank Rosenblatt of Cornell University in the 1950's. Rosenblatt and his co-workers, both at Cornell and elsewhere, became optimistic about the prospects for perceptrons as useful pattern-recognition devices. The "convergence theorem" told them that in principle perceptrons could learn to recognize patterns by making the weights used by the head demon subject to automatic control. The theorem states that any adjustment of weights in the direction of improved powers of recognition can serve as the basis of still further improvements. Actual perceptrons were built, and in some

tests on simple patterns they achieved high recognition scores.

What seemed encouraging progress at the time was, in a sense, illusory. According to Marvin L. Minsky and Seymour Papert of the Massachusetts Institute of Technology, the enthusiasts for perceptrons had been beguiled by the simplicity and apparent success of their devices. Below the surface lay some grave defects in the concept. In 1969 Minsky and Papert issued *Perceptrons*, a book that effectively punctured the balloon by pointing out (and proving) several things perceptrons cannot do.

One of the most dramatic failures discovered by Minsky and Papert was the inability of certain perceptrons to recognize when a figure is connected (that is, all in one piece). Assuming each local demon inspects only a limited locale, Minsky and Papert gave examples of four patterns designed so that one of them always stumps a perceptron whose job is to recognize connectivity. The patterns are shown in the bottom illustration on page 29. Two of them (*b* and *c*) are connected figures and the other two (*a* and *d*) are not.

Suppose someone claims to have designed a diameter-limited perceptron capable of distinguishing between connected and unconnected patterns. By



Now,
a micro-based DBMS
for the rich and
powerful.

There are well over a hundred bundles of ones and zeroes called Data Base Management Software.

Sometimes it seems that there's an answer for every hacker, a system for every machine.

And there's the rub.

If you have a data base, and more than one user, and more than one kind of desktop micro, then you might well envy the meek.

They have their answers. Where are yours?

We submit reQuest: an easy-to-use DBMS for people who really need one.

reQuest: the answer to corporate multiplication.

Look around at the desktop micros. You'll see IBM PC, and its compatibles: NCR. Burroughs. HP. Gould. Datapoint.

Look at the people. They come from every discipline, every specialty, every level of computer awareness.

It's a jumble out there. How can you accommodate all these people, at all these levels, with all these pieces of hardware, using so many operating systems (MS-DOS, PC-DOS, CT-OS)? Answer: reQuest. It can run any of these systems. And network within each system.

Nobody else can do all that.

reQuest: Not off-the-shelf; on the line. Guaranteed.

Yes, you can buy a DBMS off the shelf. You get a disk, a book, and the best wishes of the people who sold their software to you and their company to somebody else.

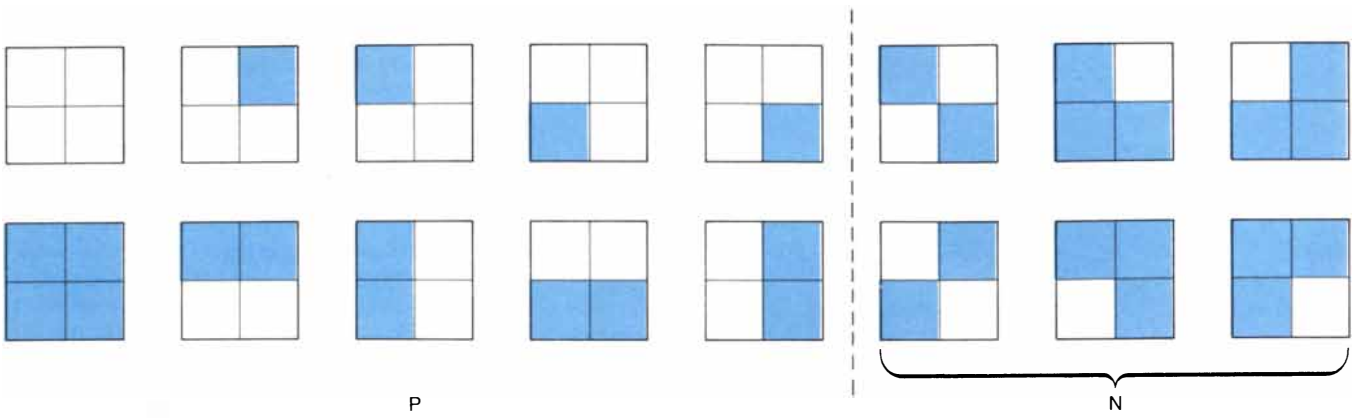
But after 17 years in the business, we can promise you this: When you buy reQuest, you buy us. All of us. All of the time. For training. For re-training. And phone rights. And visitation rights. And, if all that doesn't work, your money back.

Nobody else does all that, either. If you don't believe it, ask your data base. If it doesn't answer, call 301-565-9400.

reQuest®

System Automation
Corporation, Inc.
8555 Sixteenth Street
Silver Spring, MD 20910

reQuest is a trademark of System Automation Corporation.



The 2×2 subpatterns recognized by positive (P) and negative (N) local demons

diameter-limited I mean that for some number m every local demon can examine only the squares within an $m \times m$ window. To test the claim Minsky and Papert would prepare versions of their four patterns, with each pattern adjusted to be more than m cells long. The local demons can then be classified in three disjoint sets. Left Demons examine at least one cell on the left edge of the figure. Right Demons examine at least one cell on the right edge. Other Demons are neither Left nor Right.

When the proposed connectivity perceptron is presented with pattern a , it either fails (by saying yes) or succeeds (by saying no). If it fails, of course, the test is over. If it succeeds, the next step is to examine the sum developed by the head demon and split it into three parts: L , O and R , representing the weighted sums of the Left, Other and Right de-

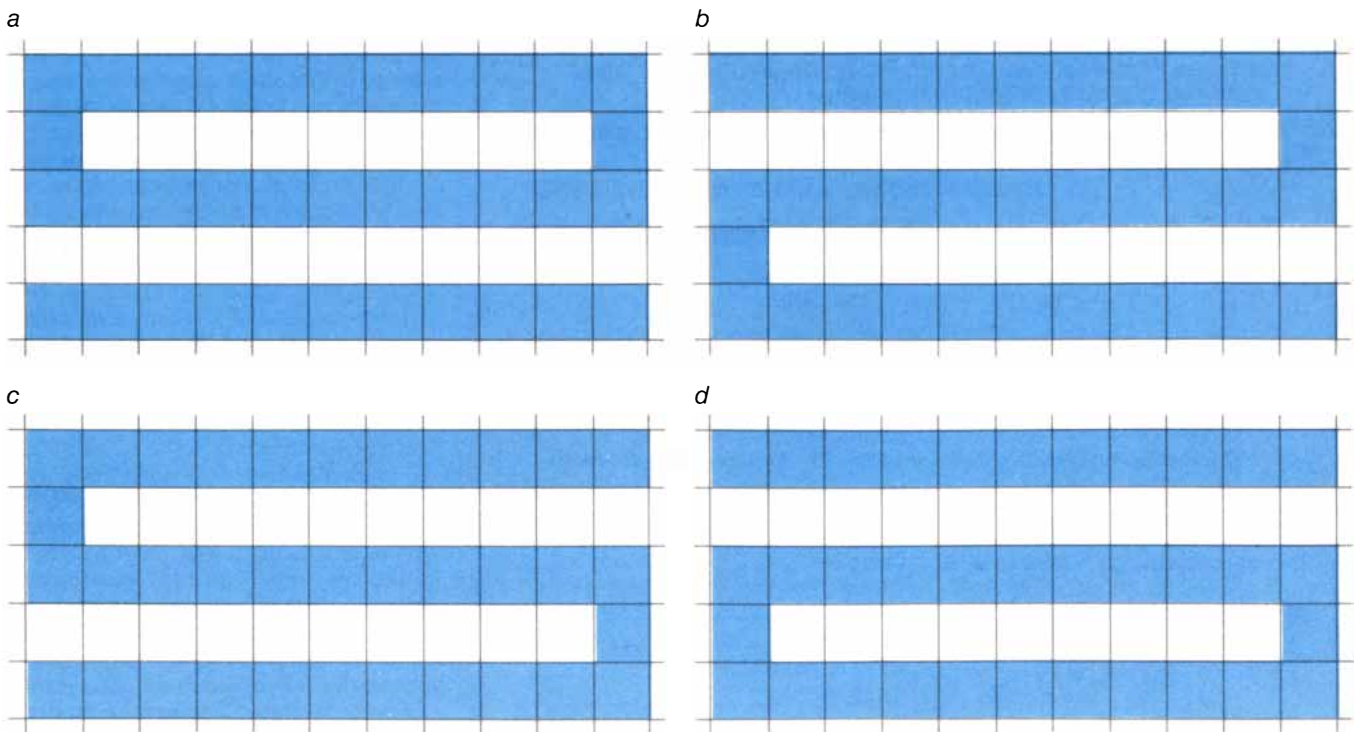
mons that signal the head demon when pattern a is projected on the retina. Since the connectivity perceptron says no, the sum of L , O and R falls short of the threshold. If pattern a is now replaced by pattern b , only the Left Demons change their response, since only the cells along the left edge of the figure change. Suppose the partial sum L becomes L' . On the other hand, if pattern a is replaced by pattern c , only the cells along the right edge change and only the Right Demons change their response, say from R to R' .

Now the perceptron has got itself into a most curious position. Since b and c are connected, it must answer yes in both cases, and so the sums $L' + O + R$ and $L + O + R'$ must be at least as large as the threshold. It is already known, however, that $L + O + R$ is less than the threshold because a was not connected.

It follows that L' is larger than L and R' is larger than R . The deathblow comes when the perceptron faces pattern d . Here both the left- and the right-hand cells of the figure have changed from the state they had in pattern a , and the head demon finds itself computing the sum $L' + O + R'$, which is certainly greater than the threshold. The head demon says yes. It is wrong.

Additional failings of perceptrons discovered by Minsky and Papert include the unrealistically large number of local demons needed for some recognition tasks and the low rate of learning (or convergence) for other tasks.

Perhaps it is not surprising that perceptrons should fail in many cases where the human visual system succeeds. I noted above that the local demons and head demon could be replaced



Four figures designed to confuse a connectivity perceptron

Announcing . . .



\$49.95

**Fine Tuned For Scientific, Business
and Engineering Applications**

(And perfect for the Weekend Programmer too)

"Finally, somebody has done it right. A powerful Pascal Z80 or 8086/88 single pass native code compiler together with a full-screen editor and error checking to make a super programming development package."

Dave Carroll, *Microsystems*, February 1984

- Automatic overlays (no addresses or memory space to calculate)
- Optional 8087 support (available for an additional charge)
- Full-screen, interactive editor
- Windowing (for IBM PC or Jr.)
- Full support of operating system facilities
- Full heap management—via dispose procedure
- Graphics, sound and color support for IBM PC or Jr.
- Occupies only 35K including editor!!!
- Extended Pascal for your IBM PC, PC Jr., Apple CP/M, MS DOS, CP/M 86, CCP/M 86 or CP/M 80 computer.

"It is, simply put, the best software deal to come along in a long time . . . buy it."

Bruce Webster, *Softalk IBM*, March 1984

To order your copy of Turbo Pascal 2.0 call:

1-800-255-8008

in CA 1-800-742-1133

Dealer and Distributor Inquiries welcome

408-438-8400

CHOOSE ONE:

- Turbo Pascal version 2.0 \$49.95 + \$5.00 shipping per copy.
- Turbo Pascal 2.0 with 8087 support \$89.95 + \$5.00 shipping per copy.

Check _____ Money Order _____
 VISA _____ MasterCard _____
 Card #: _____
 Exp. date: _____ Shipped UPS

My system is: 8 bit _____ 16 bit _____
 Operating System: CP/M 80 _____
 CP/M 86 _____ MS DOS _____ PC DOS _____
 Computer: _____ Disk Format: _____
 Please be sure model number & format are correct.

NAME: _____
ADDRESS: _____
CITY/STATE/ZIP: _____
TELEPHONE: _____

California residents add 6% sales tax. Outside U.S.A. add \$15.00. (If outside of U.S.A., payment must be by bank draft payable in the U.S. and in U.S. dollars.) Sorry, no C.O.D. or Purchase Orders. E3



Borland International
 4113 Scotts Valley Drive
 Scotts Valley, California 95066
 TELEX: 172373

by simple computational circuits. They could also be replaced by the formal neurons first described in the 1940's by Warren S. McCulloch and Walter H. Pitts in their classic work on neural networks. These formal neurons are much simpler than human neurons; likewise the complexity of a perceptron organized as a two-layer neural network does not come close to the complexity of the first two layers of the human visual cortex. Moreover, "behind" the visual cortex, as it were, there is an amazing and almost completely unknown analytic apparatus—something that is entirely lacking in the perceptron model of vision. To even begin modeling this greater complexity one would have to replace the head demon by a Turing machine, but here the argument sinks into a sea of uninformed speculation, and so I shall call a halt.

Even if perceptrons are eyes without minds, they have a certain charming simplicity and, for some patterns at least, definite powers of recognition. I wonder what other patterns readers might discover to be within the competence of the window perceptron. Those wishing to explore the tougher question of which subsets of the $16 \times 2 \times 2$ subpatterns lead to "good" window perceptrons (those that recognize at least one pattern) will find the question somewhat cleaner to handle if a constraint is added: the patterns recognized should be "translatable"—it should be possible to shift them on the retina without changing the fact that they are recognized. This requirement not only rules out certain overspecialized window perceptrons (for example the one that recognizes a single dark cell in the upper right corner of its retina) but also reflects the notion that the perceptron is looking at a real scene that shifts across the retina as the black box in my fantasy scans it.

Although diameter-limited perceptrons are unable to distinguish connected figures from unconnected ones, it may be possible to recognize connectivity in certain classes of figures. For example, within the class of all multiple-rectangle patterns the connected figures would be those that include exactly one rectangle. Can you design a perceptron that recognizes just such patterns? Your local demons must use 2×2 windows, but you may hire additional demons if necessary.

I implied above that perceptron research came to an end with the publication of Minsky and Papert's *Perceptrons*. This is true in the sense that a certain woolly and wishful attitude toward perceptrons and their powers of recognition is no longer possible. On the other hand, it was far from Minsky and Papert's intention that all research in perceptron theory be stopped. The precise powers of these simple but sometimes effective devices have yet to be discovered.

Rosenblatt, whose work extended into psychology and neurobiology, died in a tragic boating accident on his 43rd birthday, July 11, 1971, in Maryland.

Reader response to the column on analog gadgets was gratifying, with no fewer than 17 new gadgets being suggested. Three correct solutions to the light-in-a-mirrored-box problem were also submitted.

Before taking up these matters, however, I must correct an error. To my knowledge the fastest digital-computer algorithm for finding the convex hull of a set of points in the plane requires on the order of $n \log n$ operations, not $n \log \log n$. The string analog gadget that solves the same problem was invented in 1957 by George J. Minty, Jr., of Indiana University. Minty also points out that the soap-film technique for finding minimum Steiner trees originated with William Wiehle in 1958.

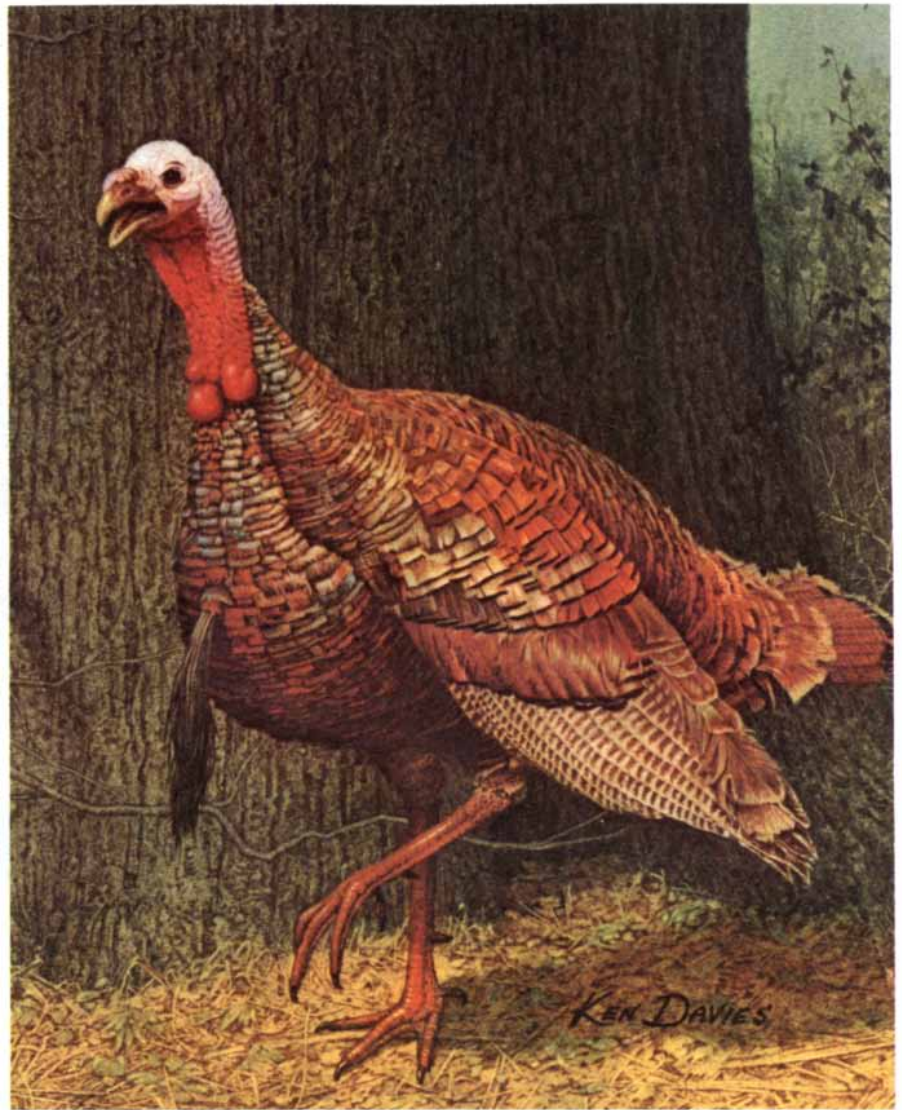
The laser gadget for discovering whether a number n is prime was criticized by David Zimmerman of Beaver Dam, Wis. The light must be reflected n times in going from the laser to the detector, he notes, and since the speed of light is finite, the solution time is proportional to n . If the problem size is defined as the number of digits in n , the solution time grows exponentially and the device is no faster than a digital algorithm.

The finite speed of light also bothered Steven P. Hendrix of New Braunfels, Tex., who remarked that in the mirrored-box problem one might have to wait a very long time for the light to emerge. I asked what property of the light path the mirrored box measures. Hendrix was among those who solved the problem by noting that the question of whether the light emerges is equivalent to the question of whether an infinite straight line in the plane intersects a point with integer coordinates.

Imagine an infinite orchard with infinitely thin trees planted on a square grid. If a bullet is fired from one tree in an arbitrary direction, will it ever strike another tree? It will if the angle with respect to the rows of trees has a rational slope. If the tree struck is p rows north and q rows east of the firing point, the slope is p/q . The mirrors in the box merely fold up the path of the bullet. John Dewey Jones of Farmington Hills, Mich., and Paul Kingsberg of Imperial, Pa., also solved the problem.

The only way to do justice to the wealth of gadgets described by readers is to devote a second column to the subject, probably early in 1985. Meanwhile I shall at least mention some of the more interesting gadgets.

Peter F. Ash of St. Joseph's University reported on solving a cubic equation by immersing solids in a tank of water. Tom Digby of Los Angeles remarked that the computational power of an ana-

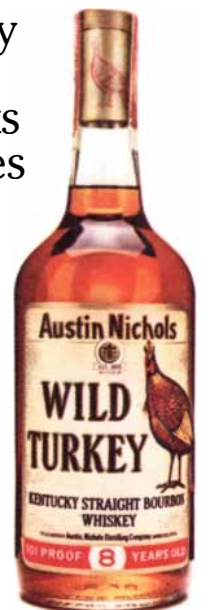


For personally signed Ken Davies print, 18" x 19", send \$10, payable to "ANCO", Box 2832-SN, NYC, 10163

Always On The Move

The Wild Turkey instinctively seeks "elbow room." If the bird senses any encroachment on its territory, it will travel many miles a day in search of a remote swamp or forest preserve.

Native only to the American continent, the Wild Turkey is a fitting symbol for America's greatest native whiskey—
Wild Turkey.



WILD TURKEY®/101 PROOF/8 YEARS OLD
AUSTIN, NICHOLS DISTILLING CO., LAWRENCEBURG, KENTUCKY © 1982

**Better
productivity
grows
on the
Peachtree.[®]**

And it's no surprise.

Only one software company so completely covers business, home and education.

Only one is backed by the resources of the world's largest independent producer of mainframe software, Management Science America, Inc. (MSA).

And only one offers you the most complete line for microcomputers. Peachtree Software®. The pioneer that's still pioneering. With 25 new products this season, 67 in all. And with applications for IBM, Apple and most leading microcomputers.

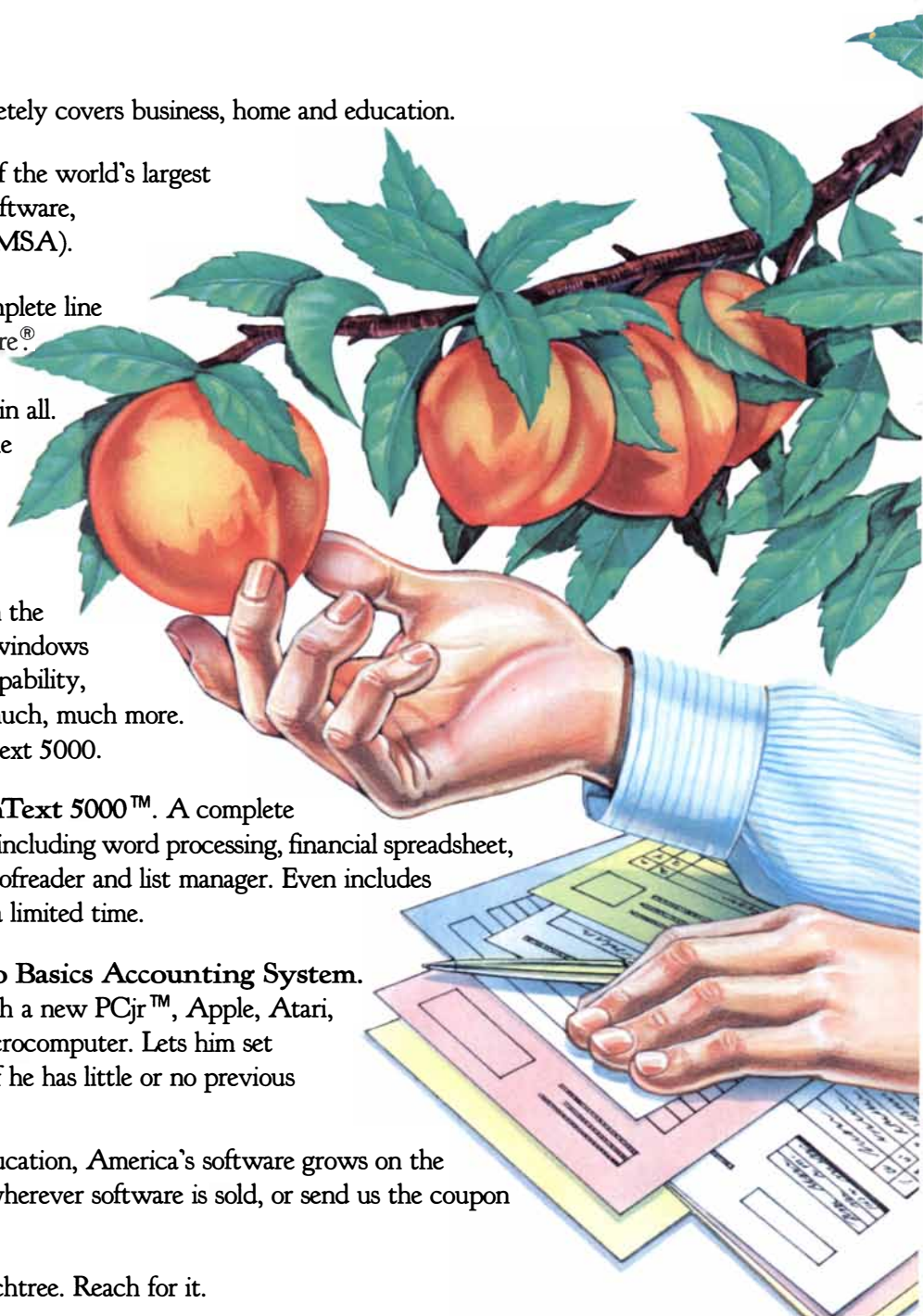
Case in point: our new Decision Manager. Created to give executives higher productivity on the job than ever before. With up to ten windows displayed at once, dramatic graphic capability, mainframe communications link and much, much more. The perfect companion to our PeachText 5000.

Case in point: our famous PeachText 5000™. A complete package for better office productivity, including word processing, financial spreadsheet, Random House thesaurus, spelling proofreader and list manager. Even includes free special instructional software for a limited time.

Case in point: our new Back To Basics Accounting System. Created for the small businessman with a new PCjr™, Apple, Atari, Commodore or almost any leading microcomputer. Lets him set up his basic accounting system even if he has little or no previous accounting knowledge.

Whether it's for business, home or education, America's software grows on the Peachtree. Look for all our products wherever software is sold, or send us the coupon for a free brochure.

Better productivity grows on the Peachtree. Reach for it.



America's Software grows on the Peachtree.

PCjr is a trademark of International Business Machines Corporation.

Peachtree and Peachtree Software are registered trademarks of Peachtree Software Incorporated, an MSA Company. © 1984.



Peachtree Software

3445 Peachtree Road N.E.
Atlanta, GA 30326 • 1-800-554-8900

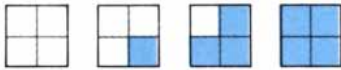
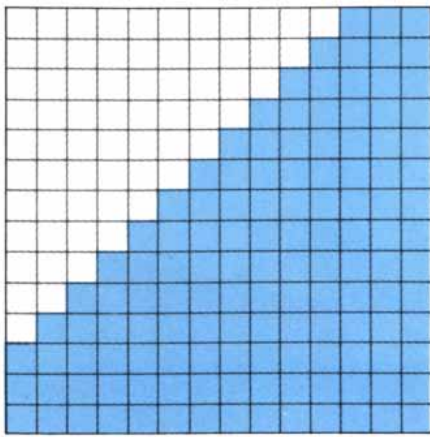
Please send me more information about Peachtree Software.
I'm interested in software for () business, () home, () education.

Name _____

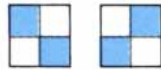
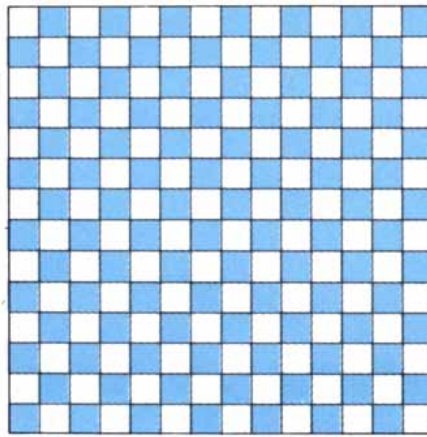
Address _____

City _____ State _____ Zip _____

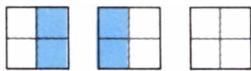
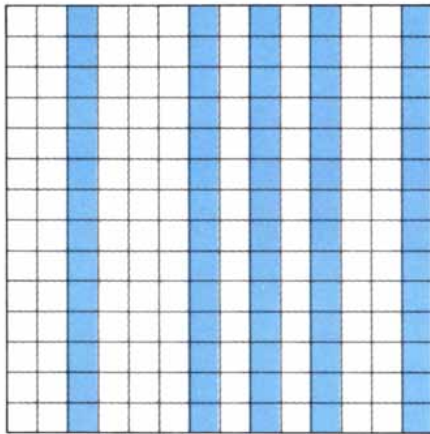
Phone _____ DC 09014AB



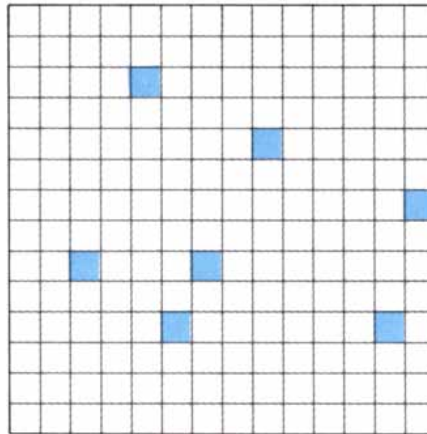
STAIRS



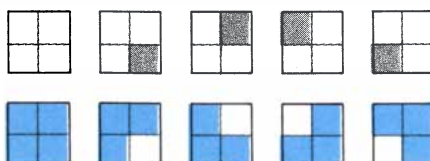
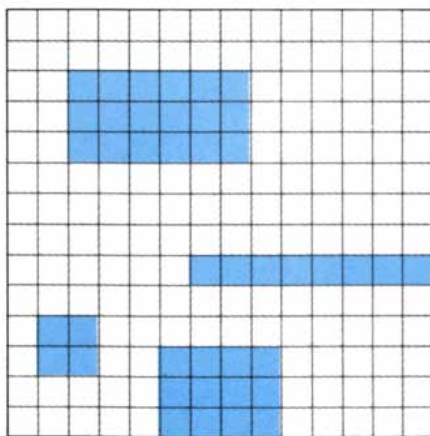
CHECKERBOARD



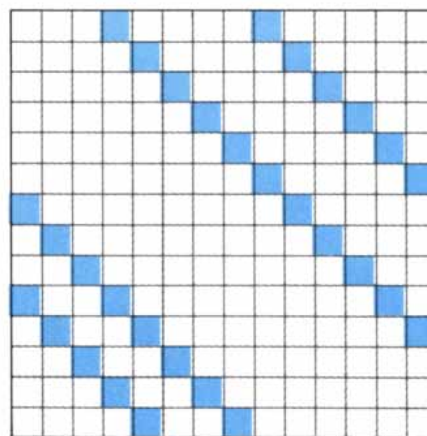
VERTICAL LINES



"BLACK HOLES"



MULTIPLE RECTANGLES



DIAGONAL LINES

log gadget can be attributed to its ability to carry out many processes in parallel. He showed how to set up n digital computers to sort n numbers in linear time, equaling the performance of the spaghetti analog gadget.

Eric Halsey of the University of Washington described a longest-path gadget made out of "snakes." Each edge of the graph is represented by an elastic string threaded through an integer number of beads. Does the longest path stand out when the gadget is stretched and then released? Another of his gadgets measures the length of the shortest path between two vertices in a graph. Make each edge a piece of fuse and put a firecracker at the second vertex. Now light the fuses at the first vertex and stand back: the time until the firecracker explodes is proportional to the length of the shortest path.

I was reminded by Palmer O. Hansen, Jr., of Largo, Fla., that the planimeter, a mechanical device for measuring area, could qualify as an analog gadget. Dale T. Hoffman of Bellevue Community College in Washington pointed out some additional problems that can be solved by soap films, including a clever computation of Snell's law. David Kimball of San Diego solves mazes by pumping water into the maze and following the current to the exit. Another pretty gadget was described by J. H. Lueth of the United States Metals Refining Company in Carteret, N.J. SLAG (the smelter-location analog gadget) finds the location for a smelter that minimizes transportation costs for limestone, coal and ore. Three holes in a board and three weights tied together with string solve the problem. The same device was also mentioned by Hendrix.

Tony Mansfield of the British National Physical Laboratory in Teddington solves linear-programming problems with a framework made up of parts from a toy construction set. Thomas A. Reisner of Université Laval in Quebec generates a contour map of a surface by spreading mosquito netting over it. A strong overhead light creates a moiré pattern as the net interferes with its own shadow.

The U.S. citrus industry apparently uses an analog gadget to sort fruit. Oranges roll in the channel between two not quite parallel pipes and fall through when the distance between the pipes is equal to the diameter of the orange. John P. Schwenker of Louisville, Colo., once found the center of gravity of a piece of equipment by a variant of Ronald L. Graham's plate-balancing technique. When the equipment is dragged by a rope across a smooth surface, the vertical plane passing through the rope also passes through the center of gravity. The intersection of three such planes identifies the center of gravity itself.

Some patterns recognized by window perceptrons

Is This a Black Hole at the Core of our Milky Way Galaxy?



ON A BARREN plain in New Mexico, 27 antennas of the Very Large Array radio telescope have charted the heart of the Milky Way Galaxy for the first time. (The core of the Galaxy is normally hidden from view by dust clouds and the glow of intervening stars.) The resulting images show three curving streams of gas which appear to be falling into a maelstrom at the center. At the very core, pulling everything in, there seems to be a supermassive "something"—perhaps a black hole.

Black holes are conjecture—hypothetical bodies so dense that the gravity field around them does not allow light or anything else to escape. Yet they appear to exist at the core of all galaxies. And perhaps elsewhere.

The mass of three Suns crushed into a space smaller than this period (!)

When a large star has spent its nuclear fuel, it blows apart in a colossal explosion which we call a supernova. All that remains is a tiny, hot, burned-out core. Compressed by its own gravity, the core shrinks, becoming denser and denser. If the star is heavier than three Suns, gravity crushes its matter into a space smaller than the period at the end of this sentence. The star "blinks off" and a black hole is the result.

An incredibly destructive force? Yes, but astronomers think the black hole—or whatever it is—at the core of the Milky Way may have been there from the beginning. And it may hold the secret to how the Galaxy began as well as how it might end.

Are we alone in the Universe?

Astronomers may get to the heart of the matter (in the ultimate sense!) within your lifetime—maybe within the next few years. Meanwhile, other fascinating discoveries are taking place.

Astronomers have recently discovered bands of solid particles orbiting several other stars, including such naked eye stars as Vega and Fomalhaut. These stars are surrounded by rings of relatively cool, small solid material perhaps no larger than grains of sand which may be the stuff of planets in some early stage of creation. The system of planets we inhabit may not be unique after all.

There are billions of stars in our Galaxy alone—so somewhere else in the Galaxy there may be a solar system like ours. It's perhaps the most exciting indication yet that "mankind is not alone."

Presenting ASTRONOMY Magazine.

If discoveries such as these interest, enthrall, or excite you, then join us now in the quest for knowledge as a subscriber to ASTRONOMY Magazine. By reading ASTRONOMY every month, you'll appreciate the significance of discoveries that few other people even begin to grasp.

ASTRONOMY is brilliantly illustrated—mainly in color—for astronomy is the most beautiful of all sciences. The magazine is authoritative—it's written largely by astronomers—but edited for

the layman. Articles take you step by step from the basics all the way to the frontiers of astronomical knowledge!

While other people remain Earthbound, you can be exploring the solar system, the Galaxy, the outer reaches of space. You can be eye-witnessing the most awesome phenomena. Contemplating the most astonishing possibilities. Challenging your mind and stretching your imagination.

You'll be involved!

ASTRONOMY involves you—intellectually and emotionally. And if you choose to become one of the tens of thousands of amateur astronomers, ASTRONOMY will be your guide also. We'll show you how to observe and photograph the planets, the stars, and celestial phenomena. How to use a telescope—or build one from scratch.

Discover ASTRONOMY Magazine now and receive a Free "MAN FLIES FREE" Poster

Return this coupon today. You'll receive 12 issues of ASTRONOMY for only \$15—\$9 off the single-copy price. And with your paid subscription, you'll receive a free color 21" x 30" wall poster commemorating the historic space walk made from the Shuttle Challenger, when "man flew free" for the first time—no tether, no lifeline.



YES, I'm fascinated by it all and I want to know more. Start my trial subscription to ASTRONOMY—satisfaction guaranteed. My introductory price is just \$15 for a full year—12 monthly issues. I save \$9 off the single-copy price. And I get a free 21" x 30" MAN FLIES FREE poster with my paid subscription.

Bill me later.

Payment enclosed—send poster right away.

NAME _____

ADDRESS _____ APT. # _____

CITY _____

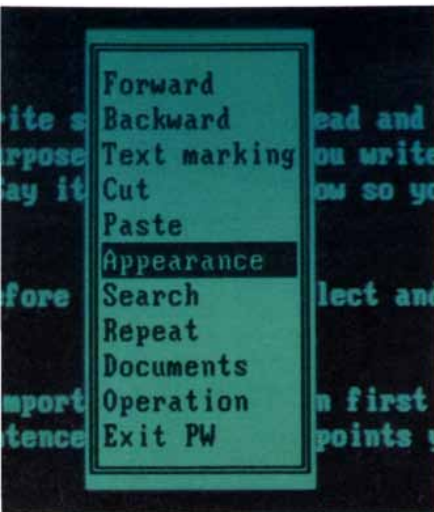
STATE _____ ZIP _____

Mail to: 5194
ASTRONOMY
P.O. Box 92788, Milwaukee, WI 53202

GUARANTEE

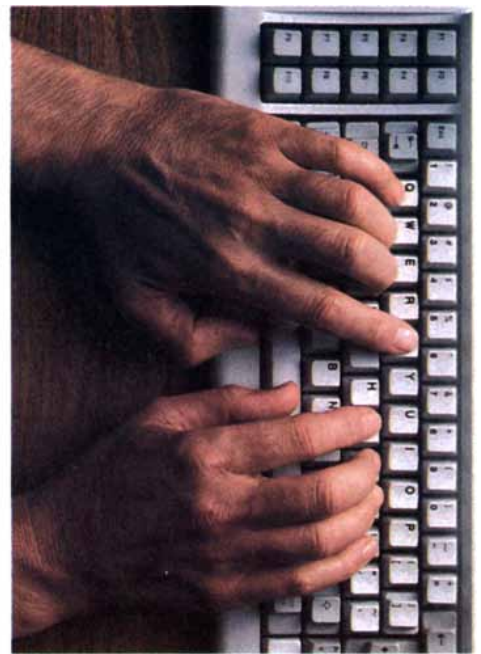
If you don't agree that ASTRONOMY is out of this world, you can cancel at any time, receive your money back on all unmailed issues, and still keep your MAN FLIES FREE poster free!

If a word processing these 8 tools for better



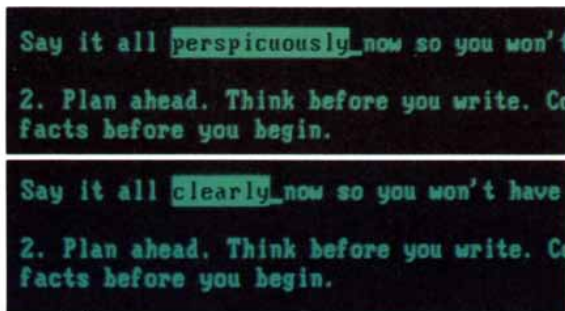
1 Pop-up menus—so you can start writing better right away.

You can't write better if you're not writing. So Perfect Writer™ has pop-up menus that make it extra easy to use. They guide you through every function with simple English language words. So you don't have to struggle with complicated commands.



2 Simple function that lets you keep your mind on what you're writing.

Instead of hard to find function or control keys, Perfect Writer lets you edit with the keys right under your fingertips. And because our menus appear right on the screen with your document, you never have to leave your text. And lose your thoughts.



3 Powerful editing features at the touch of a single key.

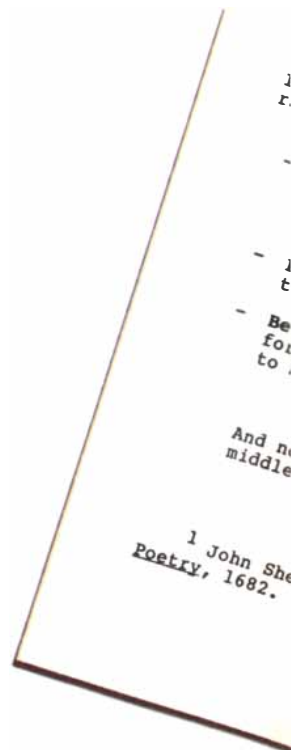
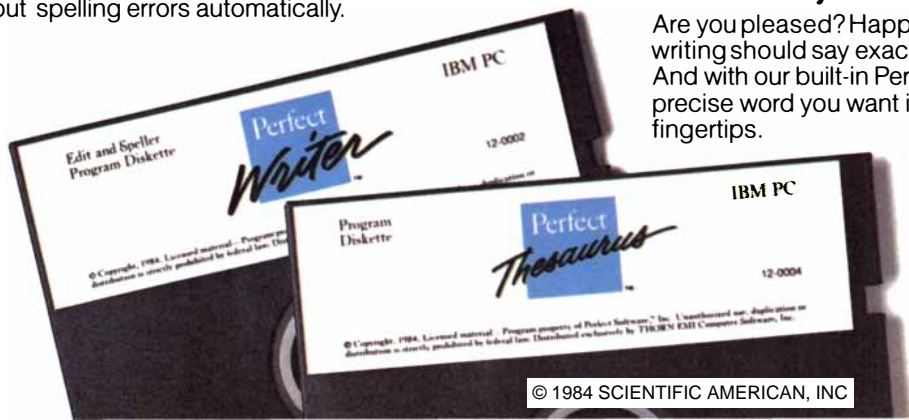
Perfect Writer makes it extra easy to make changes for the better. One key at a time is all it takes to move text. Search and replace. Add or delete words, sentences, even entire paragraphs.

4 A 50,000 word dictionary that helps you correct spelling mistakes.

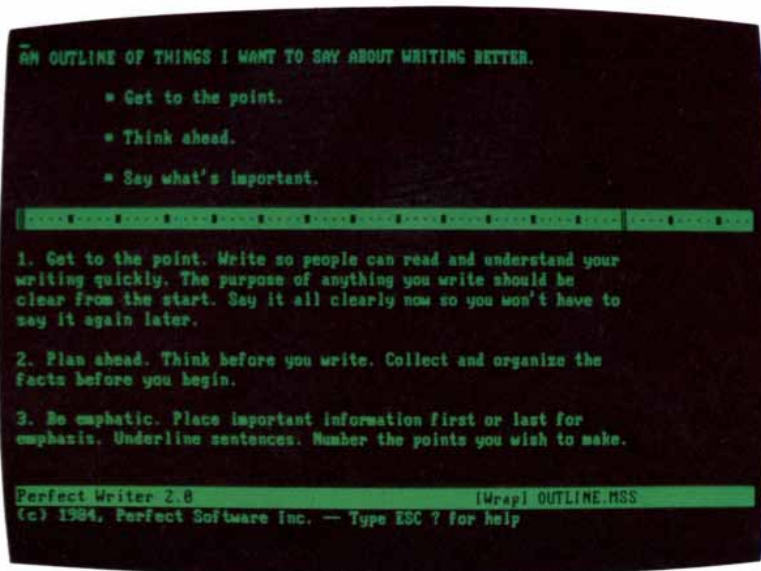
Nothing destroys good writing faster than bad spelling. So Perfect Speller™ checks your document and points out spelling errors automatically.

5 A Perfect Thesaurus™ to help you choose exactly the right word.

Are you pleased? Happy? Delighted? Good writing should say exactly what you mean! And with our built-in Perfect Thesaurus™, the precise word you want is always at your fingertips.

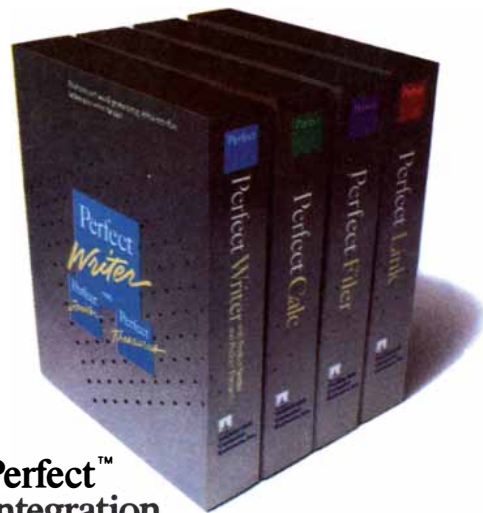


program doesn't give you writing, it's not Perfect.™



6 Split-screen windows that help keep your thoughts organized—while you write.

It's like having a notepad right on your screen. You can use one window to jot down notes, key points or an outline—as you develop your text in the other window. See? With Perfect Writer, you can really compose your prose.



7 Automatic formatting to make your writing look even better on paper.

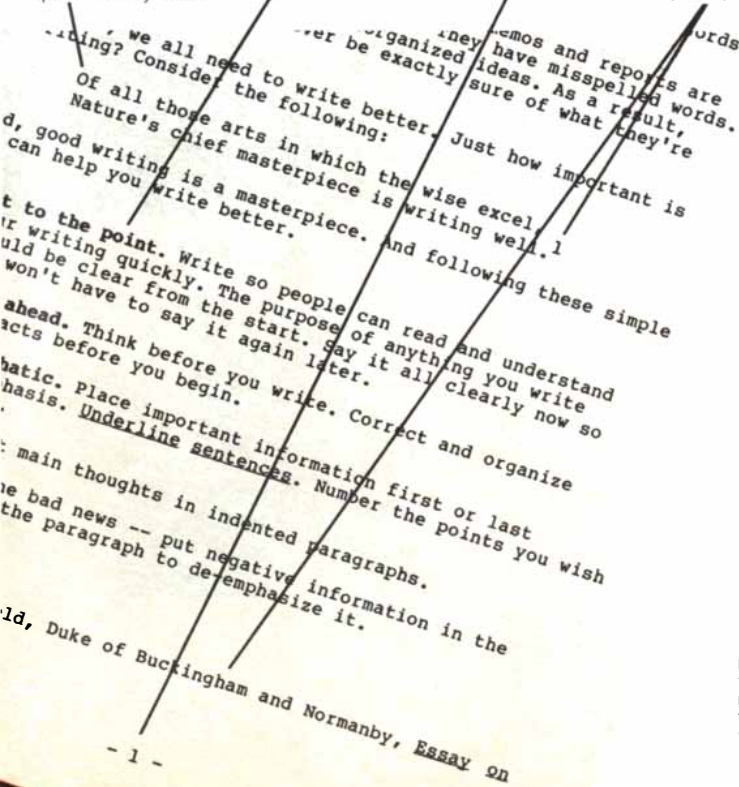
Perfect Writer works with most popular printers. And with our special document appearance features, your letter, memo, report or paper will look like a masterpiece.

It's easy to change margins and spacing to display important quotes. Or key ideas.

Boldface and italics let you write with new emphasis.

If you want page numbers, Perfect Writer can handle it—automatically.

Perfect Writer automatically numbers, positions and prints out footnotes. Whew.



8 Perfect™ integration with other Perfect software.

Perfect Writer is just part of the complete, integrated Perfect Software family. There's also Perfect Calc™ spreadsheet. Perfect Filer™ database management. And Perfect Link™ telecommunications software. You can share information between programs. And, each program uses common commands. So they work Perfect™ together—to help you work better.

Why settle for anything less than Perfect™?

Perfect The Perfect way to write better.



THORN EMI Computer Software, Inc.

THORN EMI Computer Software, Inc., 3187 C Airway Avenue, Costa Mesa, CA 92626

Perfect Software is available for the IBM® PC, IBM PCjr, Apple® Iie and Apple Iic computers, as well as for computers that use MS™-DOS, CP/M® 80 and CP/M® 86.

IBM is a registered trademark of International Business Machines Corporation. Apple is a registered trademark of Apple Computer Corporation. MS-DOS is a trademark of Microsoft Corporation. CP/M is a registered trademark of Digital Research Corporation.



You're traveling through 140° terrain
at 300 rpm.

**While some disks lose their way in the torrid zone of drive heat,
Maxell guarantees safe passage.**

A lifetime warranty. And manufacturing standards that make it almost unnecessary.

Consider this: Every time you take your disk for a little spin, you expose it to drive heat that can sidetrack data. Worse, take it to the point of no return. Maxell's Gold Standard jacket construction defies heat of 140°F. And keeps your information on track.

And Maxell runs clean. A unique process impregnates lubricants throughout the oxide layer. Extending media and head life. How good is Gold?

Maxell's the disk that many drive manufacturers trust to put new equipment through its paces. It's that bug-free.

So you can drive a bargain. But in accelerated tests, Maxell was an industry leader in error-free performance and durability. Proving that if you can't stand the heat you don't stand a chance.

maxell®
IT'S WORTH IT.

Maxell Corporation of America, 60 Oxford Drive, Moonachie, N.J. 07074 201-440-8020



BOOKS

The numbers of physics, a life in molecular biology, a 2-D world, ecology of body size

by Philip Morrison

THE CONSTANTS OF PHYSICS, edited by W. H. McCrea and M. J. Rees. The Royal Society, 6 Carlton House Terrace, London SW1Y 5AG, England (23.40 pounds sterling or dollar equivalent). Looking from the summit of success climbed in the past decade, the gauge-field theorists think they see a grander unification at the horizon, and they have boldly marched toward it. Their vision is wide indeed, albeit distant; it joins an understanding of the inwardness of the quarks to an unprecedented familiarity with the expanding universe past and present.

This book is a collection of a dozen and a half related papers buoyed by the cresting wave of theory. Some take only a page or two and can be easily understood by the informed general reader, some are long and difficult summaries meant for hard-bitten traffickers in differential equations. Most of them are followed by a transcript of brief discussion, since the thin volume records a meeting held by the Royal Society in May of last year. Those present included a wide international cut of physicists, some of them bearing very well-known names. Their topic is the key numbers, in our quantum epoch easily put into dimensionless form, that measure the intrinsic strengths of all the forces that couple particles, along with certain other parameters of microcosm and macrocosm that underlie the particulate universe we inhabit. The meeting recorded opinion almost at its most sanguine. In the past year there have come several hints from the big laboratories and the underground experiments, some confirming the electroweak theory brilliantly at the energies we now command, some more recently hinting that nature's way may not after all follow the simplest of erudite conjectures out to the requisite dizzying extrapolation.

The most familiar ground is explored in the first half-dozen papers, looking over the measurements of the basic constants. The national standards laboratories are coming to found their central units on "well-understood quantum systems." The second of time is of course fixed by the cesium atomic clock; the speed of light and the atomic clock to-

gether have just replaced the platinum bar to fix the meter, and both the volt and the ohm will soon be determined by certain precise quantum steps electrically measurable in condensed matter at low temperature. The kilogram will soon enough be defined by a count of atoms based on the spacing of crystal lattices measured by X ray. Meanwhile dimensionless constants such as the fine-structure constant—the notorious $1/137$ that serves as a measure of electric charge—are known better and better. The discrepancies now are at the level of only parts per million.

The elusive measures of gravitation, isolated from all the other constants, are closely watched: the most discussed of them is the idea, perhaps inconsistent, that the gravitational constant is changing as the expanding universe becomes more dilute. A good review establishes that the change, if any, is less than one part in 10 billion per year. This result is based on a detailed analysis of the motions of Mars using signals from the Viking Lander combined with *Mariner 9* data, and on laser reflections from a retromirror on the moon. Methods now current promise an improvement in accuracy by a factor of 10, but not much more than that. The limit for orbit precision in the solar system is now set by asteroid noise; there are quite a few big asteroids still unweighed and thousands of little fellows. They all add up, to daunt even the modelers who command supercomputers.

"There is now a small, international industry dedicated to testing relativistic theories of gravitation," states one paper. The approximate truth of general relativity is well supported by a variety of first-order results, to 1 percent or better. The time has come to probe deeper, to check second-order effects and to study quite new phenomena. Much can be done in the isolation of space, and the exquisite timing studies of the binary-pulsar orbit and lately of the millisecond pulsar challenge the space experimenters to improve on what is now being done by the ground-based radio astronomers exploiting those serendipitous natural clocks.

Setting macrocosmic gravity aside, it

is possible to seek changes in the constants of microphysics within astronomical and geologic contexts. The ratio of spectral frequencies seen in quasars and other distant sources is not hard to measure to reasonable accuracy, choosing spectral lines whose energies depend differently on the various constants. The 21-centimeter radio line can be compared with the normal lines in the visible spectrum of hydrogen, or one can compare visible lines from iron with those from hydrogen. Both allow the comparison of nuclear properties with electronic properties, the one for magnetism, the other for mass. All of this leads to a direct check of the nominal $1/137$; it is invariant over most of visible space-time within one part in 10,000.

A few pages by J. M. Irvine of Manchester clearly set out afresh the marvelous deduction first drawn in 1976 by the Russian physicist A. I. Shlyakhter. In the Oklo uranium mine in the African nation of Gabon there were found residues of several natural nuclear chain reactors, which were boiling away merrily in river sands for a while a couple of billion years ago. Now, there is just one energy value we can be sure of to high precision, even without measurement. That is the kinetic energy of a particle at rest: zero. Thermal neutrons move at the speed of sound, but that is scant energy indeed compared with the intrinsic energies of the neutrons' capture by atomic nuclei. Yet mineral analysis proves that thermal neutrons in the Oklo mine long ago were resonantly captured in exactly the same rare isotopes that would swallow them today. This implies the nuclear quantum energies have not shifted enough over the past two billion years to detune the narrow nuclear energy level away from its chance resonance at near-zero kinetic energy. The electric repulsive forces and the specifically nuclear attractive forces must have struck then nearly the same bargain they keep today; agreement implies that the relevant microconstants cannot have changed by more than a few parts in 10 billion over the entire time.

Still, the coupling constants are not truly constant under all circumstances. The very basis of the grand unification of the nuclear, weak and electromagnetic forces is the understanding that the external effects of electric charge and its less familiar analogues all depend on the energy with which particles collide. Several reports here survey the situation from various more or less conventional points of view; none of them is really accessible to the reader not familiar with gauge theory, although the conclusions are understandable. The chief conclusion for a decade has been that the three forces would all show the same effective strength at energies equivalent to about a million billion proton masses, if no other phenomenon intervene

before the tiny distances implied are reached. Thus the forces between the particles in a gas are dependent on temperature, and the way is opened to examine what happened in the very early universe, which by extrapolation is shown to be hot indeed.

A few reports sum up the inferred states of that plausibly conjectured inferno. We live now in a cold universe of protons and neutrons; when radiation and matter were as hot as the beams of today's big accelerators, all was unconfined quarks and gluons. At a much higher temperature still all microforces became equal. All was symmetric; matter was of a kind now entirely alien but governed by high symmetry. Along this line of thought, based of course on simplified models proposed for a world not known in detail, many features of our universe (or better, surmises about it) come clear. They include the asymmetry between ordinary matter and the rare but equivalent antimatter, and the measured tendency to expand with near-zero acceleration. It is all quite impressive, except for the brutal fact that the same arguments applied in hope to the gravitational field yield a rough estimate of the present energy content of cosmic space that is too high by a cool factor of some 120 powers of 10! We do not know what is so wrong; most probably gravitation is not simply one more example of a quantum field theory but, as a couple of papers here try to show, something very distinct.

A delightful exploration into known scales is made by William H. Press and Alan P. Lightman of Harvard. They estimate one after another the characteristic properties of solids and molecules, of rocks, asteroids and stars, of planet spins and wind speeds, of human stature and the speed of a human runner, in each case expressing their results for all these phenomena in terms of the fundamental constants of physics. A runner can move at a calculable fraction of the speed of light; that fraction depends only on the fine-structure constant, on its gravitational analogue and on the ratio of the mass of the electron to the mass of the proton. The calculation predicts (perhaps to some degree by good luck) a six-second record for the 100-yard dash, which is not too bad for such an elevated approach.

A less convincing study examines the so-called anthropic principle, by which we are enjoined to remember that we could not, for instance, observe too hot a stage of the universe; the existence of observers implies certain limits on their environment and hence on the physics of the cosmos. "Although our situation is not necessarily central, it is necessarily privileged to some extent," argues the author, Brendon Carter. He goes on to make quite a lot of this through models of long-term evolution; the conclu-

sions require a certain enthusiasm for simple formulas.

The most original paper here, not at all an easy one, is by the Copenhagen theorist H. B. Nielsen. He begins disarmingly enough with the example of the kinetic energy of a free particle at low speed, the familiar square of the velocity. That limit can come out of an almost arbitrary version of the true form of relativistic dynamics. The result must, apart from constants, depend on momentum alone; it cannot depend on position, since that is what is meant by a free particle. If energy is also to be independent of direction, the speed must appear as an even power. The lowest power possible gives the limit at low speeds. The familiar result follows, just the correct limit from relativity, unless the coefficient of the square term is "by chance" zero. Now, if the laws of dynamics were so complex as to approximate a randomly chosen set of functions for the true energy, the result would nonetheless usually be the one we know, derived in the limit from a few principles alone. The details at a more fundamental level may not matter much if we do physics only in a restricted range, "in a corner," say at low energy.

By extension Nielsen proposes—with examples—that the laws of physics, all the powerful symmetries and simplicities we have, are simply the evolved outcome of physics done in one corner of the room of possibilities. Underneath everything there may be levels of structure that are extremely small and far too complex to guess at. Our experiments, even in the highest-energy beam, may still be at large space-time distances compared with those unknown scales. The regularities we rely on—powerful gauge invariance and the conservation laws—may, like the velocity-squared law for kinetic energy, depend little on the unknown details at the lower levels. Random dynamical systems might give rise to the special symmetries we see with our clumsy spatial resolution and our slow gaze, without requiring the fine tuning that symmetries appear to imply. All laws stand merely as approximations valid over a wide range of underlying, unknown and perhaps disagreeably complex theories. The laws and constants of physics thus evolve, so to speak, as the cruder simplicities that roughly average out some complex and chancy substructure lying far below our vision.

The Copenhagen group offers illustrative examples closer to the genuine symmetries of today than the merely pedagogical case of kinetic energy. The four dimensions of space-time, the linear quantum equations, relativity and more might be shown to arise from deeper laws of much less simple structure. (There is another technical paper here that takes a wide look at possible

high dimensionalities, recently rather popular.) Such hidden theories might in our ignorance be just as well described by complicated systems of equations chosen at random. A few simple examples have been carried through with some success.

Given the eventual working out of Nielsen's program, the origin of the laws of physics would be understood statistically, although the laws we now know would all be approximate. As most theorists imagine their final success, however, they expect one day to set eyes on that One Golden Lagrangian, whose wonderful groups of symmetries, in some domain exact, would control everything that exists. But whence that elegant utterance? The uppercase letters would have to bear the entire burden of origins. Someday we shall see; meanwhile this slender, varied and difficult book records a scientific program at the brilliant summit of its hopes, and it holds as well a small hard seed of dissent. These papers are already in the libraries; they were published in 1983 in Vol. A310 of the *Phil. Trans. Roy. Soc. London*.

A SLOT MACHINE, A BROKEN TEST TUBE: AN AUTOBIOGRAPHY, by S. E. Luria. Harper & Row, Publishers (\$17.95). Only a chunk of paraffin, it lay on the windowsill in Edoardo Amaldi's laboratory in Rome, ready to hand for tyro Luria to melt up for his casual purpose. Fortunately Amaldi was right there to rip the piece from his assistant's hands just in time. It was no ordinary chunk of wax but "a holy relic" like the telescope of Galileo, carefully matching the hand-carved chunk of lead Fermi had compared with that very wax in that very room years before when nuclear transmutation by the absorption of slow neutrons was first found, and nuclear fission and the nuclear chain reaction were foreshadowed.

The new assistant was a bright if romantic young man from gray, restrained, "Protestant" Turin, buffeted between the lively familiarity of Roman manners and the stern expectations of Fermi's hardworking crew of physicists. They called him Signor Garzone (freely, Mr. Lab Boy); he was 25 and held a fresh M.D. In 1937 Italy an M.D. was an easy path for the studious son of school-oriented striving Jewish parents, the handsome mother a semi-invalid dependent on opiates and secretive about it. Young Salva shared no healer's vocation; his aim was a profession, although his dream was of science. The compromise career he sought was the physicist's one of radiologist. His year among the physicists was justified as a remedy for incompetent instruction in a dull specialty, as it was taught and seen in Turin.

When the next year Fermi left for Stockholm, never to return to Fascist

For the 43rd consecutive year
the Westinghouse Science Talent Search
will help hundreds of deserving young people
along the road to higher education.

Help your brightest students get involved!

Since 1942, winning "the Westinghouse"
has been one of the most elite academic
achievements that American high school
students could attain.

And the recognition associated with
participating on a national level in the only
competition of its kind has helped thousands
of seniors receive financial aid and
recommendations to America's top colleges
and universities.

The rewards last a lifetime.

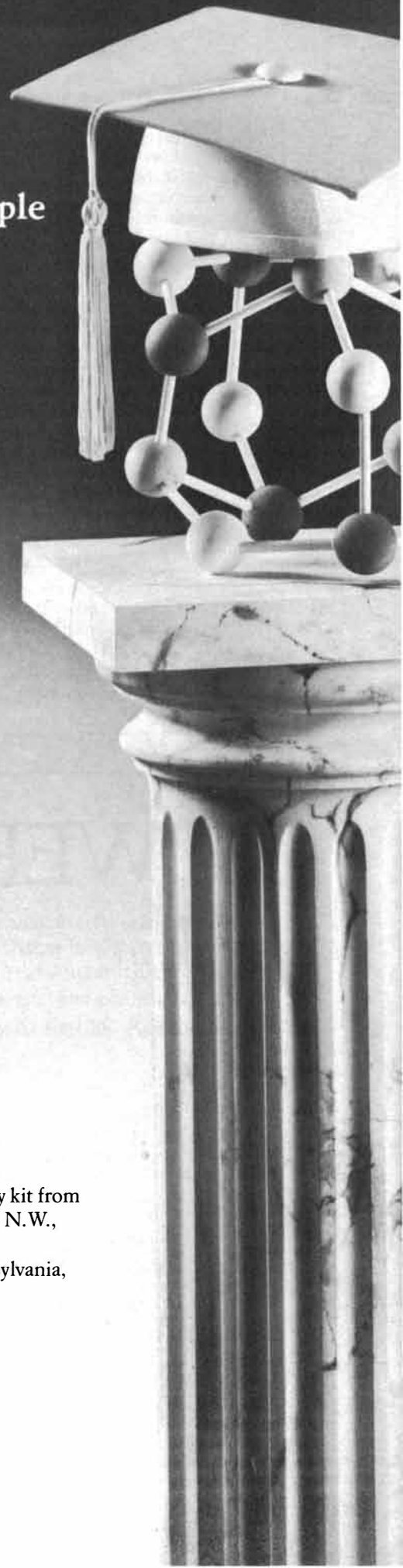
Science Talent Search winners earn
doctorates at a rate 25 times the national
average for college graduates, and go on to
assume vital roles in America's industries,
universities and research institutions.
And no fewer than five times between 1972
and 1981, former winners have been
awarded the Nobel prize.

*Research projects must be received by
December 15, 1984.*

For further information:

Request a complete contest entry kit from
Science Service, 1719 N Street, N.W.,
Washington, DC 20036.
Or call 800-245-4474. In Pennsylvania,
call 800-242-2550.

You can be sure... if it's Westinghouse



Italy, Luria, dreaming both of freedom and of science, left for Paris. He was lucky; his interests in radiation biology and bacteriophage, the one kindled by the Rome physicists, the other by a chance friendship formed on a stalled trolley in power-short Rome with a young bacteriologist, led him to a research post in generous anti-Fascist Paris. He learned both research and political concern there, until the Germans marched too close. By the summer of 1941 Luria and the German emigré theoretical physicist Max Delbrück, whose piece on the gene as a molecule had drawn the dreaming Luria along with him on the road to a genuine biophysics, were seasonal colleagues at Cold Spring Harbor on Long Island and were secure from the wars.

The organisms were called bacteriophage because they ate bacteria. It was not the right term; the phage group preferred to talk of bacterial virus particles. A virus particle is open to physical study; not long after Pearl Harbor, Luria, who to this day is uneasy with technology and finds "even laboratory instruments forbidding," worked with a biophysicist colleague, Tom Anderson, to produce with the new RCA electron microscope the first good images of phage particles. Radiation, imaging, sta-

tistics; that was how to find out about bacterial virus. The time had come for sharp reductionism: genes were molecules, viruses held strings of genes, and their bacterial hosts should be not much different, however biochemists might demur. Molecular biology was budding in those wartime years: one red bread mold gene made one enzyme. It looked as though what entered to permanently transform the pneumococcus was the substance DNA, and phage was more a tiny inert proteinaceous structure loaded with nucleic acids than a hungry predator on bacterial cells.

The experiments showed that a few bacteria always survived the fiercest attack of phage. Were these survivors resistant by some chemical action set up by contact with phage or were they born resistant by genetic inheritance? A slot machine at a weekend faculty dance at Indiana University gave Luria the concept of his test, and half of the title of his book. Steady losers trickled coins into the machine, but once in a while someone would hit the jackpot. The fluctuations are uncommonly large. It was not hard to devise a scheme to study the fluctuations in phage resistance. By Tuesday afternoon the first experiments were finished. If mixing with phage induces rare resistants in a large colony of

bacteria, the fluctuations will be small. If the resistants occur by spontaneous mutation in the course of growth, however, there will be some that mutate early, to leave huge numbers of their resistant progeny. Jackpot! Resistance was genetic. The physicists had chosen the right metaphor; individual spontaneous events worthy of study can happen to particles. Bacteria were overnight elevated to genetic leadership; no other forms could be studied genetically in such large numbers so quickly.

Nobel prizewinner Luria counts one even more important result to his credit. It was a chance result that depended on a broken test tube. What he was after was to see what happened when certain bacterial mutants appeared to be infected by the virus but did not give rise to more virus in their turn. Never a particularly neat worker, Luria lost with that tube the entire supply of phage-sensitive bacteria he was about to test. The replacement he could borrow quickly was a sample of a different bacterial species. Those bacteria should have worked as well. In fact, they worked too well: they yielded plenty of infectious phage. The effect had not been the failure of phage to engender phage progeny but rather the copious production of a modified phage, one restricted in its host range,

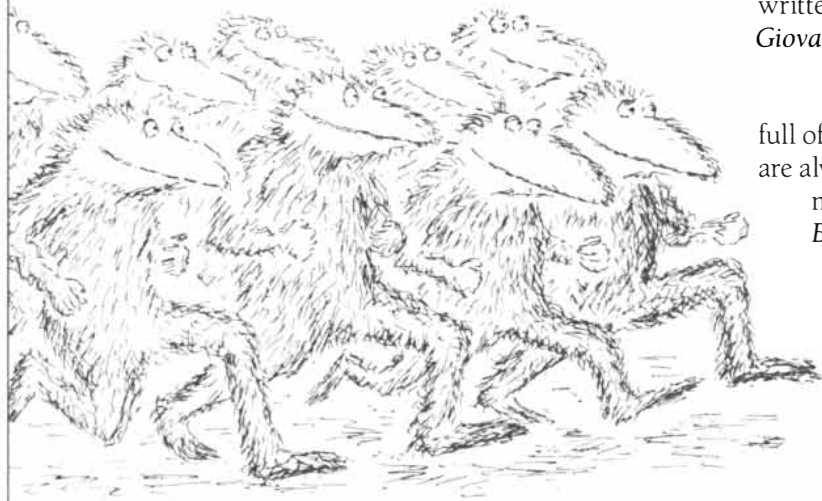
POWER-BASE™ PUTS YOU W

"Power-base™ is a menu-driven data management program with relational capabilities. It was the top-rated program overall, primarily because it was the easiest to learn and use, and rated highest in error handling." *The Ratings Book/Software Digest, March 1984*

"To track the Bank of New England's IRA sales for all 16 branches, I chose Power-base™ because it responds so well to change while protecting the integrity of my database." *Bruce Montgomery, Bank of New England*

"This program is impressive—it's easy to use, simple, logical, and powerful and comes with flawless written documentation." *Giovanni Perrone, InfoWorld, March 12, 1984*

"Because you don't have to wade through screens full of menus, the program is fast. Because your choices are always in front of you, the program is easy. Not much more you can ask for, is there?" *Bobbi Bullard, Computer Retailing, April 1984*



able to grow on the borrowed strain but not on the original. It turns out that some bacteria can become immune because they develop specific enzymes to cut intruding DNA at particular coded positions. Such restriction enzymes are now the indispensable tools of DNA manipulation.

These two discoveries lie at opposite poles of style in science. The fluctuation test was an apt product of the free imagination. If Luria had not hit on it, probably no one else would have. The proof of spontaneous bacterial mutation would nonetheless have come soon by the route pioneered by Joshua Lederberg, a clever but much more straightforward technical device able to reach the conclusion without explicit statistics. Phage modification and the restriction enzymes were found by following up on an accidental revelation; if that tube had not broken, some other phage investigator would have come on them within a year or so in systematic trial.

Three men won a Nobel prize for phage in 1969: Luria, Delbrück and Alfred Hershey. Max was surer of himself, organized, convivial, but he regarded politics as a "distasteful joke." "He was certainly not lavish with approval." When Luria's standard monograph on viruses appeared, Delbrück never said

anything about it to Luria except to point out a misprint in a footnote. Hershey was a loner, silent, his work and his writings sharing "a spare elegance." Asked once for his notion of heaven, Hershey answered: "To find a perfect experiment and do it every day for the first time."

About a third of Luria's autobiography treats these external topics, always briefly and clearly. There is all too little of younger years in Turin, Rome and Paris; even so they are convincingly present. The book nonetheless achieves much more; it is at once confessional and analytic of motive. Organized by development rather than by calendar alone, the text outlines what Luria did and who he is by describing the growth of his personal set of commitments, active guides to action within this or that sphere of a long and reflective life.

Luria's mind and heart are held by the search for pattern, pressed through the solution of problems in quest of a growing and demonstrable order. His is an active and participant mind; the large but often vague questions of origins and the cosmos do not attract him, and passive surrender to entertainment is rare. "Theater I love," but it is the actor's vivid performance rather than the play that attracts. (He reports that in spite of per-

vasive exposure, he has watched just one program on television in a lifetime.) Antipathetic to all sports, he is a dedicated worshiper of the indoors, and he would not so much as pick up an experimental crab with his fingers. "There was something forbidding to me in the appearance of the sun-etched, bearded giants in jeans and sailing clogs who used to populate nature-study laboratories... at Woods Hole (before a crowd of tennis-playing biochemists came to take their place)." Science is conceptual clarity for this logical Piemontese of the labs; biology is inward order and not bemusing diversity.

It is unlikely any other well-known scientist could recount that for years he had invited the new graduate students weekly to his home for "a Sunday-night literary seminar." There they read works around one high theme, say good and evil, the Greeks to Hesse, Proust and the Gita, as their mood and the ambience led them. Even less expected is Luria's love for poetry; he can still recite school-day set pieces, and within the past decade he has come to know and love the moderns, a Roethke or a Rich. Here again it seems that pattern is his aim; the characters of Shakespeare appear rather shallow to him. It is what they say, not what they are, that is so

WAY AHEAD OF EVERYONE.

"Much to my surprise, my search for a data-base system took six months. But at last I've come up with a winner: Power-base.™"

Paul Bonner, *Personal Computing*, February, 1984

SEND OR CALL TODAY FOR YOUR COMPLETE POWER-BASE™ DEMO

Experience Power-base™ yourself. Send \$10 with the completed coupon below for your Power-base demo.



Or call 1-800-237-4778 (In NY, 212-947-3590). Use the demo to set up actual applications (restricted to 25 records) on your personal computer.* See how simple it is to learn and use Power-base. We'll refund the \$10 when you purchase Power-base at any computer dealer.

WE MADE IT POWERFUL. BUT WE KEPT IT SIMPLE.

Send me my demo copy of Power-base.™

Mail to: PowerBase Systems, Inc. Dept. A-1
12 West 37th Street, NY, NY 10018

NAME _____ TITLE _____

COMPANY NAME _____

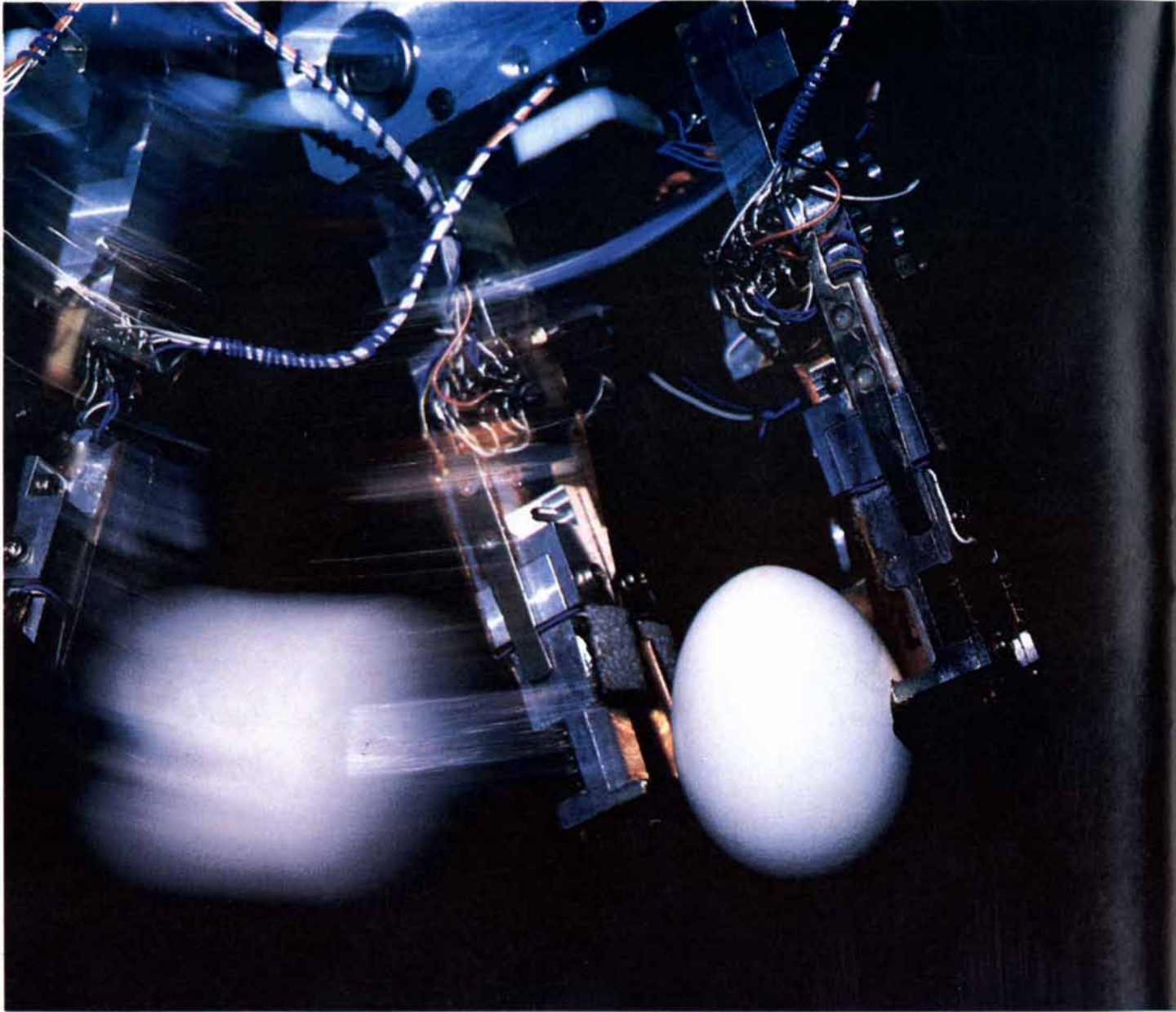
ADDRESS _____

CITY _____ STATE _____ ZIP _____

My \$10 CHECK IS ENCLOSED

VISA MASTERCARD # _____ EXP DATE _____

*For use on the IBM PC and PC/XT or compatibles. Power-base and DataZOOM are trademarks of PowerBase Systems, Inc. IBM PC and IBM PC/XT are registered trademarks of the IBM Corporation.



Hitachi's visual-tactile robot can handle objects as fragile as eggs, because its sensors detect size, shape and required pressure to attain sensitivity almost equal to that of a human hand.

ROBOT

Nearly two decades ago, Hitachi began turning common science fiction into startling industrial fact. The device: The company's first servo-manipulator, a key component in the development of real robots to eliminate the monotony, danger and dirty work of manufacturing.

Your mechanical right-hand man

Today, the results of Hitachi research are in use all around you. Robot welders using micro-computers and built-in sensors to detect weld lines automatically. Spray-painting robots capable of remembering up to 2,000 instructions and performing 99 different painting tasks. Process robots that can be programmed for new job functions through a simple teaching box. Robots on wheels for transporting parts and warehouse stock.

Our electronics and mechanical engineering experts have joined their talents to give robots the benefits of high technology. They have created models with expanded memory capacities and advanced sensing systems. And they have applied them in Hitachi's own factories, where our production specialists suggest further refinements.

In fact, we are constantly coming up with innovations and new applications. One of the

latest: A visual-tactile sensing robot with multiple arms and seven camera eyes, developed to independently assemble home appliances such as vacuum cleaners.

These examples demonstrate a few of the ways in which Hitachi is improving upon basic technology. Then using it to create practical tools that meet your needs...and those of professionals in marine exploration, aerospace, and virtually every other field you can name.

The best of worlds is yet to come

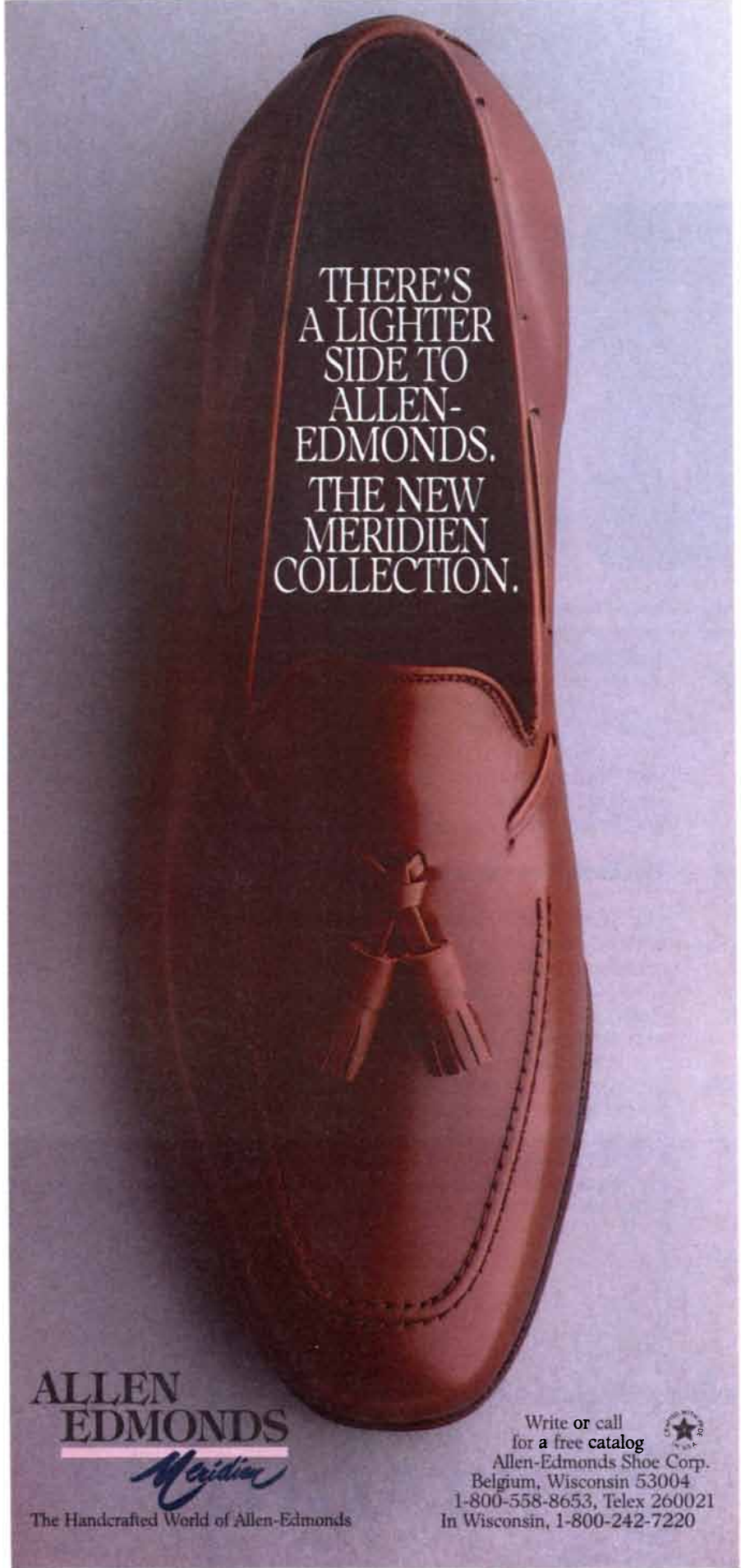
Our vision of the future includes robots with artificial intelligence that will learn from their own experiences. Flexible manufacturing systems where robots handle every step of production. Personal robots that will take the drudgery out of household chores. And much, much more.

We'd like you to share in the benefits of our scientific research, covering the next generation of lasers, sensors and other electronic devices. For improved business efficiency. For a higher quality of life. Two goals we've pursued for 74 years as part of our commitment to a better world through electronics.

WE BELIEVE ROBOTS FREE MINDS TO CREATE BY FREEING BODIES FROM TOIL



Robotics Group, Industrial Components Sales & Service Division, Hitachi America, Ltd., 50 Prospect Avenue, Tarrytown, NY 10591-4698 Tel: 914-332-5800



THERE'S
A LIGHTER
SIDE TO
ALLEN-
EDMONDS.
THE NEW
MERIDIEN
COLLECTION.

ALLEN
EDMONDS

Meridien

The Handcrafted World of Allen-Edmonds

Write or call
for a free catalog

Allen-Edmonds Shoe Corp.

Belgium, Wisconsin 53004

1-800-558-8653, Telex 260021

In Wisconsin, 1-800-242-7220



riveting, as in the modern poets word sounds and concept may fit.

Existential as a matter of philosophical stance, Luria has long been a democratic socialist, his view based neither on reasoned study nor on mere allegiance to a political creed. He has chosen to cast his lot with social justice and human equality. He has been a firmly gentle, if sometimes sharp-penned, radical, anxiously alert to effective compromise. Some of his best political work was done over the telephone, say in persuading editors to publish statements of dissent. For more than 70 years he has watched dismayed the war and slaughter of our century. "I dread [that] I may yield to the temptation to tend my own garden. Could I then preserve my self-respect?"

Would you know more of a scientist and a man? More is here: rightful if uncommon praise for his first—and best—graduate student, James D. Watson, and the honest but artful book Jim wrote about the double helix, all intensity, arrogance and joy. There is an appreciation of his long good marriage to Zella Hurwitz, a psychologist and teacher of integrity and distinction, her work as thorough in commitment to complex human issues as Luria's to the colony count. There is a little about his enemies, most of them the users of cant. He accepts no claim by science either to "responsibility for [or] absolution from the problems of society." Finally, we learn of his severe depressions, a frequent disability over the decades. For 10 years Luria has been freed of his fear by intelligent chemotherapy, long delayed in the finding.

It is easy to admire this cultivated, ironic man and his lucid, pointed book of an enviable life founded alike on reason and on commitment. For readers who find empathy for the growth of candor and self-awareness as the chapters pass, admiration will be colored with a deeper cast. There are no modern autobiographies and not many novels that can more reward readers young enough to find the grand choices in life still before them. The reward is of course not one path for Everyman but instead a way of going on.

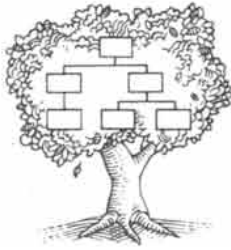
THE PLANIVERSE: COMPUTER CONTACT WITH A TWO-DIMENSIONAL WORLD, by A. K. Dewdney. Simon & Schuster, Inc. (\$16.95). Yendred was shocked to learn that the humans around the terminal his own was linked to by some mysterious informational resonance had a food channel running clear through their bodies. "Why do you not then fall into two pieces?" he sensibly asked. Yendred is a sentient two-dimensional creature, living on—or in—a disk planet lost somewhere in the depths of planiversal 2-space. All we know of him and his kind we find in this unique account by a Canadian mathemati-

cian, whose name by some coincidence resembles after simple transformation that of his alien friend. Naturally the people of planet Arde must eat just as we do, but they manage a connected existence on the model of some Earthly marine animals. They must eject the indigestible portion of their food after allowing time for absorption within their digestive pouch, which has no second outlet. Their two-dimensional biology, like their geology, hydrodynamics, domestic and industrial technology, manners, travels and much more, are presented in description and diagram. All of this was learned, we are told, in a fragmentary and personal way during long days at the graphics terminal, once a group of college students had made the still-mysterious contact ended by a strange metaphysical conversion. "To talk again is of no benefit," the newly mystic Yendred said near the end.

This is original science fantasy, as much done in the popular accents of these years, all jargon language, hackers and cult events, as the famous Flatland was an authentic piece of high Victorian social irony. That work has its centenary this very year. The science behind the planiverse is much richer and more sophisticated than Edward Abbott's was, and the everyday observations are as compelling. It is a delight to read how that flat world manages construction: nails and saws, of course, are not usable, glue dominates fastenings, and plywood is a useful composite of hairlike elements. Every lowland house—dug into the Ardean ground—must have its emergency oxygen tank. A minor flood there is no mere expensive inconvenience: it cuts off the air. Sexual life in this 2-D society is examined, as contemporary fictional form requires. Its nature can be grasped through a scene in which Yendred, a night traveler in a distant industrial city, is accosted by a young female. "You my egg to buy do want? It a beautiful blue is and very large and good to sit upon." Again Earthly natural history has provided in certain crayfish a model adaptable to an intelligent species in two dimensions.

Information technology is well observed. A book is reproduced here, although at rather poor resolution. It looks exactly like the cross section of an Earth book; the line-pages hold about one sentence each, written in a dot-dash alphabet. Linear information storage is costly, and so books are few, terse and rapidly recycled, to prevent inundation by best sellers. Only true classics are enshrined in the big libraries, each shelving 1,000 volumes (areas?) or so. A second and more conservative nation across the mountains from Punizla has a different style. There it is held impious to try to improve on the old masters, so that few new works are approved, and then only after a trial period in a single copy. A

THE BEST GIN GROWS ON TREES.



The driest and most delicate gin of all, in fact, comes from the tree at the left: the family tree of James Burrough, a distiller in 19th century London.

Burrough had a basic philosophy: if you want a thing done right, do it yourself. Not a single distillation of his Beefeater Gin left his distillery till it was approved by James himself.

He wasn't much of a delegator, but he made a beautiful gin.

His philosophy has been handed down through succeeding generations of the Burrough family, who still supervise each day's run and approve each batch of Beefeater before it leaves the distillery.

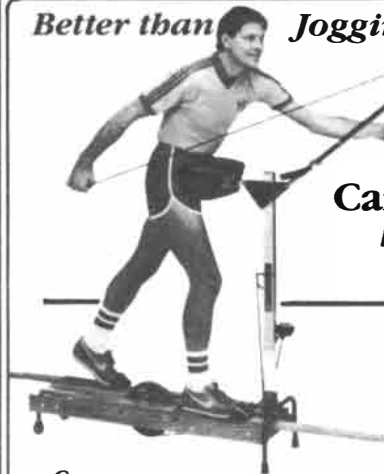
The apple never falls far from the tree. For which gin drinkers to this day are profoundly grateful.

BEEFEATER® GIN.
The Crown Jewel of England.™



IMPORTED FROM ENGLAND BY HOBSONS CORP. N.Y. N.Y. 10017-0001. GINN & CO. LONDON, ENGLAND.

Better than Jogging, Swimming, or Cycling...



NordicTrack

Jarless Total Body

Cardiovascular Exerciser

Duplicates X-C Skiing for the Best way to Fitness

Cross-country skiing is often cited by physiologists as the most perfect form of cardiovascular exercise for both men and women. Its smooth, fluid, total body motion uniformly exercises more muscles so higher heart rates seem easier to attain than when jogging or cycling. NordicTrack closely simulates the pleasant X-C skiing motion and provides the same cardiovascular endurance-building benefits—right in the convenience of your home, year 'round. Eliminates the usual barriers of time, weather, chance of injury, etc. Also highly effective for weight control.

More Complete Than Running

NordicTrack gives you a more complete work out—conditions both upper body and lower body muscles at the same time. Fluid, jarless motion does not cause joint or back problems.

More Effective Than Exercise Bikes

NordicTrack's stand-up skiing motion more uni-

formly exercises the large leg muscles and also adds important upper body exercise. Higher pulse rates, necessary for building fitness, seem easier to attain because the work is shared by more muscle mass.

Even Better Than Swimming

NordicTrack more effectively exercises the largest muscles in the body, those located in the legs and buttocks. When swimming, the body is supported by the water, thus preventing these muscles from being effectively exercised. The stand up exercising position on the NordicTrack much more effectively exercises these muscles.

A Proven, High Quality Durable Product

NordicTracks have been in production since 1976. NordicTrack is quiet, motorless and has separately adjustable arm and leg resistances. We manufacture and sell direct. Two year warrantee, 30 day trial period with return privilege.

Folds and stands on end to require only 15" x 17" storage space.

Call or write for...

FREE BROCHURE

Toll Free 1-800-328-5888

Minnesota 612-448-6987

PSI 124F Columbia Crt., Chaska, MN 55318



Show her
you're still rooting for
the home team.



De Beers

The Diamond Anniversary Ring.
A band of diamonds that says you'd marry her all over again.
A diamond is forever.

Say goodbye to the Tower of Babel

to CP/M to MS-DOS to CP/M to PC-DOS

Format, Read & Write.
It's a piece of cake
with

XENO COPY PLUS 2.0™

New! Version 2.0 formats
70 alien disks. Copy files
to and from DOS in your IBM PC.



No need for exasperating serial links or modem madness. All formatting and file transfers are easily accomplished on your PC. Runs on most PC-compatibles too. **XENO-COPY PLUS 2.0** \$149.50

Also available:

- **ADVANCED™ Option** add \$50.00
Supports 8" 96 TPI, parameter input for other formats.
- **XENO DISK™** \$379.50
Turn your PC into a disk production machine. Many features.
- **80-MATE™** \$149.50
A CP/M 2.2 emulation utility. RUN CP/M PROGRAMS UNDER DOS without an expensive co-processor board. Includes video emulation.

See your dealer or call for information.



Dept. A, 6022 West Pico Blvd.
Los Angeles, CA 90035
(213) 938-0857

IBM is a registered trademark of International Business Machines Corporation. CP/M is a registered trademark of Digital Research. MS DOS is a trademark of Microsoft Corp.

painting is also reproduced in widened form; it resembles the bar code on the supermarket package. The projection used by most painters there in the flat forces them to render many different objects with a single sequence of dots. That lends a creative ambiguity to painting, familiar to us in several periods.

An appendix to the book itself goes into more detail about the nature of that 2-D world. The most unexpected result corrects a position a couple of decades old. It was argued that a brain complex enough to support an intelligent being could not be built in two dimensions: the number of neural connections cannot be large enough without the possibility of crossing neurons. The geometric argument is simply naive. All that is needed is to send two pulses toward each other as though their fibers could cross. Naturally they cannot cross in 2-D. It is enough, however, to split each fiber and let the pulse move along each member of the new pair. A small array of relay and rectifier cells at the fourfold junction simply generates outgoing pulse trains able to mimic any two patterns entering, to relay them along the two distinct outgoing paths. That crossing, impossible passively, has simply been simulated dynamically. Axleless gears with concave teeth and clever trains of swinging cams manage to perform in two dimensions the usual tasks performed by clockwork.

Not every reader will find the campus-born fictional form of this original book an aid to the enjoyment of its ingenious 2-D world, both the scientific and the everyday. Its substance is nonetheless a delight, along with its recruitment of powerful theoretical aid to treat, if only briefly, the implications of relativity, electromagnetism, gravitation and quantum theory in two dimensions. There is wide interest in the answers, and it is pretty plain that the last word is still to come. Students ought to find challenge here for years. Readers will recognize with pleasure that the author of *The Planiverse* is now to be observed monthly in this magazine at his intellectual dance, both antic and powerful, within the theater of software.

THE ECOLOGICAL IMPLICATIONS OF **B**ODY SIZE, by Robert Henry Peters. Cambridge University Press (\$29.95). The First Law is ineluctable: what goes into an animal must come out, accounting over life and death alike, allowing for both energy and material. The thermodynamic books must balance; that allows the calculation of many partial accounts, from defecation to respiration, reproduction, locomotion and a variety of vital rates and flows. On that unshakable foundation this McGill biologist has built a brief and engaging quantitative monograph, frank, learned, painstaking and made explicitly helpful

to readers with little mathematical experience. Its overall aim is the use and reflection on one grand quantitative empirical generalization: the effect of body size on animal metabolism as a predictive tool for ecology.

After two introductory chapters, one philosophical, one mathematical, we are caught up in the substantive flow of the work. That centers on critical examination of the power laws, usually shown as empirical points tightly scattered around straight lines on log-log plots, that broadly fit relations between the size of organisms of all kinds and a wide variety of their physiological and functional features. These plots make visual some 60 tabular pages of well-documented results from the copious literature, spanning diversity such as the mass of the pancreas in a couple of hundred species of primates, the swimming speed of salmon at various water temperatures, the liver DNA content of mammals, the egg mass of crustaceans and the urine production of frogs. All are expressed as best fits to a power law in body weight. One summary graph plots 88 physiological functions, matching the empirical best-fit slope with the exponent expected from a dimensional analysis based on the single key relation between body weight and metabolism. The points cluster impressively. The data themselves remain in the 500 references cited.

One dramatic example of these relations is offered by "the gothic calculation that, on average, a full life is metered by each of 250 million breaths and 1.2 billion contractions of the heart." For mammals and birds, at least, that invariant result holds well: the fast-beating pulse of the tiny, short-lived shrew scales neatly up to the great slow rhythm of the elephant's heart, a beat maintained over most of a century. Physiological time fixes population times, entailing that a long time among plankton populations is only a moment for big fishes. Simulations that correctly scale individual growth and reproduction confirm that populations of creatures of differing size ought to vary by predictably differing clocks.

The relation between species population density and body size is less clear. It may well be that the simplest relation, according to which each species tends toward equal total mass, the head count going inversely with individual weight, is no poor guide for all life forms. The relation would imply an equal biomass in each logarithmic size class of organisms. It seems to work rather well for marine samples, where chemical or optical means can nowadays catch everything in their subtler logarithmic meshes. The particle size is measured automatically right along with the head count (if we may generalize the notion of a head to animals and plants that are

Show her she's on a winning team.



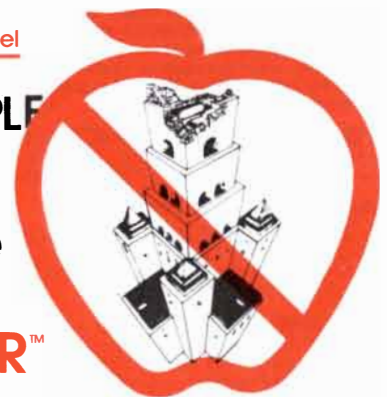
The Diamond Anniversary Ring.
A band of diamonds that says you'd marry her all over again.
A diamond is forever.

Say goodbye to the Tower of "APPLE" Babel

From IBM to APPLE to IBM to APPLE

File Transfers
are easy as pie
with

APPLE-TURNOVER™



READ, WRITE, and FORMAT Apple® II DOS 3.3 and Apple CP/M disks in your PC or most compatibles.

No need for exasperating serial links or modem madness. Leave your Apple where it is. All formatting and file-transfers are easily accomplished on your PC.

The APPLE-TURNOVER package consists of a half-sized card, friendly supporting software, and complete documentation. \$279.50.

See your dealer or call for information.

Vertex
systems inc.

Dept. A, 6022 West Pico Blvd.
Los Angeles, CA 90035
(213) 938-0857

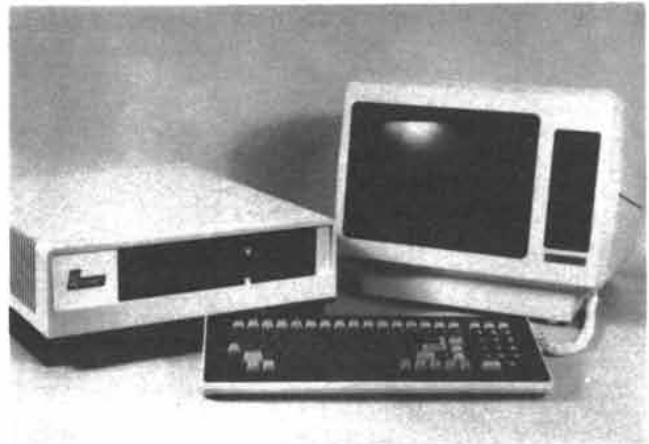
INNOVATION IN MICROCOMPUTER PRODUCTS

Apple is a registered trademark of Apple Computer Inc. IBM is a registered trademark of International Business Machines Corporation.

THE 80286 IS IN STOCK!

Get a head start on the competition! The MicroWay* MWS-286 Computer uses the most sophisticated Intel technology, including the 80286, 80287 and 8089 processors, giving you super-micro performance for as little as \$8,600. This processor combination is ideal for multi-user operating systems. It runs either Xenix-286, an Intel adaptation of Microsoft Xenix that uses special hardware to manage memory, or MicroWay's Real Time Multi-Tasking Operating System RTOS-286. MS-DOS is available for single-user work stations.

A typical multi-user system includes a seven-slot multibus backplane, 230-watt power supply, 512K bytes of high-speed, error-correcting ram, two single-board computers which manage all I/O, and a Master single-board computer - the Intel iSBC-286/10, which has six times the throughput on an 8088. The multibus architecture employed makes it possible to control up to light drives and 14 serial lines without bogging down the main cpu. Standard drives include a 19 or 40 megabyte Winchester, 360K byte floppy and five megabyte Syquest removable Winchester that is ideal for backup or storage



of data. Multi-user systems start at \$13,900. Service on all hardware is available nationwide through Intel and most configurations are available for immediate delivery.

MicroWay™ 8087 Support

87FORTRAN/RTOS™ - our adaptation of the Intel Fortran-86 Compiler generates in line 8087 code using all 8087 data types including 80-bit reals and 64-bit integers. The compiler uses the Intel large memory model, allowing code/data structures of a full megabyte, and supports overlays. Includes RTOS and support for one year **\$1350**

87PASCAL/RTOS™ is Intel's ISO-Standard Pascal with 8087-8088 exceptions. These make it possible to use all the 8087 data types directly, while generating modules in one of the three Intel Memory Models. Includes RTOS and support for one year **\$1350**

RTOS DEVELOPMENT PACKAGE includes 87FORTRAN, 87PASCAL, PL/M-86, Utilities, TX Screen Editor and RTOS. **\$2500**

REAL TIME MULTI-TASKING/ MULTI-USER EXECUTIVE - RTOS RTOS is a MicroWay configured version of iRMX-86. Includes ASM-86, LINK-86, LOC-86, LIB-86, and the ROM Hex Loader. **\$600**

OBJ → ASM™ - a multipass object module translator and disassembler. Produces assembly language listings which include public symbols, external symbols, and labels commented with cross references. Ideal for understanding and patching object modules and libraries for which source is not available **\$200**

**Information and Orders -
617-746-7341**

**University, Corporate and
Government Buyers -
617-746-7364**

PC TECH JOURNAL REVIEW: "The MicroWay package is preferable... it executes the basic operations more rapidly and MicroWay provides a free update service."

87BASIC™ includes patches to the IBM Basic Compiler and both runtime libraries for USER TRANSPARENT and COMPLETE 8087 support. Provides super fast performance for all numeric operations including trigonometrics, transcendentals, addition, subtraction, multiplication, and division **\$140**

87BASIC/INLINE™ generates inline 8087 code! Converts the IBM Basic Compiler output into an assembly language source listing which allows the user to make additional refinements to his program. Real expression evaluations run five times faster than in 87BASIC **\$200**

87MACRO™ - our complete 8087 software development package. It contains a "Pre-processor," source code for a set of 8087 macros, and an object library of numeric functions including transcendentals, trigonometrics, hyperbolics, encoding, decoding and conversions **\$150**

87DEBUG™ - a professional debugger with 8087 support, a sophisticated screen-oriented macro command processor, and trace features which include the ability to skip tracing through branches to calls and software and hardware interrupts. Breakpoints can be set in code or on guarded addresses in RAM **\$150**

FOR → BAS™ - a library of interface routines which allow MS Fortran programs to call the IBM Basic Compiler library and access features such as the RANDOM NUMBER GENERATOR, SOUND, PLAY, DRAW and SCREEN commands **\$150**

8087-3 CHIP..... \$175
including DIAGNOSTICS and 180-day warranty

64K RAM Set..... \$475⁰⁰

MATRIXPAK™ manages a **MEGABYTE!** Written in assembly language, our runtime package accurately manipulates large matrices at very fast speeds. Includes matrix inversion and the solution of simultaneous linear equations. Callable from MS Fortran 3.2, 87MACRO, 87BASIC, and RTOS each **\$150**

87/88GUIDE..... \$30

MICROSOFT FORTRAN 3.2... \$239

MICROSOFT PASCAL 3.2 \$209
These IEEE compatible compilers support double precision and the 8087

MICROSOFT C COMPILER includes Lattice C and the MS Librarian **\$350**

LATTICE C with 8087 support **\$350**

FLOAT87 for MS C **125**

SuperSoft Fortran 66 **329**

Computer Innovations C86 **345**

STSC APL★PLUS/PC **500**

TURBO PASCAL **45**

TURBO PASCAL with 8087 Support **85**

SIDEKICK **45**

HALO GRAPHICS **CALL**

GRAPHMATIC **125**

ENERGRAPHICS **295**

Professional BASIC **295**

Kidger Optical Design Program **3000**

COSMOS REVELATION **850**

SCO XENIX™ MICROSOFT **595**

Unisource VENIX/86™ THE VENIX CORP. INC. **800**

MAYNARD WS1 HARD DISK **995**

MAYNARD WS2 HARD DISK **1170**

MAYNARD ELECTRONICS Boards **CALL**

**NO CHARGE FOR CREDIT CARDS
ALL ITEMS IN STOCK
CALL FOR COMPLETE CATALOG**

**Micro
Way**

P.O. Box 79
Kingston, Mass.
02364 USA
(617) 746-7341

**You Can
Talk To Us!**

*Formerly MicroWare, Inc. - not affiliated or connected with MicroWare Systems Corporation of Des Moines, Iowa.

crudely spherical), and in one dream of this marine ecologist we may someday be able to measure the size spectrum of the organisms at any site at sea by "simply driving across it in a boat." It is an "extraordinary and audacious claim" that the amount of living matter in each logarithmic size class is a constant "from bacteria to whales." No doubt the conclusion is only tentative, but the speculation is not shown to be grossly wrong whether by land or by sea; big fierce animals are rare for good thermodynamic reason. (Deep-ocean submarines are not currently reported.)

Why do physiological rates rise with body mass as they do, by the three-fourths power? Professor Peters spends a few pages on what that "why" could mean for a scientist. Waiving all that, we do not appear to have a good explanation yet. The old workers thought animal heat loss was limited by surface area, which yields a two-thirds power law, pretty surely wrong. It must be the evolutionary engineering that fixes the result, since the biochemical properties cell by cell lie under the same overall law: the minute fuel-making mitochondria know what size body they dwell in.

An ingenious structural explanation by Thomas A. McMahon of Harvard works all too well; it derives the very result one wants, but it rests on the buckling strength of skeletal components under load, a principle less than plausible applied to the protozoa, say, that nonetheless fit the same universal relation. More abstract versions of dimensional analysis work too, but they show the same kind of flaw. One recent version, for example, firmly predicts that animals living in water, where not the weight but the displaced volume should measure the demands of aquatic locomotion, ought to follow a different scaling law. They do not. We have a good deal to learn.

The modest author ends with an appeal for more hard work. His text is an unequalled review of results in the field, although he points out it does not survey the "rich Soviet literature." He sees his book as a link between the empirical regularities and current ecological theory built on the statistical study of many individual results. The critical collection of all these striking yet thin allometric relations presents "the greatest body of quantitative general theories in biology." They differ glaringly both from our deep qualitative understanding of evolution in all its richness and from the intricate informational micro-mechanisms of molecular biology that support it. Understanding the overall criteria followed in the long, slow engineering design of the organism seems beyond our present reach, although perhaps just beyond it. The relations nonetheless stand, ready to serve a variety of powerful ecological predictions.

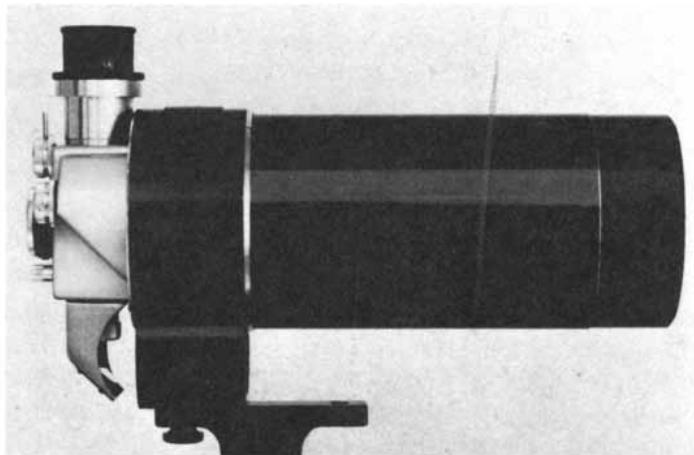
You are on the frontier . . . when it's a Questar system

Questar capabilities range from state of the art to beyond. First to explore the versatility of the great Maksutov optical system, Questar's interests have expanded from the astronomical telescope to special designs for sophisticated tracking, surveillance, and inspection instruments.

Shown here are three recently developed optical systems, all unique, photo-visual, and available for immediate delivery. Below is the QM 1, a long-range microscope which focuses at distances from 22 to 77 inches. Among its myriad uses it will let you inspect micro-circuits, observe live specimens in natural habitat, study toxic materials at safe distances. It delivers resolution below 2.5 microns.

At the right is one of a series of multifocal-length instruments, the MFL 3½. It provides tested theoretical resolution at five focal lengths, ranging from 320 mm. to 4000. One foot tall, it weighs only 6½ pounds. This design is Questar's answer to the shortcomings of the zoom lenses. It is available also in 7" and 12" apertures.

With the QM 1 and the MFL we offer full support systems, including night vision equipment and computerized analytic devices.



Third is our photo-visual Industrial 700, for the inspection of manufacturing processes. It guarantees optical perfection with theoretical resolution over the entire field, which is flat from edge to edge. Focusing capability: 10 feet to infinity; operation by a thumb and lever mechanism that permits one-hand control of focus and exposure.

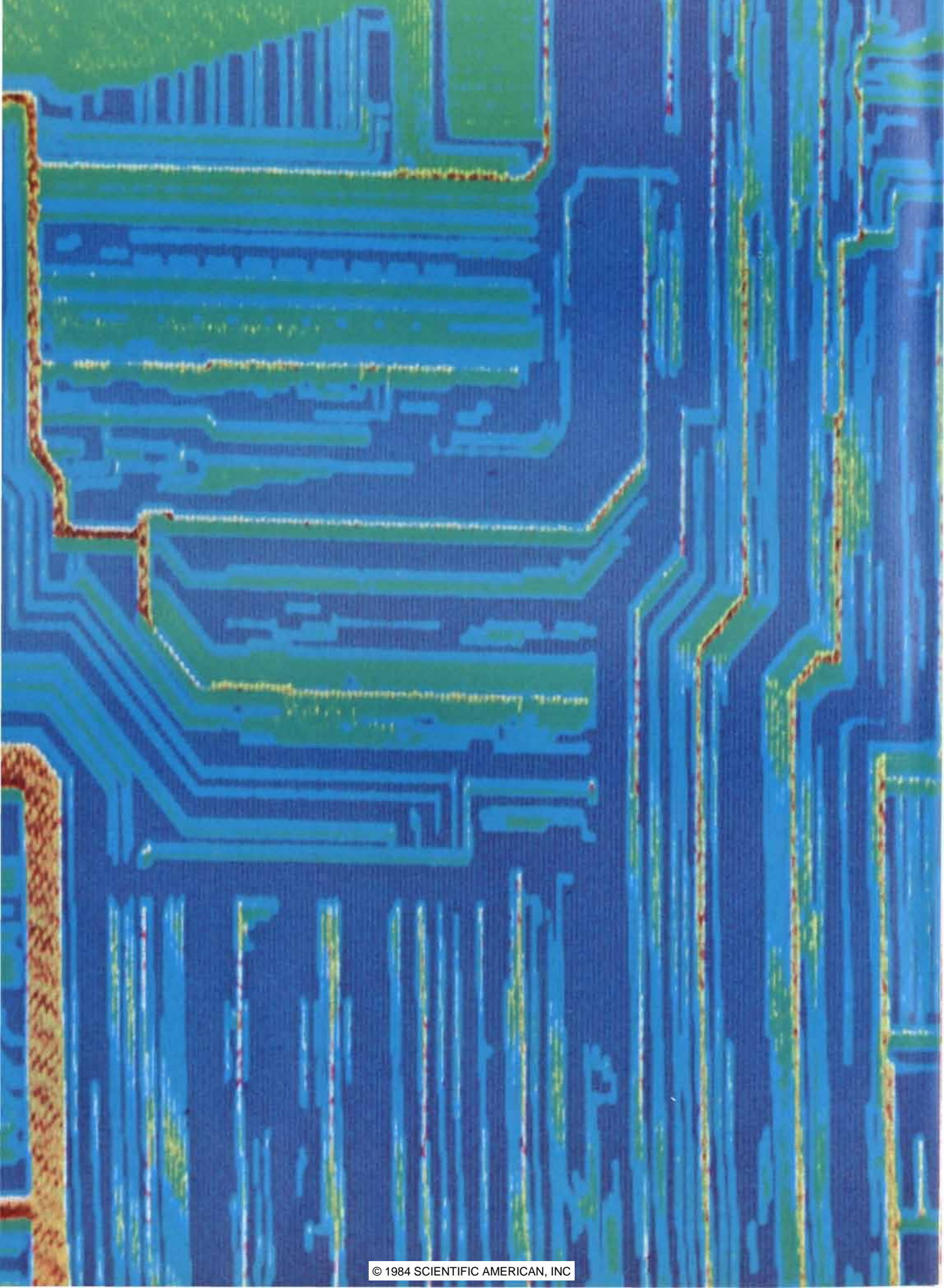
All instruments are adaptable to most SLR cameras and many video cameras. These are only a sample of Questar's ability to come up with the answers to tough problems. We invite you to send for our literature.



QUESTAR

Box 59, Dept. 209, New Hope, Pa. 18938

(215) 862-5277



Computer Software

Presenting a single-topic issue on the concepts and techniques needed to make the computer do one's bidding. It is software that gives form and purpose to a programmable machine, much as a sculptor shapes clay

by Alan Kay

Computers are to computing as instruments are to music. Software is the score, whose interpretation amplifies our reach and lifts our spirit. Leonardo da Vinci called music "the shaping of the invisible," and his phrase is even more apt as a description of software. As in the case of music, the invisibility of software is no more mysterious than where your lap goes when you stand up. The true mystery to be explored in this issue of *Scientific American* is how so much can be accomplished with the simplest of materials, given the right architecture.

The materials of computing are the tersest of markings, stored by the billions in computer hardware. In a musical score the tune is represented in the hardware of paper and ink; in biology the message transmitted from generation to generation by DNA is held in the arrangement of the chemical groups called nucleotides. Just as there have been many materials (from clay to papyrus to vellum to paper and ink) for storing the marks of writing, so computer hardware has relied on various physical systems for storing its marks: rotating shafts, holes in cards, magnetic flux, vacuum tubes, transistors and integrat-

ed circuits inscribed on silicon chips. Marks on clay or paper, in DNA and in computer memories are equally powerful in their ability to represent, but the only intrinsic meaning of a mark is that it is there. "Information," Gregory Bateson noted, "is any difference that makes a difference." The first difference is the mark; the second one alludes to the need for interpretation.

The same notation that specifies elevator music specifies the organ fugues of Bach. In a computer the same notation can specify actuarial tables or bring a new world to life. The fact that the notation for graffiti and for sonnets can be the same is not new. That this holds also for computers removes much of the new technology's mystery and puts thinking about it on firmer ground.

As with most media from which things are built, whether the thing is a cathedral, a bacterium, a sonnet, a fugue or a word processor, architecture dominates material. To understand clay is not to understand the pot. What a pot is all about can be appreciated better by understanding the creators and users of the material with meaning and to extract meaning from the form.

There is a qualitative difference between the computer as a medium of expression and clay or paper. Like the genetic apparatus of a living cell, the computer can read, write and follow its own markings to levels of self-interpretation whose intellectual limits are still not understood. Hence the task for someone who wants to understand software is not simply to see the pot instead of the clay. It is to see in pots thrown by beginners (for all are beginners in the fledgling profession of computer science) the possibility of the Chinese porcelain and Limoges to come.

Here I need spend no more time on computing's methods for storing and reading marks than molecular biology does on the general properties of atoms. A large enough storage capacity for marks and the simplest set of instructions are enough to build any further representational mechanisms that are needed, including even the simulation of an entire new computer. Augusta Ada, Countess of Lovelace, the first computer-software genius, who programmed the analytical engine that Charles Babbage had designed, understood well the powers of simulation of the general-purpose machine. In the 1930's Alan M. Turing stated the case more crisply by showing how a remarkably simple mechanism can simulate all mechanisms.

The idea that any computer can simulate any existing or future computer is important philosophically, but it is not the answer to all computational problems. Too often a simple computer pretending to be a fancy one gets stuck in the "Turing tar pit" and is of no use if results are needed in less than a million years. In other words, quantitative improvements may also be helpful. An in-

INTANGIBLE MESSAGE embedded in a material medium is the essence of computer software. Here the message is made visible in a voltage-contrast image: a scanning-electron micrograph of a small part of an Intel 80186 microprocessor. The features of the image are formed not by the conductors and transistors on the chip but by the signals passing through them. The trajectory of the secondary electrons emitted in response to the microscope beam is affected by electromagnetic fields at the surface of the chip: regions of higher voltage attract electrons, weakening the image-forming signal. The microscope beam is pulsed on only when the microprocessor is in a particular electronic state: when certain logic elements are "on." The colors of the lines indicate the voltages in metal communications lines leading to logic elements. Where a signal is traveling along a line there is a region of high voltage. The false-color image has been processed so that such regions, and thus "messages," are seen in light blue. Low-voltage regions are green, intermediate-voltage regions yellow. The red lines are conductors at ground potential, or zero volts. The micrograph was made by Timothy C. May of the Intel Corporation.

crease in speed may even represent a qualitative improvement. Consider how speeding up a film from two frames per second to 20 (a mere order of magnitude) makes a remarkable difference: it leads to the subjective perception of continuous movement. Much of the "life" of visual and auditory interaction depends on its pace.

As children we discovered that clay can be shaped into any form simply by shoving both hands into the stuff. Most of us have learned no such thing about the computer. Its material seems as detached from human experience as a radioactive ingot being manipulated remotely with buttons, tongs and a television monitor. What kind of emotional contact can one make with this new stuff if the physical access seems so remote?

One feels the clay of computing through the "user interface": the software that mediates between a person and the programs shaping the computer into a tool for a specific goal, whether the goal is designing a bridge or writing an article. The user interface was once the last part of a system to be designed. Now it is the first. It is recognized as being primary because, to novices and professionals alike, what is presented

to one's senses *is* one's computer. The "user illusion," as my colleagues and I called it at the Xerox Palo Alto Research Center, is the simplified myth everyone builds to explain (and make guesses about) the system's actions and what should be done next.

Many of the principles and devices developed to enhance the illusion have now become commonplace in software design. Perhaps the most important principle is WYSIWYG ("What you see is what you get"); the image on the screen is always a faithful representation of the user's illusion. Manipulating the image in a certain way immediately does something predictable to the state of the machine (as the user imagines that state). One illusion now in vogue has "windows," "menus," "icons" and a pointing device. The display frames called windows make it possible to present a number of activities on the screen at one time. Menus of possible next steps are displayed; icons represent objects as concrete images. A pointing device (sometimes called a mouse) is pushed about to move a pointer on the screen and thereby select particular windows, menu items or icons.

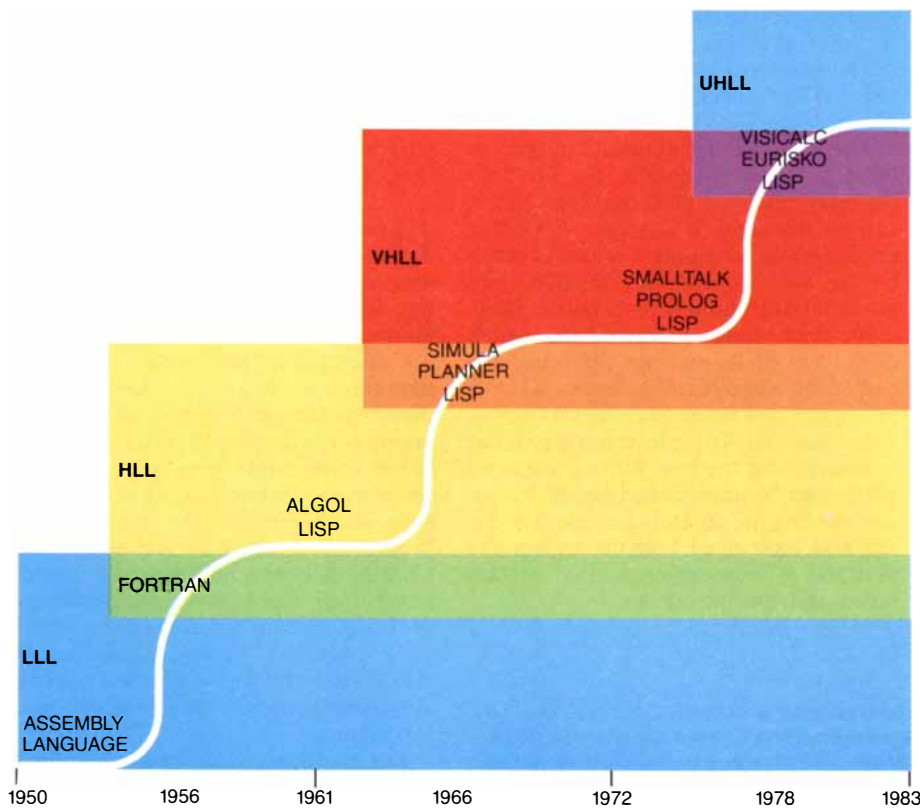
All of this has given rise to a new generation of interactive software that capi-

talizes on the user illusion. The objective is to amplify the user's ability to simulate. A person exerts the greatest leverage when his illusion can be manipulated without appeal to abstract intermediaries such as the hidden programs needed to put into action even a simple word processor. What I call direct leverage is provided when the illusion acts as a "kit," or tool, with which to solve a problem. Indirect leverage will be attained when the illusion acts as an "agent": an active extension of one's purpose and goals. In both cases the software designer's control of what is essentially a theatrical context is the key to creating an illusion and enhancing its perceived "friendliness."

The earliest computer programs were designed by mathematicians and scientists who thought the task should be straightforward and logical. Software turned out to be harder to shape than they had supposed. Computers were stubborn. They insisted on doing what was said rather than what the programmer meant. As a result a new class of artisans took over the task. These test pilots of the binary biplane were often neither mathematical nor even very scientific, but they were deeply engaged in a romance with the material—a romance that is often the precursor of new arts and sciences alike. Natural scientists are given a universe and seek to discover its laws. Computer scientists make laws in the form of programs and the computer brings a new universe to life.

Some programmers breathed too deeply of the heady atmosphere of creating a private universe. They became what the eminent designer Robert S. Barton called "the high priests of a low cult." Most discovered, however, that it is one thing to be the god of a universe and another to be able to control it, and they looked outside their field for design ideas and inspiration.

A powerful genre can serve as wings or chains. The most treacherous metaphors are the ones that seem to work for a time, because they can keep more powerful insights from bubbling up. As a result progress is slow—but there is progress. A new genre is established. A few years later a significant improvement is made. After a few more years the improvement is perceived as being not just a "better old thing" but an "almost new thing" that leads directly to the next stable genre. Interestingly, the old things and their improvements do not disappear. Strong representatives from each past era thrive today, such as programming in the 30-year-old language known as FORTRAN and even in the ancient script known as direct machine code. Some people might look on such relics as living fossils; others would point out that even a very old



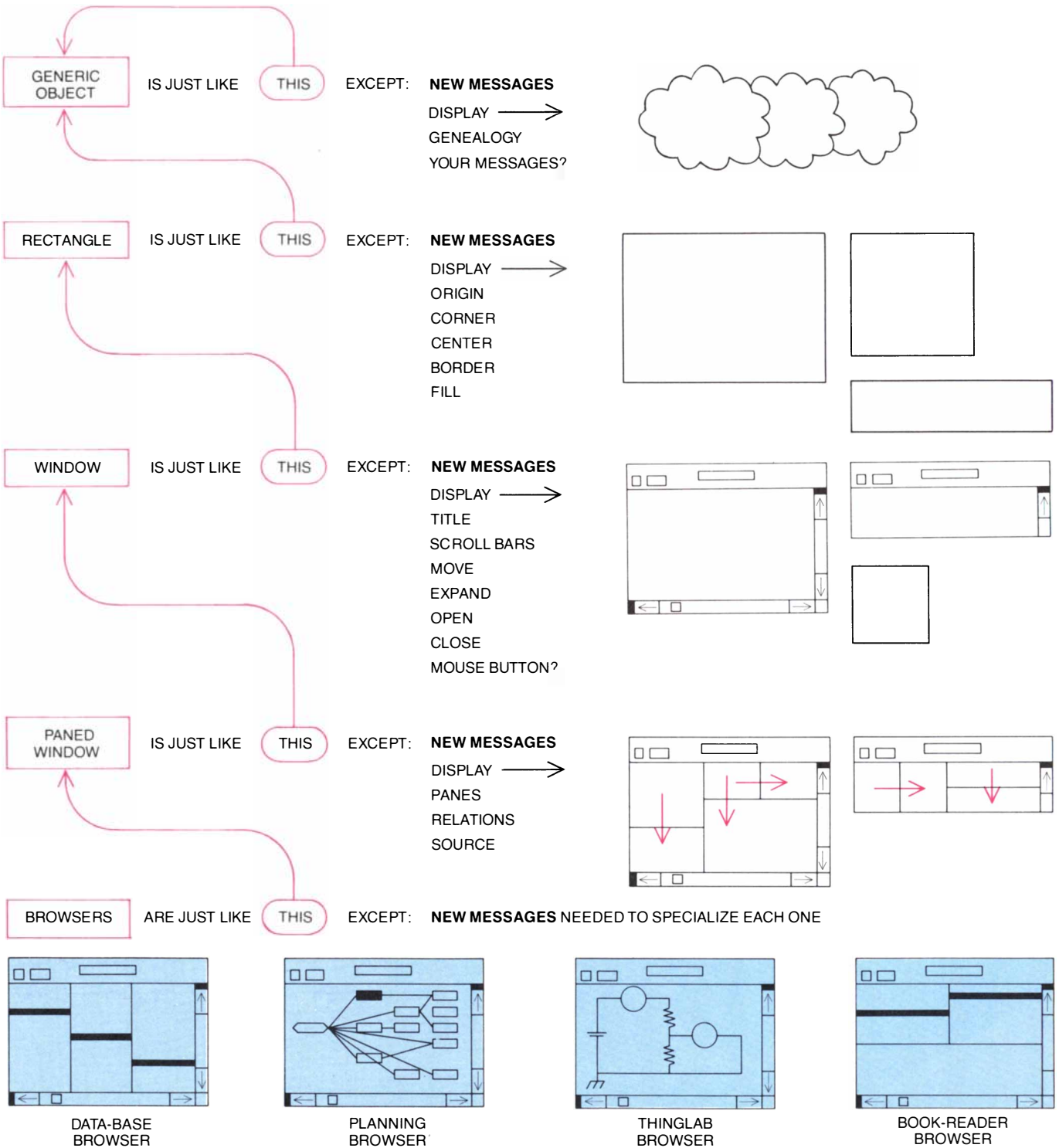
SOFTWARE GENRES succeed one another at sporadic intervals, as is shown here through the example of some programming languages. Languages are categorized rather arbitrarily by level, although the levels (colored bands) overlap. There are low-level languages (LLL), high-level languages (HLL), very-high-level languages (VHLL) and ultrahigh-level languages (UHLL). In the evolution of programming languages a genre is established (horizontal white lines), then after a few years an improvement is made (curved white lines). In time the improved language is seen to be not merely a "better old thing" but an "almost new thing," and it leads to the next stable genre. The language Lisp has changed repeatedly, each time becoming a new genre.

species might still be filling a particular ecological niche.

The computer field has not yet had its Galileo or Newton, Bach or Beethoven, Shakespeare or Molière. What it needs first is a William of Occam, who

said "Entities should not be multiplied unnecessarily." The idea that it is worthwhile to put considerable effort into eliminating complexity and establishing the simple had a lot to do with the rise of modern science and mathematics, particularly from the standpoint of creating

new aesthetics, a vital ingredient of any growing field. It is an aesthetic along the lines of Occam's razor that is needed both to judge current computer software and to inspire future designs. Just how many concepts are there really? And how can metaphor, the magical



INHERITANCE PROGRAMMING shows the power of differential description. A generic "object" (top) is displayed as a cloud. One can make a rectangle from the undifferentiated object by saying, in effect, "I want something just like that, except . . .," and then specifying such properties as the location of the origin (the upper left corner), the width, the height and so on. A further elaboration of the idea is a "window," a rectangular area of the display screen that gives a view of the output of a program. In creating a window one can allow it to "in-

herit" applicable properties of the rectangle and add new features such as scroll bars (to move the window about over the material being viewed), a title and facilities for changing the window's size and position. A more complex window with panes is made by adding new display methods to shape the panes and establish communications among them (colored arrows). Paned windows can be manipulated to make "browsers": systems enabling one to retrieve resources without remembering names. Four examples of browsers are shown (bottom).

process of finding similarity and even identity in diverse structures, be put to work to reduce complexity?

The French mathematician Jacques S. Hadamard found, in a study of 100 leading mathematicians, that the majority of them claimed to make no use of symbols in their thinking but were instead primarily visual in their approach. Some, including Einstein, reached further back into their childhood to depend on "sensations of a kinesthetic or muscular type." The older parts of the brain know what to say; the newer parts know how to say it. The world of the symbolic can be dealt with effectively only when the repetitious aggregation of concrete instances becomes boring enough to motivate exchanging them for a single abstract insight.

In algebra the concept of the variable, which allows an infinity of instances to be represented and dealt with as one idea, was a staggering advance. Metaphor in language usually accentuates the similarities of quite different things as though they were alike. It was a triumph of mathematical thinking to realize that various kinds of self-compari-

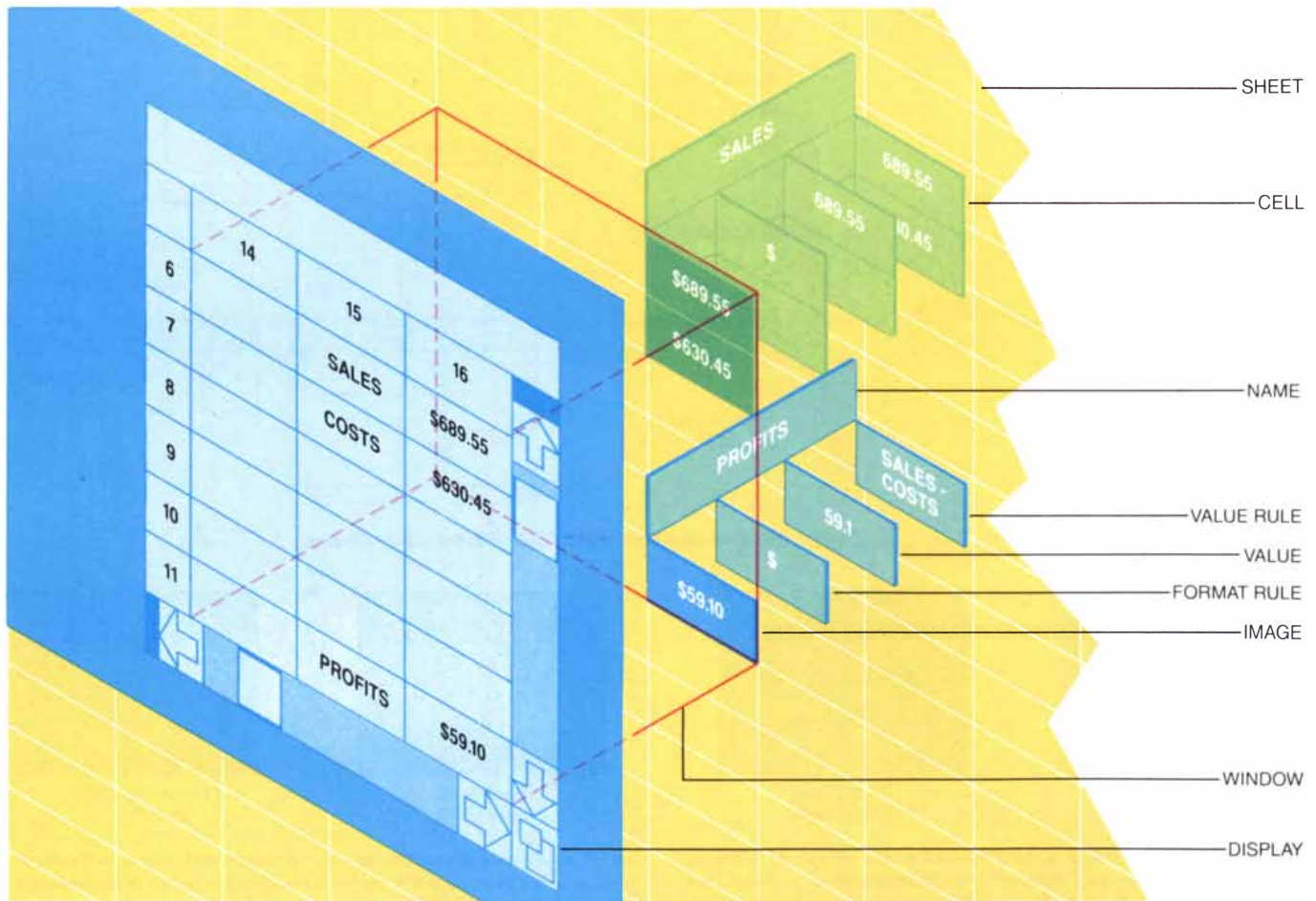
son could be even more powerful. The differential calculus of Newton and Leibniz represents complex ideas by finding ways to say "This part of the idea is like that part, except for..." The designers of computing systems have learned to do the same thing with differential models, for example with programming methods that have the property called inheritance. In recent years models based on the idea of recursion have been formulated in which some of the parts actually are the whole: a description of the entire model is needed to generate the representation of a part. An example is the fractal geometry of Benoit B. Mandelbrot, where each subpart of a structure is similar to every other part. Chaos is captured in law.

Designing the parts to have the same power as the whole is a fundamental technique in contemporary software. One of the most effective applications of the technique is object-oriented design. The computer is divided (conceptually, by capitalizing on its powers of simulation) into a number of smaller computers, or objects, each of which can be given a role like that of an actor in a play.

The move to object-oriented design represents a real change in point of view—a change of paradigm—that brings with it an enormous increase in expressive power. There was a similar change when molecular chains floating randomly in a prebiological ocean had their efficiency, robustness and energetic possibilities boosted a billionfold when they were first enclosed within a cell membrane.

The early applications of software objects were attempted in the context of the old metaphor of sequential programming languages, and the objects functioned like colonies of cooperating unicellular organisms. If cells are a good idea, however, they really start to make things happen when the cooperation is close enough for the cells to aggregate into supercells: tissues and organs. Can the endlessly malleable fabric of computer stuff be designed to form a "superobject"?

The dynamic spreadsheet is a good example of such a tissuelike superobject. It is a simulation kit, and it provides a remarkable degree of direct leverage. Spreadsheets at their best combine the



DYNAMIC SPREADSHEET is a simulation kit: an aggregate of software objects called cells that can get values from one another. The window selects a rectangular part of the sheet for display. Each cell can be imagined as having several layers behind the sheet that compute the cell's value and determine the format of the presenta-

tion. The cell's name can be typed into an adjoining cell. Each cell has a value rule, which can be the value itself or a way to compute it; the value can also be conditional on the state of cells in other parts of the sheet. The format rule converts the value into a form suitable for display. The image is the formatted value as displayed in the sheet.

genres established in the 1970's (objects, windows, what-you-see-is-what-you-get editing and goal-seeking retrieval) into a "better old thing" that is likely to be one of the "almost new things" for the mainstream designs of the next few years.

A spreadsheet is an aggregate of concurrently active objects, usually organized into a rectangular array of cells similar to the paper spreadsheet used by an accountant. Each cell has a "value rule" specifying how its value is to be determined. Every time a value is changed anywhere in the spreadsheet, all values dependent on it are recomputed instantly and the new values are displayed. A spreadsheet is a simulated pocket universe that continuously maintains its fabric; it is a kit for a surprising range of applications. Here the user illusion is simple, direct and powerful. There are few mystifying surprises because the only way a cell can get a value is by having the cell's own value rule put it there.

Dynamic spreadsheets were invented by Daniel Bricklin and Robert Frankston as a reaction to the frustration Bricklin felt when he had to work with the old ruled-paper versions in business school. They were surprised by the success of the idea and by the fact that most people who bought the first spreadsheet program (VisiCalc) exploited it to forecast the future rather than to account for the past. Seeking to develop a "smart editor," they had created a simulation tool.

Getting a spreadsheet to do one's bidding is simplicity itself. The visual metaphor amplifies one's recognition of situations and strategies. The easy transition from the visual metaphor to the symbolic value rule brings the full power of abstract models to bear almost without notice. One powerful property is the ability to make a solution generic by "painting" a rule in many dozens of cells at once without requiring users to generalize from their original concrete level of thinking.

The simplest kind of value rule makes a cell a static object such as a number or a piece of text. A more complex rule might be an arithmetic combination of other cells' values, derived from their relative or absolute positions or (much better) from names assigned to them. A value rule can test a condition and set its own value according to the result. Advanced versions allow a cell's value to be retrieved by heuristic goal seeking, so that problems for which there is no straightforward method of solution can still be solved by a search process.

The strongest test of any system is not how well its features conform to anticipated needs but how well it performs when one wants to do something the designer did not foresee. It is a question less of possibility than of perspicuity:

Can the user see what is to be done and simply go do it?

Suppose one wants to display data as a set of vertical bars whose height is normalized to that of the largest value, and suppose such a bar-chart feature was not programmed into the system. It calls for a messy program even in a high-level programming language; in a spreadsheet it is easy. Cells serve as the "pixels" (picture elements) of the display; a stack of cells constitutes a bar. In a bar displaying one-third of the maximum value, cells in the lowest third of the stack are black and cells in the upper two-thirds are white. Each cell has to decide whether it should be black or white according to its position in the bar: "I'll show black if where I am in the bar is less than the data I am trying to display; otherwise I'll show white" [*see illustration on next page*].

Another spreadsheet example is a sophisticated interactive "browser," a system originally designed by Lawrence G. Tesler, then at the Xerox Palo Alto Research Center. Browsing is a pleasant way to access a hierarchically organized data base by pointing to successive lists. The name of the data base is typed into the first pane of the display, causing the subject areas constituting its immediate branches to be retrieved and displayed in the cells below the name. One of the subject areas can be chosen by pointing to it with a mouse; the chosen area is thereby entered at the head of the next column, causing its branches in turn to be retrieved. So it goes until the desired information is reached [*see illustration on page 59*]. Remarkably, the entire browser can be programmed in the spreadsheet with just three rules.

The intent of these examples is not to get everyone to drop all programming in favor of spreadsheets. Current spreadsheets are not up to it; nor, perhaps, is the spreadsheet metaphor itself. If programming means writing step-by-step recipes as has been done for the past 40 years, however, then for most people it never was relevant and is surely obsolete. Spreadsheets, and particularly extensions to them of the kind I have suggested, give strong hints that much more powerful styles are in the offing for novices and experts alike. Does this mean that what might be called a driver-education approach to computer literacy is all most people will ever need—that one need only learn how to "drive" applications programs and need never learn to program? Certainly not. Users must be able to tailor a system to their wants. Anything less would be as absurd as requiring essays to be formed out of paragraphs that have already been written.

In discussing this most protean of media I have tried to show how effectively design confers leverage, particularly when the medium is to be shaped as a tool for direct leverage. It is clear

that in shaping software kits the limitations on design are those of the creator and the user, not those of the medium. The question of software's limitations is brought front and center, however, by my contention that in the future a stronger kind of indirect leverage will be provided by personal agents: extensions of the user's will and purposes, shaped from and embedded in the stuff of the computer. Can material give rise to mentality? Certainly there seems to be nothing mindlike in a mark. How can any combination of marks, even dynamic and reflexive marks, possibly show any properties of mentality?

Atoms also seem quite innocent. Yet biology demonstrates that simple materials can be formed into exceedingly complex organizations that can interpret themselves and change themselves dynamically. Some of them even appear to think! It is therefore hard to deny certain mental possibilities to computer material, since software's strong suit is similarly the kinetic structuring of simple components. Computers "can only do what they are programmed to do," but the same is true of a fertilized egg trying to become a baby. Still, the difficulty of discovering an architecture that generates mentality cannot be overstated. The study of biology had been under way some hundreds of years before the properties of DNA and the mechanisms of its expression were elucidated, revealing the living cell to be an architecture in process. Moreover, molecular biology has the advantage of studying a system already put together and working; for the composer of software the computer is like a bottle of atoms waiting to be shaped by an architecture he must invent and then impress from the outside.

To pursue the biological analogy, evolution can tell the genes very little about the world and the genes can tell the developing brain still less. All levels of mental competence are found in the more than one and a half million surviving species. The range is from behavior so totally hard-wired that learning is neither needed nor possible, to templates that are elaborated by experience, to a spectrum of capabilities so fluid that they require a stable social organization—a culture—if full adult potential is to be realized. (In other words, the gene's way to get a cat to catch mice is to program the cat to play—and let the mice teach the rest.) Workers in artificial intelligence have generally contented themselves with attempting to mimic only the first, hard-wired kind of behavior. The results are often called expert systems, but in a sense they are the designer jeans of computer science. It is not that their inventors are being dishonest; few of them claim for a system more than it can do. Yet the label "expert" calls up a vision that leads to dis-

illusionment when it turns out the systems miss much of what expert (or even competent) behavior is and how it gets that way.

Three developments have very low probabilities for the near future. The

first is that a human adult mentality can be constructed. The second is that the mentality of a human infant can be constructed and then "brought up" in an environment capable of turning it into an adult mentality. The third is

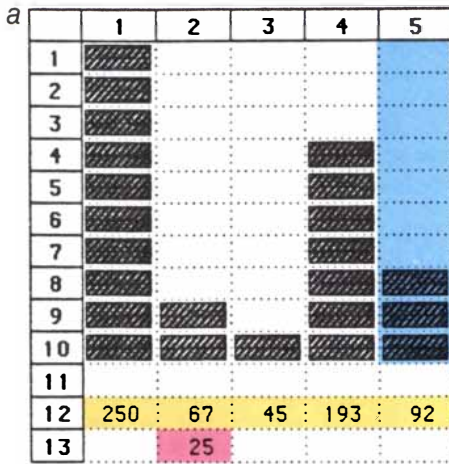
that current artificial-intelligence techniques contain the seeds of an architecture from which one might construct some kind of mentality that is genuinely able to learn competence. The fact that the probabilities are low emphatically does not mean the task is impossible. The third development is likely to be achieved first. Even before it is there will be systems that look and act somewhat intelligent, and some of them will actually be useful.

What will agents be like in the next few years? The idea of an agent originated with John McCarthy in the mid-1950's, and the term was coined by Oliver G. Selfridge a few years later, when they were both at the Massachusetts Institute of Technology. They had in view a system that, when given a goal, could carry out the details of the appropriate computer operations and could ask for and receive advice, offered in human terms, when it was stuck. An agent would be a "soft robot" living and doing its business within the computer's world.

What might such an agent do? Hundreds of data-retrieval systems are now made available through computer networks. Knowing every system's arcane access procedures is almost impossible. Once access has been gained, browsing can handle no more than perhaps 5,000 entries. An agent acting as a librarian is needed to deal with the sheer magnitude of choices. It might serve as a kind of pilot, threading its way from data base to data base. Even better would be an agent that could present all systems to the user as a single large system, but that is a remarkably hard problem. A persistent "go-fer" that for 24 hours a day looks for things it knows a user is interested in and presents them as a personal magazine would be most welcome.

Agents are almost inescapably anthropomorphic, but they will not be human, nor will they be very competent for some time. They violate many of the principles defining a good user interface, most notably the idea of maintaining the user illusion. Surely users will be disappointed if the projected illusion is that of intelligence but the reality falls far short. This is the main reason for the failure so far of dialogues conducted in ordinary English, except when the context of the dialogue is severely constrained to lessen the possibility of ambiguity.

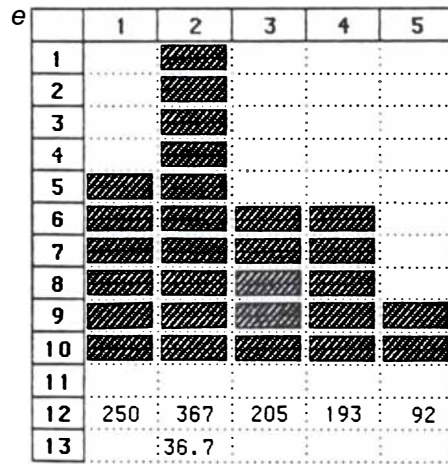
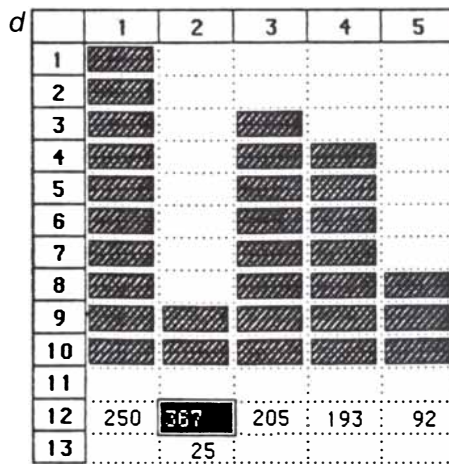
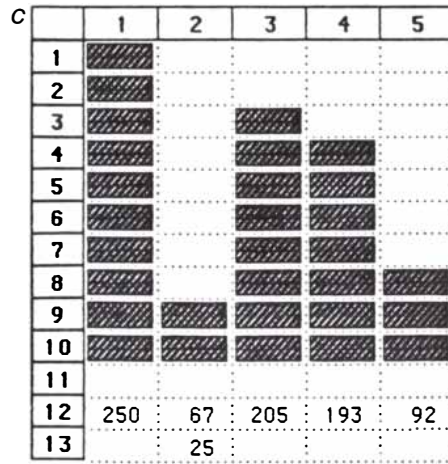
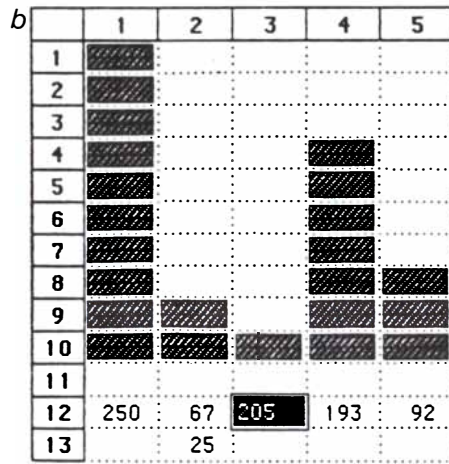
Context is the key, of course. The user illusion is theater, the ultimate mirror. It is the audience (the user) that is intelligent and can be directed into a particular context. Giving the audience the appropriate cues is the essence of user-interface design. Windows, menus, spreadsheets and so on provide a context that allows the user's intelligence to keep choosing the appropriate next step.



BAR: Value rule for each cell is "Show black if (11 - vertical location) × pixel height is less than data [horizontal location] else show white."

DATA: Value rule for each cell is either the number itself or a number fetched from some other part of the sheet.

PIXEL HEIGHT: Maximum datum ÷ 10.



BAR CHART can be constructed out of the standard materials of a spreadsheet. A bar is a column of cells, where each cell serves as a pixel, or picture element. One cell associated with each column holds the datum, or value, to be represented by the height of the corresponding bar. Within a bar all the cells are governed by the same rule. The quantity represented by the height of a single pixel is the maximum datum divided by the number of pixels in the longest bar; in chart *a* there are 10 pixels per bar and each pixel represents 25 units. Each cell shows black if its vertical position in the bar multiplied by the number of units per pixel is less than the datum for that bar; otherwise it shows white. When a new datum is entered in a column (*b*), a new bar appears in that column (*c*). If a new datum is larger than the previous maximum (*d*), the set of bars is replotted (*e*) on the basis of the new number of units per pixel, which in this case is 36.7.

Data Structures and Algorithms

They are the basic elements of every computer program. The choice of data structures and the design of procedures to manipulate them hold the key to verifying that a program does what it is meant to do

by Niklaus Wirth

Data structures and algorithms are the materials out of which programs are constructed. Furthermore, the computer itself consists of nothing other than data structures and algorithms. The built-in data structures are the registers and memory words where binary values are stored; the hard-wired algorithms are the fixed rules, embodied in electronic logic circuits, by which stored data are interpreted as instructions to be executed. Thus at the most fundamental level a computer can work with only one kind of data, namely individual bits, or binary digits, and it can act on the data according to only one set of algorithms, those defined by the instruction set of the central processing unit.

The problems people undertake to solve with the aid of a computer are seldom expressed in terms of bits. Instead the data take the form of numbers, characters, texts, events, symbols and more elaborate structures such as sequences, lists and trees. The algorithms employed to solve the problems are even more varied; indeed, there are at least as many algorithms as there are computational problems. How can a vast spectrum of problems be solved by a single machine that always acts according to fixed rules? The explanation is that the computer is a truly general-purpose device, whose nature can be transformed altogether by the program given it. The underlying principle was first set forth by John von Neumann. A stream of information is at one moment data being processed by a program, and at the next moment the same information is interpreted as a program in its own right. Hence a program is formulated in terms of familiar notions convenient to the problem at hand; then another program, called an assembler or a compiler, maps those notions onto the facilities available in the computer.

In this way it is possible to construct systems of extraordinary complexity. The programmer sets up a hierarchy of abstractions, viewing the program first in broad outline and then attending to

one part at a time while ignoring the internal details of other parts. The process of abstraction is not merely a convenience; it is a necessity, because programs of more than trivial size simply could not be created if one had to work with an undifferentiated, homogeneous mass of bits. Without higher-level abstractions a program could not be understood fully even by its creator.

Specifying the abstract data structures and algorithms of a program requires a formal notation, one in which the meaning of any legal statement is defined precisely and unambiguously. Such formal notations for programming have come to be known as languages, but the term is misleading because programming is only superficially like writing. I prefer to think of programming as the activity of designing a new machine (to be implemented with the aid of an existing, general-purpose machine). The design is specified in terms of the facilities provided by the notation, just as an electronic device is designed by drawing the symbols for basic circuit elements and their connections. If one views programming as the design of a machine,

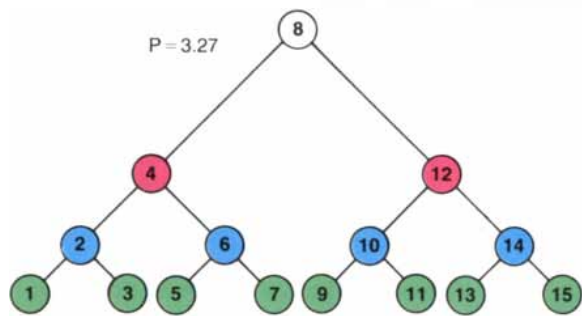
the need for precision becomes all the more obvious.

Among the facilities provided by almost all programming languages is the ability to refer to an item of data by assigning it a name, or identifier. Some of the named quantities are constants, which have the same value throughout the segment of the program in which they are defined; for example, *pi* might be assigned the value 3.14159. Other named quantities are variables, which can be assigned a new value by statements within the program, so that their value cannot be known until the program is run. The variables *diameter* and *circumference* might take on new values each time a calculation is done.

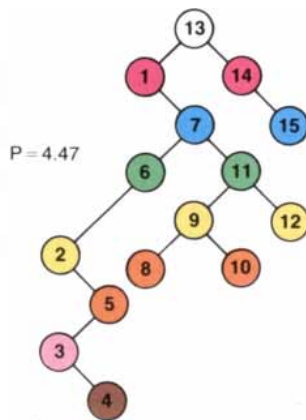
The name of a constant or a variable is a mnemonic aid to the programmer, but it has no meaning to the computer. The compiler that translates a program text into binary code merely associates each identifier with an address in memory. If an instruction calls for multiplying *diameter* by *pi*, the computer fetches whatever numbers are stored at the specified addresses and calculates the product; if the result is to become the new value of *circumference*, it is



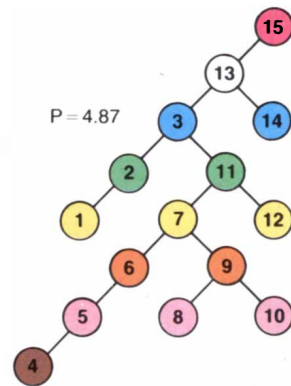
FOREST OF BINARY TREES illustrates the close interaction of data structures and algorithms. An ordered binary tree, which by convention grows from the root downward, is a data structure that is often chosen when items are to be retrieved at random from a large body of information. The tree is made up of nodes identified by a key value; in the diagrams on the opposite page the key is an integer between 1 and 15. Each node has at most two "children," which are arranged so that the child to the left invariably has a key that is smaller than the parent's key and the child to the right has one that is larger. The optimum tree is the one at the upper left: it is fully balanced, so that the average number of nodes that must be traversed to reach a given node is minimized. (The path length to each node is indicated by color according to the key above.) The other trees were generated by a random-insertion algorithm, which adds a node at the first position a key is allowed to occupy without moving any other nodes to maintain the balance of the tree. A more elaborate algorithm could reduce the average path length somewhat, but the algorithm itself would then be more complicated. The random-insertion algorithm and the benefits of tree-balancing are explored in the illustrations on pages 68 and 69.



P=3.27

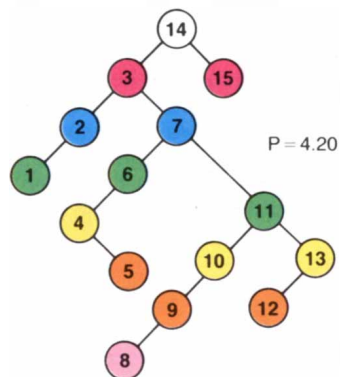


P=4.47

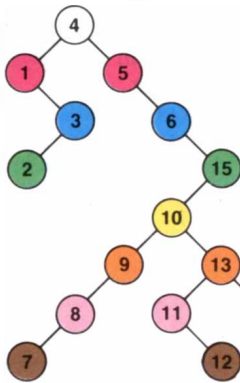


P=4.87

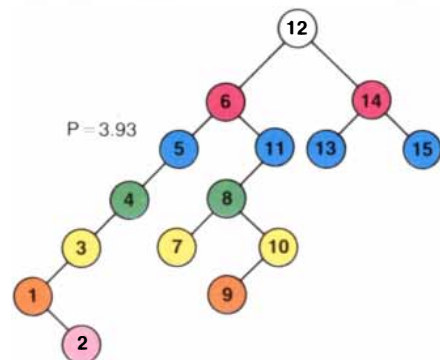
13 14 1 7 11 9 12 6 2 10 8 5 3 15 4 15 13 3 2 11 7 9 6 5 4 10 12 8 14 1



P=4.20



P=4.87

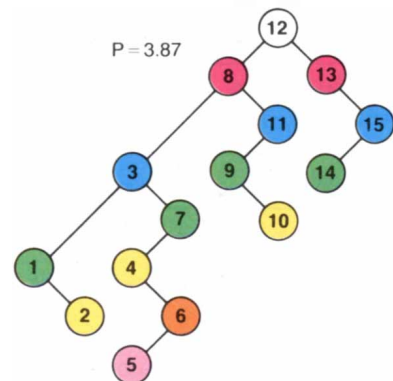


P=3.93

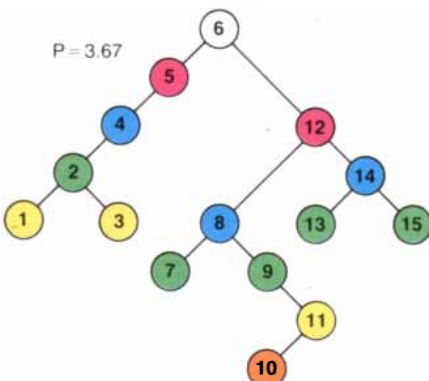
14 3 7 11 10 9 13 6 8 2 12 4 15 5 1

4 1 5 3 6 15 10 2 9 13 11 8 7 14 12

12 14 6 5 15 11 4 8 7 3 1 13 10 2 9



P=3.87

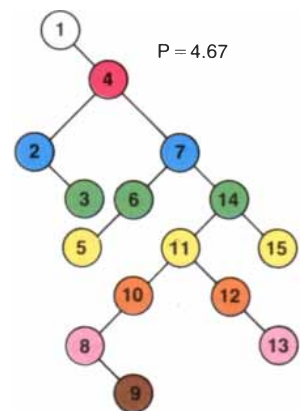


P=3.67

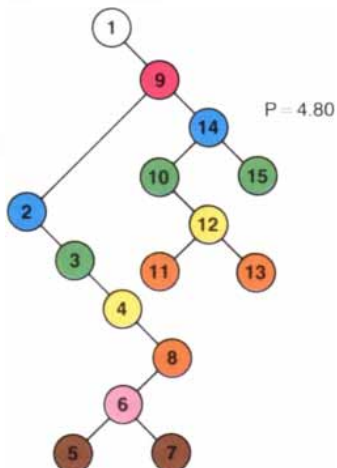
12 13 8 11 3 7 9 1 4 15 10 6 2 14 5

6 5 12 14 8 9 4 2 13 11 7 3 10 1 15

13 8 15 7 5 14 11 1 9 2 6 12 3 4 10



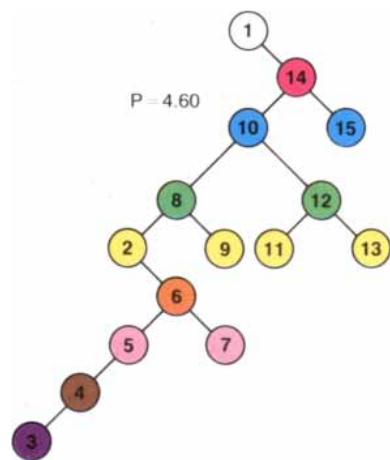
P=4.67



P=4.80

1 4 7 14 2 15 6 11 12 10 5 8 3 9 13

1 9 2 3 14 4 15 8 10 12 6 13 5 11 7



P=4.60

1 14 10 8 2 6 12 7 9 5 11 4 3 15 13

stored in memory at the address corresponding to that label.

The naming of constants and variables in programming is similar to the use of symbolic expressions in algebra, but for a computer to handle the process some additional information must be supplied. The information gives the "type" of each named quantity. A person working a problem by hand has an intuitive grasp of data types and the operations that are valid for each type; it is known, for example, that one cannot take the square root of a word or capitalize a number. One reason such distinctions are easily made is that words, numbers and various other symbols are represented quite differently. For the computer, however, all types of data ultimately resolve into a sequence of bits, and the type distinctions must be made explicit.

Suppose in the course of some operation the seven-bit binary value 1010011 has been read into a register in the central processing unit of a computer. How is the value to be interpreted? One possi-

bility is that it represents a cardinal, or counting, number, in which case the equivalent in decimal notation would be 83. In many programming languages the value could also represent a signed integer equal to decimal -45. The same binary data could encode not a number but a character; in the American Standard Code for Information Interchange (ASCII) binary 1010011 specifies the letter *S*. Several other possibilities exist. (Indeed, the binary code might not be data at all but an instruction to the computer; its interpretation would then depend on the particular processor.)

The data types recognized by common programming languages include cardinal numbers, integers, real numbers (approximated as decimal fractions), sets, characters and strings of characters. Information on each variable's type is needed not only to interpret the binary representation but also to set aside the correct amount of space in storage. In many modern computer systems a single character is allocated eight bits, or one byte, of mem-

ory, whereas a cardinal or an integer might be given two or four bytes and a real number might take up as many as eight bytes.

In some programming languages the compiler infers the type of a constant or a variable from certain clues in the way the assigned value is written; the presence of a decimal point, for example, might indicate a real number. Other languages require the programmer to state each variable's type explicitly. Explicit declaration of data types may seem bothersome if it can be avoided, but it has an important advantage. Although the value of a variable may change repeatedly as a program executes, its type should never change; hence the compiler can check to make certain that all operations carried out on the variable are consistent with the type declaration. Such consistency checking can be done by examining the program text, and so it holds for all possible computations specified by the program. A "trial run" of the compiled program, on the other

TYPE	DISPLAYED FORM	INTERNAL REPRESENTATION
CARDINAL	83	00000000 00000000 00000000 01010011 MAGNITUDE
INTEGER	-83	1 11111111 11111111 11111111 10101101 SIGN MAGNITUDE
REAL	83.0	0 1000111 10100110 00000000 00000000 00000000 00000000 00000000 WEIGHTED EXPONENT SIGN OF MANTISSA NORMALIZED MANTISSA
SET	0,1,4,6	00000000 00000000 00000000 01010011 MEMBERS OF SET
CHARACTER	S	01010011 ASCII CODE
STRING	SET 83	S E T <SPACE> 8 3 00000110 01010011 01000101 01010100 00100000 00111000 00110011 ASCII CODES OF CHARACTERS NUMBER OF CHARACTERS

ELEMENTARY DATA TYPES are given a predetermined internal representation by the compiler program that translates statements in a programming language. The type of a variable must be specified so that the compiler can allocate space in storage and put the data in the correct format. In the data representations shown here a cardinal number or an integer is stored in 32 bits, or four bytes. A real number is accorded 64 bits and is represented in scientific notation, with

an exponent, a mantissa and a sign bit. A set can be represented as a string of bits, where a 1 indicates that an element is a member of the set and a 0 indicates that it is not. Characters are generally given seven- or eight-bit values specified by the ASCII standard code. A string of characters consists of the individual character codes with an additional byte to give the length. The division of the binary values into groups is for readability; the gaps would not appear in memory.

hand, can verify correct operation only for the specific input values tried.

The notion of a data type has been extended, primarily through the programming language Pascal, to encompass the description of data structures. A structured variable is one with multiple elements, or components, that can nonetheless be referred to as a single entity. In a calendar, for example, one must be able to specify a particular date, but there should also be a way to refer to entire months and years. The type declaration for a structured variable establishes the number of elements making up the variable, it enables the compiler to allocate the necessary storage for them and it provides information about the intended method of access to them.

If all the elements are of the same type (and so have the same storage requirement), the structured variable is said to be homogeneous, and it can be declared as an array. A structured variable to be given the name *Sept* and consisting of 30 cardinal numbers might have the following type declaration:

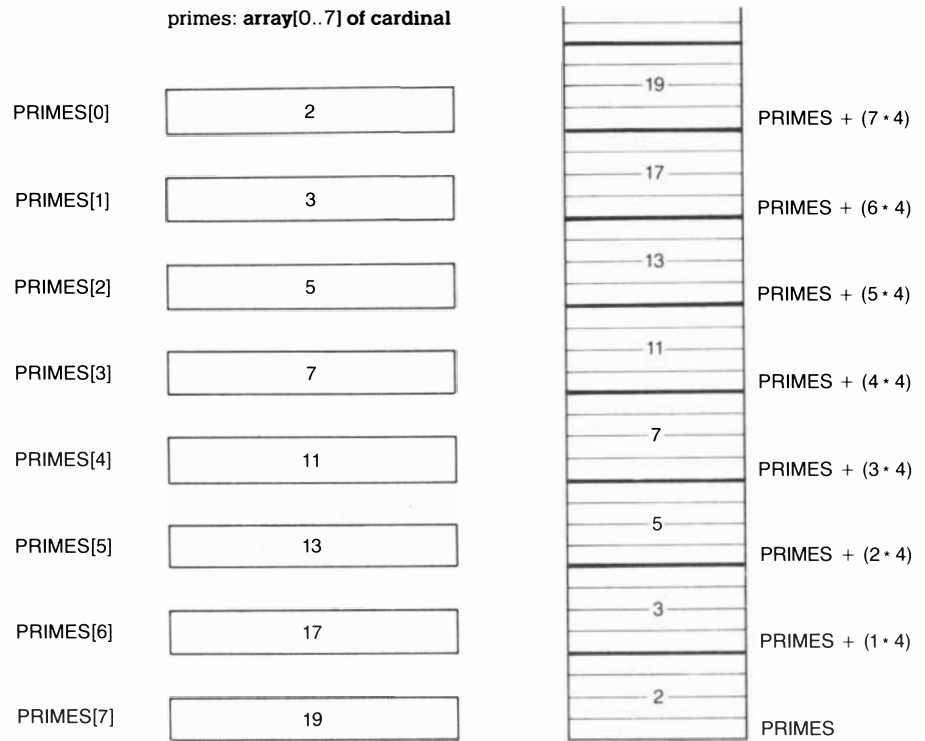
Sept: array[1..30] of cardinal

In an array individual elements are readily identified by means of a computed index, which represents a position in the sequence of elements. The address of the fifth element, for example, is simply the address of the first element plus five times the size of an element.

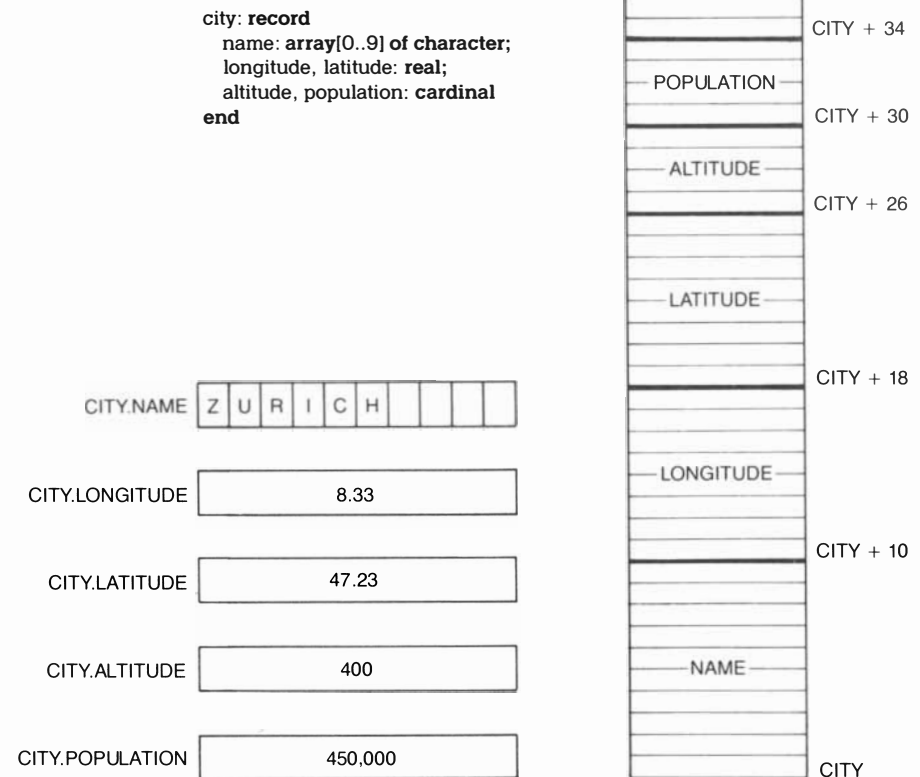
If the elements of the structured variable are not all of the same type, this simplicity of access is lost. On the other hand, elements that differ in type are likely to differ in other ways as well, and there is less need to refer to them by means of a computed index. Instead each element is given its own identifier. The entire structured variable is called a record and the elements are called fields. A record designed to hold information on cities is shown in the bottom illustration at the right. The record is referred to by the variable name *city*; individual fields are designated *city.name*, *city.population* and so on.

Another fundamental structured data type is the set. It is of use where the value of an element is not of immediate interest and only its presence or absence matters. If a variable named *primes* is declared to be a set of cardinal numbers, a set-membership operation can be defined that will yield the logical value True if a number is a member of the set and the value False otherwise. Sets can be efficiently implemented and manipulated; each element of the set is represented by a single bit, where the presence of the element is indicated by a 1 and its absence by a 0.

Arrays, records and sets are called basic structures. In many contexts more complicated structures are needed, but



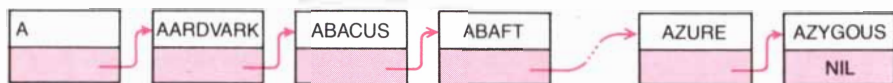
ARRAY OF DATA consists of a specified number of elements that are all of the same elementary data type. The compiler can allot the same amount of space to each element, and so a particular element can be found by a simple calculation of its address from its index value. Here an array of eight elements has been given the name *primes*, which also represents the address in memory at which the array begins. The elements are cardinal numbers with a storage requirement of four bytes, so that the address of any element in the array is calculated by multiplying the element's index by 4 and adding that number of bytes to the address of *primes*.



RECORD STRUCTURE holds heterogeneous information. Here a record named *city* has been defined to include five fields, including a string of characters, two real numbers and two cardinal numbers. A particular field can be accessed by giving the record name and the field name separated by a period. Because different types of information have different storage requirements, the fields are not all the same length. For each field the compiler must record an offset value: the distance in memory from the start of the record as a whole to the start of the field.

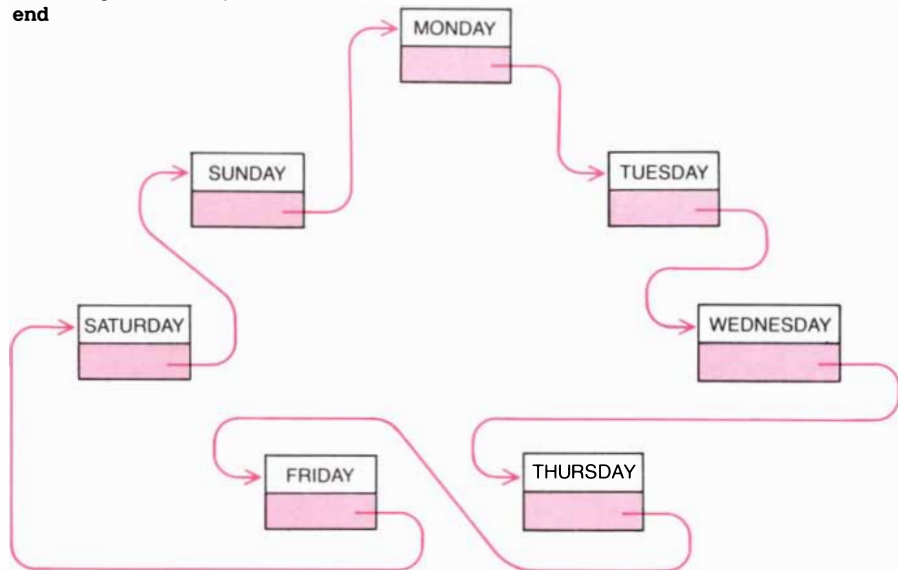

```

type word =
  record
    spelling: array[0..20] of character;
    next: pointer to word
  end
end
    
```



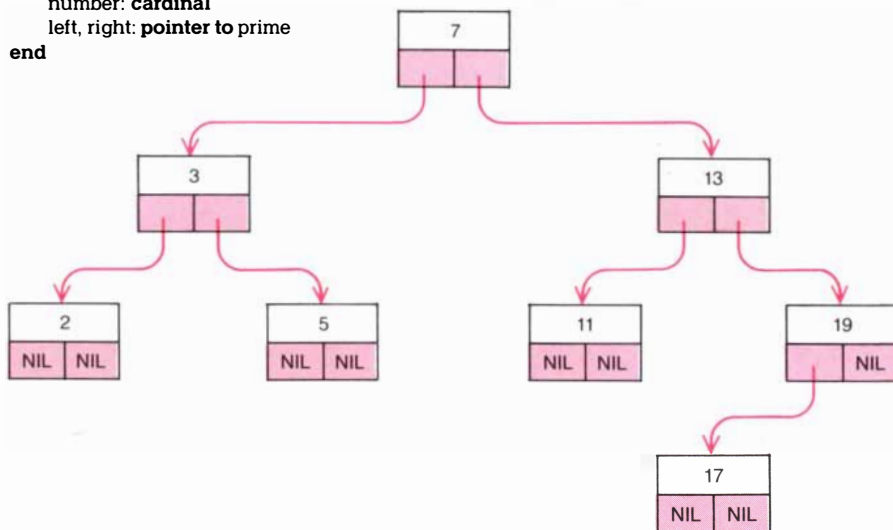
```

type day =
  record
    weekday: (MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
              SATURDAY, SUNDAY)
    next: pointer to day
  end
end
    
```



```

type prime =
  record
    number: cardinal
    left, right: pointer to prime
  end
end
    
```



DYNAMIC DATA STRUCTURES can expand or contract or even be reorganized under the direction of the algorithmic part of the program. The structures are made up of nodes, which are generally records, that include pointers to other nodes (*color*); a pointer that points to nothing is given the special value *nil*. The simplest dynamic structure is the linked list, where each node has a pointer to the next one. The example shown here might form part of a dictionary. A list can be transformed into a ring by making the last element point to the first one. In a binary tree each node has two pointers, which give the addresses of the left and right subnodes.

rather than trying to invent notations for all of them it is preferable to introduce a general facility for building arbitrary structures. Whereas the structure of an array, of a record or of a set remains constant during the execution of a program, the more elaborate structures can be allowed to grow, shrink or alter their topology. The basic structures are static, the derived ones dynamic.

Because the size of a dynamic structure is subject to change, it cannot be specified by an invariant declaration; it must be defined by the algorithmic part of the program. For the same reason the program, rather than the compiler, must allocate storage space for the structure. This would appear to be a treacherous situation, since the correctness of the structure cannot be checked during compilation. One property of the structure does remain fixed, however, and therefore can be declared in advance, namely the types of the elements of which the structure is ultimately composed. Only the number of elements and the connections between them can vary during execution.

The mechanism for creating dynamic structures consists of a way to generate basic components called nodes and a way to establish connections between nodes. The nodes are generally records; the connections are defined by variables called pointers. As the name suggests, a pointer points to an element of the dynamic structure; it can also be assigned the special value *nil*, in which case it points to nothing.

One dynamic structure that can readily be created out of nodes and pointers is a linked list. Each entry in the list is a record, one of whose fields is a pointer to the next record. The pointer in the last entry has a value of *nil*. To add a record the program allocates the necessary space in storage and changes the last pointer to point to the new data element. Making the last element point back to the first one converts the list into a ring. A tree is created by having each node include pointers to all its subnodes. Examples of several dynamic structures are shown in the illustration at the left.

The implementation of pointers is straightforward: the value of a pointer (unless it is *nil*) is the address of the node it points to. Why not simply call a pointer an address? The distinction is worth preserving because a pointer is declared to point to a variable of a known type, whereas an address could specify any location in memory. The compiler can thereby ascertain that each pointer is associated with an object of the right type.

A data structure is an essentially spatial concept; it can be reduced to a map of how information is organized in the computer's memory. An algorithm is the corresponding procedural element

in the structure of a program; it is a recipe for computation.

The first algorithms were invented to solve numerical problems such as multiplying numbers, finding the greatest common divisor, calculating trigonometric functions and so on. Today non-numerical algorithms are of equal importance; they have been devised for tasks such as finding the smallest element in a sequence, searching for a given word in a text, scheduling events and sorting data into some specified order.

Nonnumerical algorithms operate on data that are not necessarily numbers; moreover, no deep mathematical concepts are needed to design or understand them. It does not follow, however, that mathematics has no place in the study of such algorithms; on the contrary, rigorous, mathematical methods are essential in finding the best solutions to nonnumerical problems, in proving the correctness of the solutions and in determining their effectiveness. Programming remains a highly mathematical discipline. The roles have been exchanged, however: whereas computing methods were once employed to solve mathematical problems, mathematical methods are now applied to the solution of computing problems.

Here I shall not attempt to give a general survey of the various categories of algorithms or even to discuss methods for constructing new algorithms. Instead I shall illustrate the modern approach to reasoning about algorithms. Given an algorithm that purports to solve some problem, how can one understand the algorithm, and in particular how can one gain confidence in its correctness without resorting to a computer to "test a few cases"?

If an algorithm is viewed as a temporal series of operations, a fundamental issue is how the flow of operations is controlled. When the execution of a program has reached a particular statement, how does the computer determine which statement to execute next?

It has been shown that just three controlling principles are sufficient to describe any algorithm. The first principle is so obvious that it is often overlooked: it is the notion of sequence. Unless the computer is instructed otherwise, it executes the statements of a program sequentially. The second principle is conditional execution, which is generally designated in the program text by an "if...then" construction. In the statement "if *B* then *S*," *B* is a Boolean expression, one with a value of either True or False, and *S* is any statement or group of statements. *B* is evaluated, and if it is True, *S* is executed; otherwise *S* is skipped.

The third principle is repetition, which can be indicated by a "while...do" construction. "While *B* do *S*" tests the value of *B* and, if it is True, ex-

```

a := x; b := y; c := 0;
while b ≠ 0 do
  {assertion: a•b + c = x•y}
  {assertion: b ≠ 0}
  b := b - 1; c := c + a
end
{assertion: a•b + c = x•y and b = 0 yields c = x•y}

```

OPERATION	EVALUATION	LOOP INVARIANT	GUARD
x := 7; y := 13			
a := 7; b := 13; c := 0	a = 7, b = 13, c = 0	7 • 13 + 0 = 7 • 13	b ≠ 0
b := b - 1; c := c + a	a = 7, b = 12, c = 7	7 • 12 + 7 = 7 • 13	b ≠ 0
b := b - 1; c := c + a	a = 7, b = 11, c = 14	7 • 11 + 14 = 7 • 13	b ≠ 0
b := b - 1; c := c + a	a = 7, b = 10, c = 21	7 • 10 + 21 = 7 • 13	b ≠ 0
b := b - 1; c := c + a	a = 7, b = 9, c = 28	7 • 9 + 28 = 7 • 13	b ≠ 0
b := b - 1; c := c + a	a = 7, b = 8, c = 35	7 • 8 + 35 = 7 • 13	b ≠ 0
b := b - 1; c := c + a	a = 7, b = 7, c = 42	7 • 7 + 42 = 7 • 13	b ≠ 0
b := b - 1; c := c + a	a = 7, b = 6, c = 49	7 • 6 + 49 = 7 • 13	b ≠ 0
b := b - 1; c := c + a	a = 7, b = 5, c = 56	7 • 5 + 56 = 7 • 13	b ≠ 0
b := b - 1; c := c + a	a = 7, b = 4, c = 63	7 • 4 + 63 = 7 • 13	b ≠ 0
b := b - 1; c := c + a	a = 7, b = 3, c = 70	7 • 3 + 70 = 7 • 13	b ≠ 0
b := b - 1; c := c + a	a = 7, b = 2, c = 77	7 • 2 + 77 = 7 • 13	b ≠ 0
b := b - 1; c := c + a	a = 7, b = 1, c = 84	7 • 1 + 84 = 7 • 13	b ≠ 0
b := b - 1; c := c + a	a = 7, b = 0, c = 91	7 • 0 + 91 = 7 • 13	b = 0

```

a := x; b := y; c := 0;
while b ≠ 0 do
  {assertion: a•b + c = x•y}
  {assertion: b ≠ 0}
  c := c + a • (b MOD 10);
  a := 10 • a;
  b := b DIV 10
end
{assertion: a•b + c = x•y and b = 0 yields c = x•y}

```

OPERATION	EVALUATION	LOOP INVARIANT	GUARD
x := 7; y := 13			
a := 7; b := 13; c := 0	a = 7, b = 13, c = 0	7 • 13 + 0 = 7 • 13	b ≠ 0
c := c + a • (b MOD 10)	c = 0 + 7 • 3 = 21		
a := 10 • a	a = 10 • 7 = 70		
b := b DIV 10	b = 13 DIV 10 = 1	70 • 1 + 21 = 7 • 13	b ≠ 0
c := c + a • (b MOD 10)	c = 21 + 70 • 1 = 91		
a := 10 • a	a = 10 • 70 = 700		
b := b DIV 10	b = 1 DIV 10 = 0	700 • 0 + 91 = 7 • 13	b = 0

```

a := x; b := y; c := 0;
while b ≠ 0 do
  {assertion: a•b + c = x•y}
  {assertion: b ≠ 0}
  if Odd (b) then c := c + a end;
  a := 2 • a;
  b := b DIV 2
end
{assertion: a•b + c = x•y and b = 0 yields c = x•y}

```

OPERATION	EVALUATION	LOOP INVARIANT	GUARD
x := 7; y := 13			
a := 7; b := 13; c := 0	a = 7, b = 13, c = 0	7 • 13 + 0 = 7 • 13	b ≠ 0
if Odd (b) then c := c + a end	c = 0 + 7 = 7		
a := 2 • a	a = 2 • 7 = 14		
b := b DIV 2	b = 13 DIV 2 = 6	14 • 6 + 7 = 7 • 13	b ≠ 0
if Odd (b) then c := c + a end	c = 7		
a := 2 • a	a = 2 • 14 = 28		
b := b DIV 2	b = 6 DIV 2 = 3	28 • 3 + 7 = 7 • 13	b ≠ 0
if Odd (b) then c := c + a end	c = 7 + 28 = 35		
a := 2 • a	a = 2 • 28 = 56		
b := b DIV 2	b = 3 DIV 2 = 1	56 • 1 + 35 = 7 • 13	b ≠ 0
if Odd (b) then c := c + a end	c = 35 + 56 = 91		
a := 2 • a	a = 2 • 56 = 112		
b := b DIV 2	b = 1 DIV 2 = 0	112 • 0 + 91 = 7 • 13	b = 0

DEVELOPMENT OF AN ALGORITHM for multiplying two cardinal numbers proceeds through three stages. The first algorithm (*top*) employs the method of repeated addition; its correctness can be verified through an assertion called a loop invariant, which must remain true at all stages of the calculation. A more efficient method (*middle*) is the one people generally use in doing multiplication by hand; it calls for dividing the multiplier by 10 instead of decrementing it by 1. In the example of 7×13 the faster algorithm reduces the number of passes through the loop from 13 to three. Because a digital computer uses binary and not decimal arithmetic, an algorithm based on division by 2 (*bottom*) is still faster in spite of requiring more repetitions.

ecutes S ; the two steps are repeated until a test of B yields False. In most cases a statement within S eventually changes the value of B , so that the loop does not continue indefinitely. In both the conditional and the looping constructs B has the function of a “guard,” an expression that allows S to be executed only if the condition defined by B is satisfied.

Consider an algorithm for multiplying two cardinal numbers, x and y , by means of repeated addition. A formal statement of the algorithm is given in the illustration on the preceding page. The first step is to set up three variables: the multiplicand a , the multiplier b and

a partial sum c , which ultimately becomes the product. The variables are given their initial values in the three statements $a := x$, $b := y$ and $c := 0$. Here the symbol “:=” is an assignment operator; whereas an equal sign constitutes an assertion of equality, the assignment operator actually creates a condition of equality, that is, it assigns to the symbol on the left the value of the expression on the right. It can be read as “Let a become equal to x .”

The heart of the algorithm is a *while* loop in which the guard is the statement $b \neq 0$. As long as b remains greater than 0 two operations are carried out repeatedly. In the first operation b is decremented by 1; in the second opera-

tion a is added to the current value of c . The operations are expressed formally in two assignment statements:

$$b := b - 1; c := c + a .$$

The intent of the algorithm is obvious, and the procedure is straightforward. With c initially equal to 0, a is added to it b times, which directly implements the definition of multiplication.

How can one be sure, however, that a program written to express this intuitively clear idea actually embodies the correct algorithm? One approach is to enter the program into a computer, compile it and test a few cases. That method can never lead to absolute confidence in the program’s correctness, simply because the number of potential test cases is infinite. A better answer is to include in the program “assertions,” or statements of conditions that must be true (if the algorithm is correct), no matter what path the computation has taken up to that point. In this case the assertion is a “loop invariant,” a statement that holds no matter how many times the loop has been executed.

What assertion about the quantities in the multiplication problem remains true throughout the execution of the program? Since a and b are initially set equal to x and y , it is clear that at the outset $a * b$ must be equal to $x * y$. (The asterisk signifies multiplication, a convention in many programming languages.) Similarly, when the calculation is finished, c is taken as the product and so c must then also be equal to $x * y$. In the various stages between the start and the end of the procedure, b is decreased by 1 each time c is increased by a . From this analysis it follows that the equation $a * b + c = x * y$ holds throughout the calculation. That assertion is therefore the essential invariant of the *while* loop; together with the condition for termination ($b = 0$) it establishes the desired result ($c = x * y$).

In this instance the truth of the assertion employed to confirm the correctness of the program is scarcely more obvious than the correctness of the program statements themselves. The algorithm, however, is a simple one. The power of assertions as a means of verifying program correctness becomes apparent when the algorithm is refined in order to make it more efficient. The assertion formulated in the simplest case remains valid even as the program becomes more complicated.

The loop in the algorithm for multiplication by repeated addition must be executed y times. There are much faster methods. The algorithm for multiplication taught in primary school is an example; it is based on the same principle, but b is reduced in larger steps. Instead of being decremented by 1, it is divided by 10, a particularly easy operation in

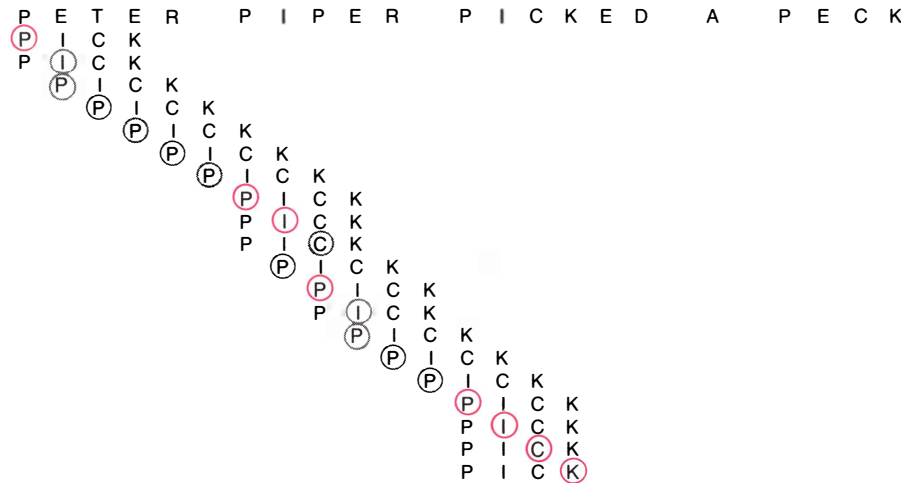
```

text: array[0..M-1] of character
j =      0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
text[j] = P E T E R   P I P E R   P I C K E D   A   P E C K

word: array[0..N-1] of character
i =      0  1  2  3
word[i] = P I C K

i := 0; j := 0;
while (i < N) and (j < M - N) do
  i := 0
  while (i < N) and word[i] = text[j + i] do i := i + 1 end;
  if i < N then j := j + 1 end
end

```



- P: $\forall i: (0 \leq i < N): \text{word}[i] = \text{text}[j + i]$
- Q: $\forall k: (0 \leq k < j): \exists i: \{(0 \leq i < N): \text{word}[i]\} \neq \text{text}[k + i]$
- R: P and Q and ($J \geq m - n$ or $i = n$)

SEARCH FOR A WORD IN A TEXT is done by a series of letter-by-letter comparisons. Both the text and the word are declared to be arrays of characters; in this case the text has 25 letters and the word has four. The first letter of the word is compared with the first letter of the text; they happen to match (indicated here by a colored circle), and so the second letters are compared. This time the letters differ (indicated by a gray circle); the word is thus moved one character position to the right and the testing of letter pairs begins anew at the start of the word. Only when all the letters match has the word been found. The conditions to be satisfied by the algorithm are specified by the three propositions in predicate calculus given at the bottom of the illustration. The first assertion (P) states that when the word is aligned at position j in the text array, for all values of the word-array index i , the letter $\text{word}[i]$ is the same as the letter $\text{text}[j + i]$. The second proposition (Q) states that there is no matching sequence for any smaller value of j ; thus the first occurrence of the word in the text must be found. The third proposition, which must hold at the end of the search, states that P and Q will both be satisfied and either i will have the value n (indicating that a match was found at position j) or j will have a value larger than that of any possible matching sequence (indicating that the word is not present in the text).

the decimal system. Indeed, in this context one generally does not even think of the process as division but rather as simply separating the multiplier into its component digits. The isolation of digits can be done in algorithmic terms by applying two mathematical operators, DIV and MOD, that yield respectively the integer part of the quotient and the remainder after division.

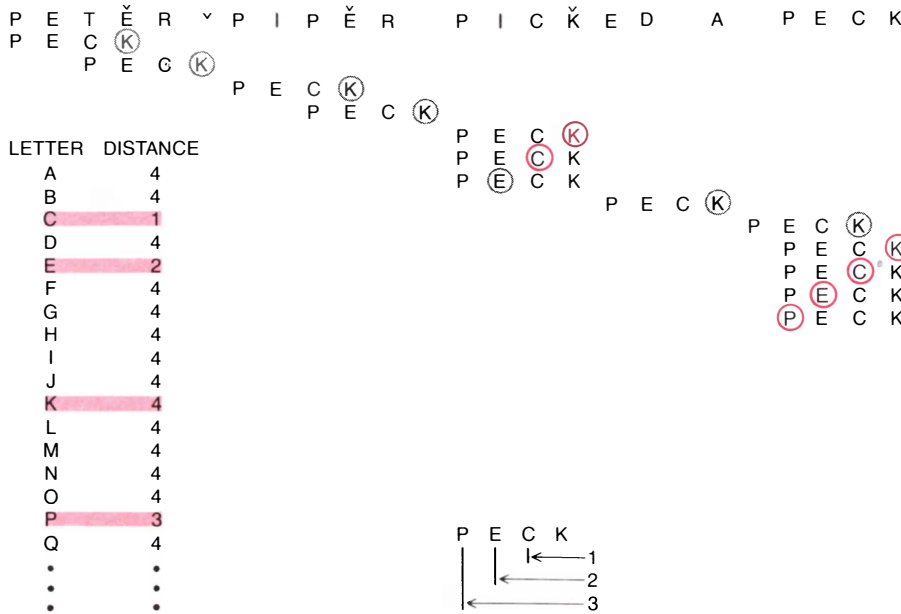
The modification of the algorithm to exploit this more efficient procedure can be guided directly by the need to preserve the invariance of the assertion $a * b + c = x * y$. The assignment statement $b := b - 1$ in the original algorithm is to be replaced by $b := b \text{ DIV } 10$; preserving the invariance requires that a be multiplied by 10, and so the statement $a := a * 10$ is added to the program. If b is not a multiple of 10, the remainder ($b \text{ MOD } 10$) is subtracted from b and, in order to preserve the invariant, $a * (b \text{ MOD } 10)$ is added to c .

The factor of 10 in the multiplication procedure is chosen because of its convenience in the decimal number system. Since the computer uses binary numbers internally, it is advantageous to employ a factor of 2 instead. A simple substitution of 2 for 10 could be made, but further refinements are possible. The resulting algorithm is used in all computers. The gain in efficiency is substantial: the number of iterations of the loop is reduced from y to the logarithm of y to the base 2. Without this improvement computers would spend most of their time multiplying.

A second example is drawn from the realm of nonnumerical algorithms. Given a text stored as a sequence of characters the task is to find within it the first occurrence of a word, which can be defined as any sequence of characters no longer than the text. Algorithms built on this model are important in many areas of computer science, perhaps most obviously in word-processing programs.

The first step in constructing the algorithm is to specify the precise result wanted. In the illustration on the opposite page that is done in a formal notation (the predicate calculus); here I shall give a verbal description. The two variables, the text and the word, can be declared as arrays of characters, so that any chosen character can be retrieved by giving a computed index value. Assume the text is an array of m characters and the word is an array of n characters, where n is less than or equal to m . (In most cases n is much smaller than m .)

What condition is guaranteed to be satisfied when a matching sequence of characters has been found? The answer can be stated in terms of the index variables i and j that specify positions in the word array and the text array respectively. A match exists if for every value of i from 0 to $n - 1$ (that is, for the en-



FASTER SEARCH ALGORITHM was devised in 1976 by Robert S. Boyer and J. Strother Moore, now at the University of Texas at Austin. The search begins at the start of the text, but in each stage the letters are compared starting at the end of the word. The program must maintain a table giving the distance from the end of the word to the last occurrence of each letter in the word; if the letter is not included in the word, the table entry is the full length of the word. When a letter in the word fails to match a letter in the text, the table entry for the text letter is retrieved; the word is then moved to the right that many character positions. In the first comparison a k in the word does not match an e in the text, and so the word is moved two spaces.

tire allowed range of the index) the designated character in the word array is the same as the character in the text array specified by an index value of $j + i$. The value of j for which this condition holds points to the first character in the matching sequence, and it can serve as the result returned by the search algorithm.

Finding a matching sequence of characters is not all that was asked, however; the statement of the problem calls for the first matching sequence. Another condition must therefore be put on the algorithm: for all values of the text-array index j less than the value at which the matching sequence begins, the word array and the text array must differ in at least one character. The result is valid only if both conditions are met.

One more possibility must be taken into account: the word may not be present in the text at all. Such an oversight is a common cause of program failure. It can be corrected by specifying that if the word is not found, the returned value of j should be larger than the largest possible value for the start of a matching sequence, namely $m - n$.

The three conditions defined here—that the characters in the word and in the text correspond for all values of i and $j + i$, that there are no regions of correspondence for smaller values of j and that j is less than $m - n$ —represent assertions helpful in verifying the correctness of an algorithm. It turns out they are also useful as a framework for constructing the algorithm itself. The obvious approach to searching the text is by

repetition. A *while* loop is set up with the first and third conditions as guards and with the second condition as a loop invariant. The initial value of j is set equal to 0, so that the search begins with the start of the text. On each pass through the loop the guard conditions are tested, and if the word matches the text or if j is greater than $m - n$, the program exits from the loop; otherwise j is incremented by 1 and the loop is repeated.

What remains to be specified is how the match itself is detected, that is, how the characters in the text are compared with those of the word over the range of index values from $i = 0$ to $i = n - 1$. The answer is a loop within the main loop; for each value assigned to j the inner loop passes through the entire range of values of i , comparing the n characters one at a time. At the first discrepancy the inner loop is aborted. The value of i on exit from the loop indicates whether or not a match was found: if i is less than n , the comparison ended prematurely because of a mismatch.

The text-search algorithm above is straightforward but relatively inefficient. In essence the word and the text are superposed, starting at the beginning of both, and compared character by character. If a mismatch is detected, the word is shifted one position to the right and the comparison is repeated. This process continues until a match is found or the word has been shifted all the way to the end of the text. When there is no matching word in the text, a minimum

of $m - n$ comparisons are needed, and the number is generally higher.

For a task as fundamental as searching a text it may seem unlikely that any significantly better methods would be discovered after some 30 years of work in computer science. Nevertheless, in 1976 Robert S. Boyer and J. Strother Moore II, now at the University of Texas at Austin, found a faster way. Their idea allows j to be incremented by more than 1 in the program's main loop. The comparison of the word with a segment of the text starts at the end of the word and proceeds toward the beginning. If a letter in the word fails to match the corresponding letter in the text, the word is shifted forward to bring into alignment the next letter that does match at this position, which I shall call the pivot position. If no letter in the word matches at the pivot position, the word is shifted forward so that its last letter is one space beyond the pivot.

An immediate question raised by this procedure is how the next matching letter pair is found; if it must be done by comparing characters one at a time, nothing has been gained. There is another way: the program can maintain a table listing the distance from the end of the word to the last occurrence of each letter in the word. Of course time must

be invested in compiling the table, but it needs to be done only once; if the text is long enough, it is worth the effort.

The Boyer and Moore algorithm may be faster, but can one have confidence in its correctness? In particular, how can one be certain in shifting the word several places to the right without making any comparisons that no matching alignments were passed over? An informal argument is that a match requires identity of all the letter pairs, and the alignments passed over necessarily differ in at least one position, namely the pivot position.

Correctness and efficiency are the main concerns of programmers. I have shown that analytical methods can and must be used to establish correctness because an exhaustive empirical test would take too long even for simple problems. For precisely the same reason efficiency and performance cannot be measured empirically. The tool for their analysis is the calculus of probabilities.

Suppose one is given an array of n numbers and asked to find the maximum value among them. The obvious method is to scan the array sequentially, comparing each element with the largest one found up to that point. One might declare a variable named *max* and make

its initial value that of the first element. In a *while* loop each subsequent element is compared with *max*, and if the element is larger, *max* is assigned the value of the element.

It is evident that the loop must be repeated n times, which sets a lower bound on the execution time of the procedure. How often is the assignment statement executed? If the first element happens to be the largest one, the assignment is done only once; on the other hand, if the sequence is an increasing one, *max* is assigned a new value n times. Hence 1 and n are the extreme values, but what is the average? The question cannot be answered by experiment. There are $n!$ possible arrangements of the numbers, which is too many to examine even for small values of n . (With an array of just 16 elements and a computer capable of checking a million arrangements per second, an exhaustive analysis would take more than half a year.)

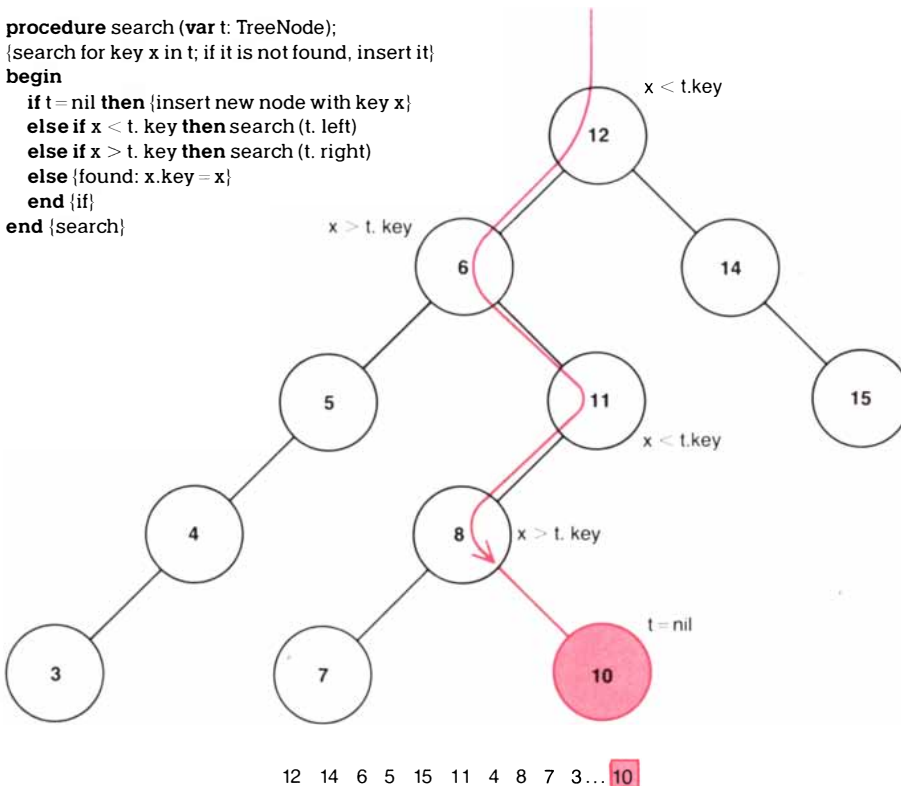
The analytical method of determining the average is quite simple. The initial assignment of a value is always executed once, and so the count of executions begins at 1. Assuming all permutations are equally likely, the probability that the second element is larger than the first is $1/2$; the probability that the third element is larger than either of the first two elements is $1/3$. The analysis can be continued in this way, so that the average number of assignments is equal to the sum $1 + 1/2 + 1/3 + \dots + 1/n$, which is known as the harmonic series. The 18th-century Swiss mathematician Leonhard Euler showed that the sum of the series is approximately equal to the natural logarithm of n plus a constant, now called Euler's constant, with a value of about .577. The logarithm of n grows much slower than n itself, and so the time spent assigning values to *max* is negligible compared with the time spent making comparisons and incrementing the array index. It is therefore reasonable to say that the effort needed to find the maximum among n numbers is proportional to n .

Most practical problems do not yield so readily, and the analysis of algorithms is an active field of research. In many cases it is enough to know how the execution time varies as a function of some measure of the size of the problem. For example, the time might increase in proportion to the size, or in proportion to the square of the size, or it might rise exponentially; exponential growth makes the algorithm all but useless. The study of such issues is called complexity analysis.

The methods of complexity analysis can be illustrated by an example: the construction of a binary tree, a data structure often adopted when the quick retrieval of information is important. The tree has two notable properties: each node can have at most two sub-

```

procedure search (var t: TreeNode);
{search for key x in t; if it is not found, insert it}
begin
  if t = nil then {insert new node with key x}
  else if x < t.key then search (t.left)
  else if x > t.key then search (t.right)
  else {found: x.key = x}
  end {if}
end {search}
  
```



RANDOM-INSERTION ALGORITHM is a single routine for maintaining a binary tree; it can both add information and retrieve it. The algorithm is a recursive procedure applied in exactly the same way at each node. Given a key value x , the algorithm compares it with the key of the node being examined; if x is less than the node's key, the left branch is searched, and if x is greater, the right branch is searched. If the value of x is equal to that of the key, the node being sought has been found. If the key is *nil*, the node does not exist and a new node is created at that position in the tree. In the example shown here a tree is searched for the key value $x = 10$.

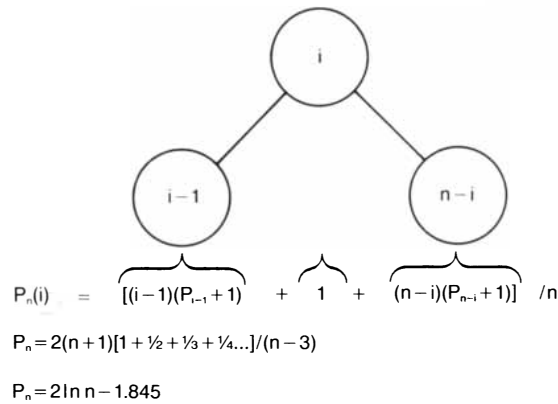
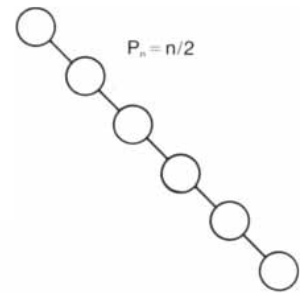
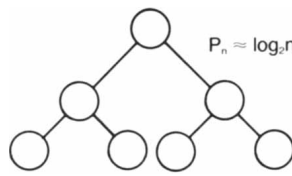
nodes, and the keys that identify the nodes are arranged so that at any node the smaller key is in the left subtree. Searching for a particular key can be very efficient; a comparison at each node indicates whether to take the left branch or the right, and so the number of remaining possibilities is halved at each node.

The efficiency is at a maximum when the tree is perfectly balanced, that is, when every node has exactly two subnodes. The average path length—the expected number of key comparisons—is then equal to the logarithm to the base 2 of the number of nodes. In the worst case, where the tree has degenerated to a simple list with just one subnode linked to each node, the average path length is one-half the number of nodes. From this result it appears one ought to take some care to keep the tree balanced as it grows. The balancing in itself requires a considerable effort, however, and so the question arises of whether the additional effort will pay off in faster searches. Surprisingly, in most cases it does not.

Suppose n key values are read in random sequence and inserted into a tree, which is initially empty. The left-to-right ordering of the keys is established as they are placed, but no effort is made to balance the tree. A single procedure can be designed both to add a new key and to search for one that is already present; the procedure merely finds the place in the tree where the key belongs and inserts it if it is not there. An algorithm for this purpose is shown in the illustration on the opposite page. It is a recursive procedure, one that invokes itself. At each node the algorithm takes one of four possible actions. If the value of the key at the node is *nil*, the new key is inserted there. If the value of the node's key is equal to that of the new key, a successful search is reported. If the new key is less than the one found, the procedure calls a new copy of itself to search the subnode to the left. If the new key is greater, the recursive search is directed to the right.

What is the average path length in a tree constructed by such random insertion? Again empirical measurements are out of the question because there are $n!$ possible insertion sequences, but a probabilistic estimate can be made. Assume that the keys are the integers 1 through n and that all permutations of them are equally likely. Some key, i , must be the first to arrive and so it becomes the root of the tree. When the rest of the keys have been inserted, there will be $i - 1$ nodes to the left of the root and $n - i$ nodes to the right. If i happens to be the midpoint of the range, the tree is perfectly balanced at this highest level; if i is equal to 1 or to n , the first branches of the tree are completely unbalanced.

The crucial idea in the analysis is that



BALANCING A BINARY TREE turns out to offer only a moderate improvement in efficiency in the average case. A fully balanced tree with n nodes (*top left*) has an average path length proportional to the logarithm of n . In the worst case, that of a tree degenerated to a list (*top right*), the average path length is $n/2$. For a tree constructed by random insertion, the average length must be determined by a probabilistic analysis. If the root of the tree is element i , the subtree to the left must include $i - 1$ nodes and the subtree to the right $n - i$ nodes. The total average path length is equal to the sum of the lengths for the two subtrees plus 1 for the root. The analysis can be repeated for each subtree, until ultimately the leaves of the tree are reached, where each subtree has a path length of either 0 or 1. Average path length in the random tree is a logarithmic function of n some 38 percent larger than the function in the balanced tree.

exactly the same reasoning can be applied recursively to each subtree. If the second key to arrive is j and it is less than i , then when the tree is filled, there will be $j - 1$ nodes in the branch to the left of j and $i - j$ nodes in the branch to the right. What is more, from this recursive description of the tree the average path length can be calculated by another recursive procedure. At the root the path length is equal to 1 (for the root node itself) plus the length of a subtree with $i - 1$ nodes plus the length of a subtree with $n - i$ nodes. These lengths are not known, but they can be calculated by applying the same procedure at the next level in the tree. Ultimately the end of each branch is reached, where every node has a path length of either 0 or 1.

The recursive definition of the path length must be averaged for all possible values of i from 1 through n . The result, shown in the illustration above, is an expression that again includes the harmonic series. The average path length of a tree constructed without concern for balancing is a logarithmic function of the number of nodes, and it differs from the optimum by a constant factor. The average case is much closer to the optimum than it is to the worst case: the path is longer by 38 percent.

The programs I have discussed here are not trivial, but they are very short. In practical applications programs tend to be long and intricate, and

they seem to grow as fast as the memory capacity of the computers available to run them. Can the methods of analysis I have outlined be applied to such programs? It is my conviction that they must, because complex systems require exact reasoning even more urgently than simple ones.

Most large software systems rely on few "deep" algorithms; rather they are built up out of basic algorithms such as multiplication and searching, which appear in many variations and combinations. The data structures, on the other hand, tend to be exceedingly complex. As a result the choice of the right data representation is often the key to successful programming, and it may have a greater influence on the program's performance than the details of the algorithm employed.

It is unlikely there will ever be a general theory for choosing data structures. The best that can be done is to understand the basic building blocks and the structures built up from them. The ability to apply this knowledge in constructing large systems is above all a matter of engineering skill and experience. In gaining such skill the programmer must constantly fight complexity, refuse to rely on a method not fully understood and never give up the search for simpler, more elegant solutions. In this effort no modern tool of software engineering can replace the programmer's faculty of precise, constructive reasoning.

Programming Languages

*They offer a great diversity of ways to specify a computation.
A language transforms the computer into a "virtual machine"
whose features and capabilities are determined by the software*

by Lawrence G. Tesler

A programming language is more than a notation for giving instructions to a computer. A language and the software that "understands" it can totally remake the computer, transforming it into a machine with an entirely different character. The hardware components of a typical computer are registers, memory cells, adders and so on, and when a programmer writes in the computer's native language those are the facilities he must keep in mind. A new language brings with it a new model of the machine. Although the hardware is unchanged, the programmer can think in terms of variables rather than memory cells, of files of data rather than input and output channels and of algebraic formulas rather than registers and adders. A few languages even give the computer a split personality: it becomes a collection of independent agencies that do their own calculations and send messages to one another.

Programming languages and their dialects number at least several hundred, and possibly a few thousand. The natural languages of human communication may be more numerous still, but in some respects programming languages are more diverse. Each language has its own distinctive grammar and syntax, its own manner of expressing ideas. In principle most computational tasks could be accomplished with any of the languages, but the programs would look very different; moreover, writing a program for a given task would be easier with some languages than with others. Here I shall describe some of the programming languages in common use and attempt to give an impression both of the elements they have in common and of the features that distinguish them.

The illustration on page 72 shows several stages in the development of a short program in Logo, a language devised in the late 1960's by Seymour Papert and his colleagues at the Massachusetts Institute of Technology. One interesting feature of Logo is the ability to control a "turtle," a small robotic device that can move forward and back-

ward, turn in place and raise or lower a pen that leaves a trace of the turtle's path. In many cases the turtle is not a real device but instead is simulated on a video display.

The initial version of the program consists entirely of commands to the turtle. First the pen is lowered, then the two commands *forward 50* and *right 144* are repeated five times and then the pen is raised. When the turtle follows the instructions, it draws a five-pointed star. The command *forward 50* causes it to draw a straight line 50 units long; *right 144* specifies a clockwise turn through 144 degrees, the change of heading at each vertex in a five-pointed star.

If writing a list of commands to be executed serially were the only method of conveying one's intentions to a computer, creating a complex program would be all but impossible. Actually Logo and other programming languages provide a number of facilities for simplifying and generalizing instructions. In this case the part of the program most conspicuously in need of improvement is the fivefold repetition of the *forward* and *right* statements. Whenever possible a programmer avoids writing anything more than once, and not just because the typing is onerous. If the program could be condensed, it would take up less space in memory. Furthermore, repetition increases the likelihood of a typographical error, particularly when the program is being revised. The repetition can be eliminated by replacing the five turtle-movement commands with the statement *repeat 5 [forward 50 right 144]*.

Suppose now the programmer wants to draw a nine-pointed star with edges 80 units long. That could be accomplished by means of a statement such as *repeat 9 [forward 80 right 160]* but it is apparent the same basic program structure is being duplicated, with differences only in detail. A better solution is to define a more general procedure in which the number of points and the length of a side are given as variable quantities. In Logo the word *to* introduces a procedure definition. Thus the

phrase *to star* indicates the instructions that follow should be stored as the method for drawing a star. Thereafter *star* becomes a new command in the language, one that can be entered into a program just as the built-in commands *forward* and *right* would be.

The variables in the *star* procedure are of the kind called parameters, which are "passed" to the procedure at the time it is invoked. In Logo the name of a parameter is preceded by a colon. Hence the procedure would be defined with a phrase such as *to star :size :points*; typing *star 80 9* would assign a value of 80 to the parameter *size* and a value of 9 to *points*, thereby generating a nine-pointed star 80 units on a side.

One further refinement might be added to the *star* procedure. In Logo a defined procedure can be called not only by the programmer but also by another procedure. This is an important source of power, but it increases the hazard of a procedure's being given inappropriate parameters. For example, in the intricacy of a program one might not realize that the turtle would be asked to draw a star with only one or two points. The problem can be addressed by adding the clause *if points > 2* to the program. The *if* clause serves as a "guard" that allows the turtle to draw only if the number of points specified is greater than two.

From the example above it can be seen that Logo has at least a superficial resemblance to a natural language such as English. It has a vocabulary of words, numbers and other symbols (collectively called tokens) that can be strung together to form larger constructs analogous to sentences. Some of the tokens are "key words" with a fixed meaning; others are defined by the programmer. Some of the tokens act as verbs; others function as nouns, modifiers or marks of punctuation. The rules governing how tokens can be combined constitute a grammar.

The sentences of a programming language are generally classified as declarations and statements. A declaration de-

defines what something is, what it means or how it is structured. In the Logo program, *to star :size :points* is a declaration that defines *star* as the name of a procedure and defines *size* and *points* as variables that serve as parameters to *star*. A statement, on the other hand, generally describes part of an algorithm; it specifies some action to be taken. In most cases a statement has the form of an imperative: it begins with a verb, which is followed by an object or a modifier. In the *repeat* statement *repeat* itself is a verb, the number following it serves as an adverb and the material in brackets is the object of the verb.

The vocabulary and the syntax of the *star* procedure are peculiar to Logo, but the mechanisms that control the flow of execution through the procedure can be found in the great majority of programming languages. In the absence of any explicit control statement, execution is

sequential. If one imagines the computer as reading the program, it reads the lines from top to bottom, so that unless the flow is altered each statement is executed exactly once.

One program element that alters the flow of execution is the *repeat* statement, which is an example of a looping, or iterative, construct. When the computer encounters *repeat n* followed by a group of statements in brackets, it reads and executes the material in brackets *n* times. Another way of controlling the computer's progress through a program is conditional execution, which in Logo (as in many other languages) is embodied in the *if* statement. The statements governed by an *if* are executed only if some specified condition is met.

There are many variations on the basic ideas of iterative and conditional execution. The *repeat* statement is useful only if it is known before the loop is

entered how many times it is to be executed. Other constructs allow the termination of the loop to be controlled by events within the loop itself; in essence a conditional expression is incorporated into the loop. In a number of languages a statement beginning with the key word *while* is repeated as long as a stated condition remains true. Another way of diverting program flow is an "unconditional jump," or *goto* statement, which simply shifts execution to a new point in the program. In recent years the *goto* statement has been in some disrepute among computer scientists on the grounds that programs with many such jumps are difficult to follow (for the human reader, not for the computer).

The notion of procedure definition itself is a vital element of programming. It is the chief mechanism of abstraction, the process of converting specific instances (a five-pointed star 50 units on a

The screenshot displays the PECAN program development system with several windows:

- Interpreter for SumOddNumbers:** Shows execution status:


```
>>> Begin execution ...
>>> User halted execution
>>> Step complete
>>> Step complete
>>> Step complete
```
- Stack Display:**

Program	DATA
myterms	<ARRAY>
Function sumodds	<UNDEFINED>
n	4
terms	<ARRAY>
i	4
SUM	39
- Symbol Table:**

```
SCOPE *INITIAL [module]
sumoddsnumbers: [program]

SCOPE sumoddsnumbers [block]
termindex: [type] TYPE(TYPE)
termarray: [type] TYPE(TYPE)
myterms: [variable] TYPE(termarray)
sumodds: [function] TYPE(PROC)

SCOPE sumodds [subprogram]
sumodds: [return] TYPE(integer)
n: [variable] TYPE(termindex)
terms: [variable] TYPE(termarray)

SCOPE sumodds [block]
i: [variable] TYPE(termindex)

SCOPE termst[i] [loop]
```
- Source Text Display:**

```
PROGRAM SumOddNumbers ;

TYPE
  TermIndex = 1 .. 100;
  TermArray = ARRAY [ TermIndex ] OF integer;

VAR
  MyTerms : TermArray;

FUNCTION SumOdds (n : TermIndex; terms : TermArray): integer;

VAR
  i : TermIndex;
  sum : integer;

BEGIN ( Function SumOdds )
  sum := 0;
  FOR i := 1 TO n DO
    IF odd(terms[i]) THEN
      sum := sum+terms[i];
  SumOdds := sum
END
```
- Expression Display:** Shows a binary tree for the assignment `sum := sum + terms[i]`:

```

  ASSIGN
  /  \
sum  ADD
      /  \
    sum INDEX
        /  \
      terms i
```
- Flowchart:** Illustrates the logic of the `FOR` loop and conditional execution:

```

graph TD
  Start(( )) --> I_FOR_MAX{i := FOR_MAX}
  I_FOR_MAX --> Odd{odd(terms[i])}
  Odd --> SumAdd[sum := sum + terms[i]]
  SumAdd --> SumUpdate[sumodds := sum]
  SumUpdate --> Return[RETURN sumodds]
  Odd --> Next{NEXT}
  Next --> End(( ))
```

SNAPSHOT OF A PROGRAM in execution is given by a program-development system called PECAN. Most of the source text, or original program, is displayed in the large window at the upper right; it is a program in the Pascal language for summing the odd terms in an array of integers. Commands controlling the execution of the program are given in the window at the upper left. Execution has been stopped at the statement where the actual calculation of the sum is done; the

statement is enclosed in a box in the source-text-display window. Below the source text, part of a flow chart for the program is visible, and to the left of that is a binary tree showing the logical structure of the assignment statement. The nested boxes at the lower left indicate the scope of symbols in the program. The stack-display window gives a view of the program's data structures. PECAN was developed by Steven P. Reiss of Brown University, who also created this illustration.

side) into general concepts (a star with any number of points and any size). A procedure is defined only once and is stored in memory only once, but it can be invoked from many places in a program, thereby allowing a product of mental labor to be used many times. Each time a procedure is called, execution transfers to the area in memory where it is stored; when the procedure has been completed, execution resumes with the instruction immediately following the call. A specialized kind of procedure, a function, returns a value of some kind to the calling program. The tangent function, for example, is given an angle as a parameter when it is called, and it returns a value equal to the tangent of the angle.

Among the hundreds or thousands of programming languages only a dozen or so are in widespread use. The illustrations on pages 74 through 76 show the same problem programmed

```
pendown
forward 50 right 144
forward 50 right 144
forward 50 right 144
forward 50 right 144
forward 50 right 144
penup
```

```
pendown
repeat 5 [forward 50 right 144]
penup
```

```
pendown
repeat 9 [forward 80 right 160]
penup
```

```
to star :size :points
  pendown
  repeat :points [forward :size right 720/:points]
penup
```

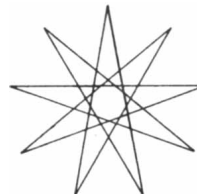
```
to star :size :points
  if :points > 2
    [pendown
     repeat :points [forward :size right 720/:points]
    penup]
```

DEVELOPMENT OF A PROGRAM IN LOGO is traced through five stages. The program gives instructions to a "turtle," a mechanical drawing device. In the first version of the program all the instructions for drawing a five-pointed star are given explicitly. The *repeat* statement in the second version condenses the program and reduces the likelihood of error. The third version has the same basic program structure but draws a larger star with nine points. In the fourth version a procedure is defined in which the length of a side and the number of points are variable quantities. In the final version of the program an *if* clause allows the procedure to execute only if the number of points specified is greater than two. The *repeat* and *if* statements are examples of control structures important in virtually all programming languages.

in six languages: BASIC, Pascal, COBOL, Fort, APL and Lisp. These six were chosen in part because they are well-established languages with a sizable population of programmers fluent in their use. They also illustrate well the great diversity of ways a single idea can be expressed. In each case an attempt has been made to write in a style that would be natural or comfortable to a programmer accustomed to the language.

The problem is not one of great intrinsic interest. It was chosen because a solution can readily be programmed in all the languages and because it demonstrates the essential mechanisms for defining variables and procedures and for controlling iterative and conditional execution. The problem is to sum the odd numbers in a set of integers.

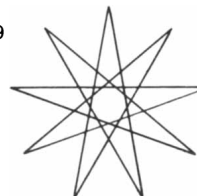
The BASIC programming language was developed in 1965 by John G. Kemeny and Thomas E. Kurtz of Dartmouth College, primarily as a language for introductory courses in computer science.



star 50 5



star 80 9



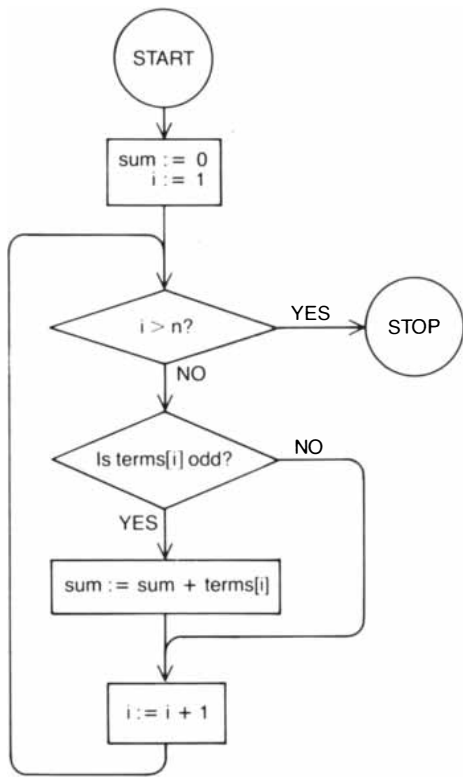
It has since fallen from favor somewhat in the academic world, but it has become popular in other contexts, notably the programming of microcomputers. In BASIC every line is identified by a number, and the control of flow through the program is based largely on references to the line numbers. The heart of the sample program is a loop in which all the statements between a *FOR* statement and a *NEXT* statement are executed repeatedly. The actual calculation is done in an assignment statement, which begins with the key word *LET* and gives a new value to a variable.

Pascal was designed in about 1970 by Niklaus Wirth of the Swiss Federal Institute of Technology in Zurich. It is another language meant for teaching that has been adapted to many other purposes. Pascal, unlike BASIC, requires the programmer to declare each variable and to specify its type; in this case the variables are integers and arrays of integers. Procedures and functions are referred to by name rather than by line number, which improves the readability of programs.

Pascal has been particularly important as a progenitor of later languages. For example, Wirth has recently designed a language called Modula-2 that builds on many of the concepts introduced in Pascal but emphasizes the construction of a program as a set of independent modules. Ada, a language developed under the sponsorship of the Department of Defense, is also based largely on Pascal, although it is considerably more complex.

COBOL was created in 1960 by a joint committee of computer manufacturers and users. The name is an acronym for Common Business-oriented Language, and COBOL has long been the principal language for large-scale data processing in government, banking, insurance and similar areas. A COBOL program is made up of four divisions, or parts: identification, environment, data and procedure. Only the data division, where variables are declared, and the procedure division, where algorithms are set forth, are shown in the bottom illustration on page 74. Whereas many programming languages are modeled on the notation of mathematics or formal logic, COBOL is modeled on the syntax of the English sentence; programs are highly readable but often verbose.

Forth was invented in about 1970 by Charles H. Moore, who was then at the National Radio Astronomy Observatory. The aim was a language for process control, in particular the control of telescopes, but again the language has been extended to other domains. It has been adopted for many minicomputer and microcomputer applications, in part because Forth programs tend to take up little space in memory. In contrast to COBOL, Forth programs are difficult

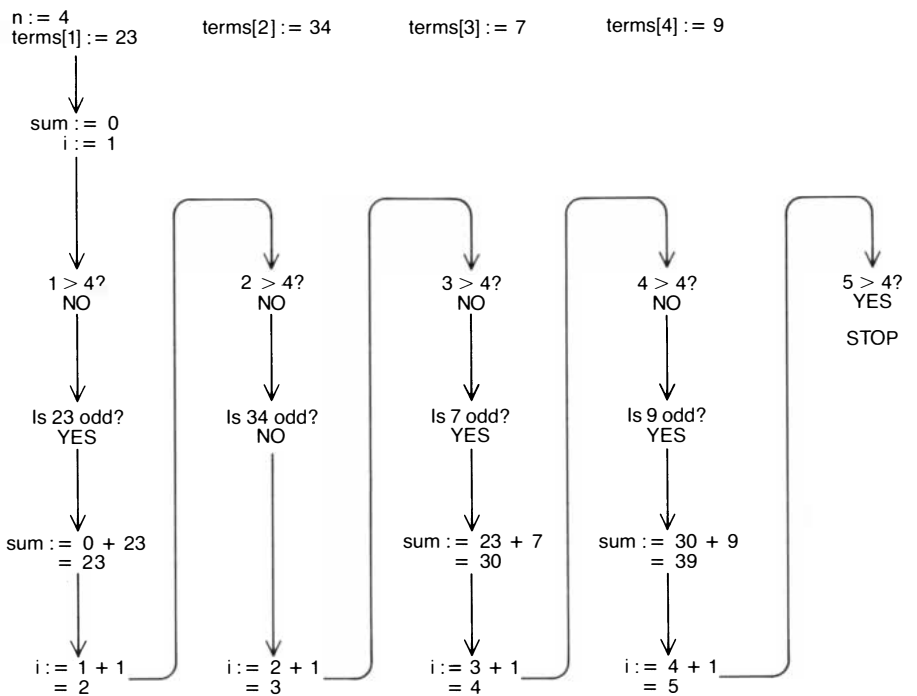


SAMPLE CALCULATION employed to illustrate the characteristics of several programming languages finds the sum of the odd elements in an array of n integers. The algorithm outlined in the flow chart at the left is embodied directly in the Pascal, BASIC and COBOL programs shown on the next page. The heart of the algorithm is a loop executed n times. On each passage through the loop a term of

to read and extremely terse; several key words are mere marks of punctuation.

In Forth the central facility of the computer is the "stack," an area of memory organized like a stack of cafeteria trays, so that the first item put on the stack is the last one removed. In the Forth version of the sample program it is assumed that the array of numbers and its size are at the top of the stack when the function is called; all calculations are done on the stack, and the value of the function is returned at the top of the stack. No variables are defined.

APL has a syntax even more concise than that of Forth. The initials stand for A Programming Language, but in 1961, when a book on APL was first published (by Kenneth E. Iverson of the International Business Machines Corporation), the language was merely a notation for expressing problems in applied mathematics; implementation on a computer came later. A distinguishing feature of APL is that it can deal as easily with an entire array of numbers as with a single value; a command that adds two numbers can be applied without change to arrays with thousands of elements. The sample APL program sums the odd elements of an array in a single statement. Taking the remainder of each number modulo 2 identifies the odd elements, which are then extracted from the array and added.



the array is examined; if it is an odd number, it is added to the running total. At the right the successive values assumed by the variables in the procedure are traced as the calculation is done for an array of four numbers. The symbol ":= " gives to the variable on the left the value computed on the right. A number in brackets, as in $terms[i]$, is equivalent to a subscript: it identifies an element of the array $terms$.

Lisp is in some respects the simplest of the languages considered here. It has only one kind of statement, namely a function call; its great source of power is that the value returned by a function can be another function. Lisp was developed in the late 1950's by John McCarthy, then at M.I.T., and since then it has been the preeminent language of those pursuing the goal of artificial intelligence. The name is derived from "list processing"; both programs and data are structured as lists.

The sample program could be written in Lisp as an iterative loop, but a Lisp programmer would be more likely to choose a recursive technique, in which a procedure calls on itself; the called procedure then issues another call to itself, and so on. Some means of escape must be provided or the recursion will become an infinite regress. The usual means is a conditional statement within the procedure: when the condition is satisfied, execution returns to the next-higher level.

In the sample program the set of numbers takes the form of a linked list. If the list is empty, the function returns a result of zero. Otherwise if the first element of the list is odd, it is added to the sum, and the function calls on itself with an argument consisting of the list that remains after the first element is removed. Eventually all the elements

are stripped off in this way, at which point the entire chain of pending calculations is completed.

In general computers are not built to "understand" Logo or BASIC or other languages that operate at a similar level of abstraction. The circuitry of the computer recognizes only the electronic embodiment of binary numbers. A program stored in a form that can be executed—the form called machine code—is a sequence of such numbers. Some of them represent instructions to the central processor, some of them are data and some are addresses in memory.

It is possible to write a program directly in machine code, but it is tedious, and the probability of completing even a small project without error is slight. (The computer has no trouble distinguishing 01101001 from 01011001 and remembering what each code means, but the human eye and mind are not used to best advantage in such tasks.) In the late 1940's and early 1950's, in an attempt to reduce the toil of writing machine code, programmers invented a notation called assembly code. Instead of writing down the binary digits for each machine instruction, the programmer wrote a short word or abbreviation, such as ADD, SUB or MOVE. Similarly, the address in memory where a variable is stored was replaced by a

```

program SumOddNumbers;
type TermIndex = 1..100;
   TermArray = array [TermIndex] of integer;
var myTerms: TermArray;
function SumOdds(n: TermIndex; terms: TermArray): integer;
begin
  var i: TermIndex;
      sum: integer;
begin
  sum := 0;
  for i := 1 to n do
    if Odd(terms[i]) then
      sum := sum + terms[i];
  SumOdds := sum;
end;
begin
  myTerms[1] := 23; myTerms[2] := 34; myTerms[3] := 7; myTerms[4] := 9;
  WriteLn(SumOdds(4, myTerms))
end.

```

PASCAL PROGRAM for summing the odd numbers in an array employs a function named *SumOdds* with two parameters: an integer *n* and an array *terms*. The function consists of the statements in the panel of color; the remainder of the program sets up a particular array on which *SumOdds* operates. In Pascal every variable must be introduced in a declaration that gives the variable's type. Some types, such as *integer*, are built into the programming language; others, such as *TermIndex*, are defined by the programmer. The loop is designated by the *for...to...do...* statement and the conditional is designated by the *if...then...* statement.

```

100 DIM T(100)
200 READ N
300 FOR I = 1 TO N
400   READ T(I)
500 NEXT I
600 GOSUB 1100
700 PRINT S
800 GOTO 2000
900 DATA 4
1000 DATA 23, 34, 7, 9

1100 REM MAKE S THE SUM OF THE ODD ELEMENTS IN ARRAY T(1..N)
1200 LET S = 0
1300 FOR I = 1 TO N
1400   IF NOT ODD(T(I)) THEN GOTO 1600
1500   LET S = S + T(I)
1600 NEXT I
1700 RETURN
2000 END

```

BASIC PROGRAM employs a subroutine to add up the odd terms in an array. The subroutine, indicated by the panel of color, has no name but must be referred to by line number; it is called by the *GOSUB 1100* statement. A BASIC subroutine also has no parameters; values are assigned to "global" variables, which the subroutine can then access. A variable does not have to be declared in BASIC unless it has subscripts, as in an array; in this example the *DIM* (for "dimension") declaration states that the array *T* can have as many as 100 elements. The *FOR...NEXT...* statement defines a loop and the *IF...THEN...* statement defines a conditional.

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01 NUMERIC-VARIABLES USAGE IS COMPUTATIONAL.
  02 TERMS PICTURE 9999 OCCURS 100 TIMES INDEXED BY I.
  02 N PICTURE 999.
  02 SUM PICTURE 9999999.
  02 HALF-TERM PICTURE 9999.
  02 RMDR PICTURE 9.

```

PROCEDURE DIVISION.

```

EXAMPLE.
  MOVE 23 TO TERMS(1).
  MOVE 34 TO TERMS(2).
  MOVE 7 TO TERMS(3).
  MOVE 9 TO TERMS(4).
  MOVE 4 TO N.
  PERFORM SUM-ODDS.

```

```

SUM-ODDS.
  MOVE 0 TO SUM.
  PERFORM CONSIDER-ONE-TERM VARYING I FROM 1 BY 1 UNTIL I = N.
CONSIDER-ONE-TERM.
  DIVIDE 2 INTO TERMS(I) GIVING HALF-TERM REMAINDER RMDR.
  IF RMDR IS EQUAL TO 1; ADD TERMS(I) TO SUM.

```

COBOL PROGRAM for the sum-of-the-odd-numbers calculation uses a procedure named *SUM-ODDS* that calls another procedure named *CONSIDER-ONE-TERM*. A COBOL procedure cannot have parameters, and so before *SUM-ODDS* is called by a *PERFORM* statement, values are assigned to *N* and to the first *N* elements of *TERMS*. The key words *PERFORM...VARYING...* define the loop and *IF...* introduces the conditional clause. In the data division the numbers 01 and 02 designate two levels in a hierarchy of data structures. *PICTURE* specifies how values are to be displayed. Only an excerpt from the complete program is shown.

name assigned to the variable. Numerical values were expressed in decimal notation. The words representing instructions were chosen to be more easily remembered than binary values, and so they came to be known as mnemonics.

At first the translation from assembly code to machine code was done by hand. It is a straightforward process: a table records the unchanging correspondence between instruction mnemonics and their binary codes, and a similar table can be constructed for the variable names appearing in a program. The process is clearly suitable for mechanization, and programs called assemblers were soon written to do the translation.

Some programming is still done in assembly code, because it offers direct access to all the facilities of the computer. Carefully written assembly code is fast and efficient, and if a compromise must be made between speed of execution and program size, the programmer is in direct control of such decisions. Modern assemblers are sophisticated translation programs. Nevertheless, there is still a rough one-to-one correspondence between lines of assembly code and machine instructions, so that programs tend to be quite long, and the possibilities for error are legion. The control structures available in most assembly codes are primitive. What is most important, assembly code remains closer to the computer's language than to the programmer's. Algorithms must be expressed in terms of what the machine is to do rather than in whatever terms might be natural to the problem at hand. (An assembly-code version of the odd-element-sum calculation is shown in the top illustration on page 77.)

In planning the solution to a problem one is unlikely to think in terms of registers and memory addresses; rather, the problem itself suggests the appropriate notation. If the problem is one in physics, the design of a program might begin with an equation such as $F = ma$; in business the formula chosen might be $profit = revenue - expenses$. The operations specified by the formula are then translated into explicit instructions to the machine. Early programmers recognized that this translation too might be mechanized. In that idea is the genesis of languages such as FORTRAN, BASIC and Pascal. For some years languages of this kind were called high-level languages; it now seems appropriate to refer to them simply as programming languages, because machine code and assembly code do not really qualify as languages.

Since 1960 most software has been written with the aid of programming languages. They have many advantages over lower-level representations. Because one statement can give rise to many machine instructions, programs tend to be shorter, which both reduces the labor invested in writing them and

improves their clarity. Working with concepts pertinent to the problem rather than with those defined by the machine also reduces the chance of error. Furthermore, it introduces the possibility of "machine independence," of writing a single program that can be run on many computers.

It is important to distinguish between a programming language and an implementation of the language. The language itself is the notation, the set of rules that define the syntax of a valid program. An implementation of a language is a program that converts the high-level notation into sequences of machine instructions.

There are two main kinds of language implementation: compilers and interpreters. A compiler translates the entire text of a high-level program in one continuous process, creating a complete machine-code program that can then be executed independently of the compiler. Working in a compiled language generally has three stages: the program text is created with a text editor or word-processing program, then the text is compiled and finally the compiled program is executed. The term compiler was coined in 1951 by Grace Murray Hopper, then at Remington-Rand Univac, to describe her first translator program. As part of the translation process the program retrieved standard sequences of machine instructions from tables stored on magnetic tape and compiled them into a complete program.

An interpreter executes a program one statement at a time, transforming each high-level construct into machine instructions on the fly. The difference between a compiler and an interpreter is analogous to the difference between a translator of literary works and a conversational interpreter. The translator takes a completed manuscript and delivers a new text in another language. The conversational interpreter renders each phrase or sentence as it is spoken. Actually most interpreter programs do some initial processing of the text before execution begins; key words are converted into shorter tokens and variable names are replaced by addresses. Still, the two kinds of implementation remain distinct: for an interpreted program to be executed, the interpreter must be present in main memory, whereas once a program has been compiled the compiler is no longer needed.

In principle any programming language could be either interpreted or compiled, but in most cases custom has made one or the other kind of implementation more common. FORTRAN, COBOL and Pascal are generally compiled; Logo, Forth and APL are almost always interpreted; BASIC and Lisp are widely available in both forms. The chief advantage of compilation is speed; be-

cause an interpreter must determine a suitable sequence of instructions each time a statement is executed, an interpreted language is almost inevitably slower. On the other hand, an interpreted language is often more convenient for the programmer; it is well suited to an interactive style of program develop-

ment. Sections of a program can be written, tested and executed without leaving the interpreter, and when an error is found, it can be fixed immediately, without the need to return to a text-editing program and then compile the program again.

The inner mechanism of a compiler or

```

: SUMODDS
0 SWAP 0
DO
  SWAP DUP 2 MOD
  IF +
  ELSE DROP
  THEN
LOOP

```

23 34 7 9 4 SUMODDS .

WORD	STACK	COMMENT
23	23	
34	23 34	
7	23 34 7	
9	23 34 7 9	
4	23 34 7 9 4	
SUMODDS	23 34 7 9 4	Call SUMODDS.
0	23 34 7 9 4 0	
SWAP	23 34 7 9 0 4	
0	23 34 7 9 0 4 0	
DO	23 34 7 9 0 4 0	Remove loop-control values.
SWAP	23 34 7 0 9 9	
DUP	23 34 7 0 9 9 9	
2	23 34 7 0 9 9 9 2	
MOD	23 34 7 0 9 9 1	
IF	23 34 7 0 9 9	TOS = 1; do IF to ELSE.
+	23 34 7 9 9	
ELSE	23 34 7 9 9	Skip past THEN.
DROP	23 34 7 9 9	Skipped.
THEN	23 34 7 9 9	Skipped.
LOOP	23 34 7 9 9	Return to DO.
DO	23 34 7 9 9	
SWAP	23 34 9 7 7	
DUP	23 34 9 7 7 7	
2	23 34 9 7 7 7 2	
MOD	23 34 9 7 7 1	
IF	23 34 9 7 7	TOS = 1; do IF to ELSE.
+	23 34 16 16	
ELSE	23 34 16 16	Skip past THEN.
DROP	23 34 16 16	Skipped.
THEN	23 34 16 16	Skipped.
LOOP	23 34 16 16	Return to DO.
DO	23 34 16 16	
SWAP	23 16 34 34	
DUP	23 16 34 34 34	
2	23 16 34 34 34 2	
MOD	23 16 34 34 0	
IF	23 16 34 34	TOS = 0; do ELSE to THEN.
+	23 16 34 34	Skipped.
ELSE	23 16 34 34	
DROP	23 16 34 34	
THEN	23 16 34 34	
LOOP	23 16 34 34	Return to DO.
DO	23 16 34 34	
SWAP	16 23 23	
DUP	16 23 23 23	
2	16 23 23 23 2	
MOD	16 23 23 1	
IF	16 23 23	TOS = 1; do IF to ELSE.
+	39 39 39	
ELSE	39 39 39	Skip past THEN.
DROP	39 39 39	Skipped.
THEN	39 39 39	Skipped.
LOOP	39 39 39	No more iterations.
	39	Return from SUMODDS.
	<empty stack>	Print the result.

FORTH PROGRAM for summing the odd numbers in an array declares no variables or other data structures but works exclusively with values on a "pushdown stack." When *SUMODDS* is called, the elements of the array are expected to be on the stack, with the number of elements at the top. The line below the procedure definition would be typed to execute the program with an array of four elements. A complete trace of the program's execution is then given, showing the content of the stack after each word is executed. A numeric "word" such as 0 or 2 pushes the number onto the stack; *SWAP* exchanges the top two elements; *DUP* pushes a copy of the top element onto the stack; *DROP* discards the top element. Operators such as "+" and *MOD* replace the top two elements on the stack with the result of the operation. The loop construct *DO* removes two elements (say *i* and *j*) from the stack and executes the words up to *LOOP* a total of *i - j* times. The conditional *IF* executes the words between *IF* and *ELSE* when the top of the stack (TOS) is nonzero and otherwise executes the words between *ELSE* and *THEN*.

an interpreter is a subject too large to be covered here, but the structure of a typical compiler can be described in outline. There are at least three phases in the compilation process. The first phase is a lexical analysis, in which the compiler identifies the various symbols in the text of the program and classifies them as key words, numerical values, variable names and so on. The next phase is parsing, in which the compiler determines the syntactic relations of the key words and builds a skeleton representation of the program's structure. Each *if*, for example, is associated with a subsequent *then*. In the third phase machine code corresponding to the parsed structure is generated. Some compilers add a fourth phase of optimization, in which the code is revised to improve its efficiency.

Over the past 30 years much careful thought has been given to the design of compilers, and there is now a well-developed methodology for their construction. The first step is to define the language itself in a completely explicit form. It is now common practice to

specify the grammar in terms of "production rules" that can be applied recursively to generate all the possible statements of the language. The creation of the compiler is then a comparatively straightforward job of programming; there are even compiler compilers that can automate part of the task.

The idea of a programming language has been around almost as long as there have been large-scale digital computers. In 1945 the German mathematician Konrad Zuse invented a notation he called Plankalkül. Statements in the language had a two-dimensional format: variables and their subscripts were aligned vertically and operations on them were laid out along the horizontal axis. Zuse wrote Plankalkül programs on paper—including one that made simulated chess moves—but he did not implement the language. Many of the ideas he developed, however, have been introduced into modern languages.

Surely the most influential of all programming languages was FORTRAN,

developed by John Backus and his colleagues at IBM between 1954 and 1957. The name stands for "formula translation," and the language was intended for scientific and numerical calculations, for which it is still in use. At the time the project was greeted with considerable skepticism. Computing machinery was then a scarce and valuable resource, with the result that much emphasis was put on program efficiency. It was assumed that a higher-level language would inevitably compromise efficiency, but Backus and his group performed an extraordinary feat: they created a compiler whose output was equal in quality to a hand-coded program.

At about the same time, Hopper and her co-workers at Remington-Rand Univac developed a programming language called Flow-Matic for business data processing. It was less sophisticated than FORTRAN, but experience gained with it over a period of several years was the main inspiration for COBOL. Another major language introduced in the late 1950's was Algol (which stands for "algorithmic language"). Algol-58, the first version, was designed by an international committee that drew on both the pragmatic syntax of FORTRAN and the more elegant notation of Plankalkül. The result was a language that is both readable and practical and that has had an important place in the genealogy of later languages, including Pascal.

Quite a number of other languages trace their roots to the same era. COMIT was created for text analysis and APT for the control of machine tools. JOVIAL, a derivative of Algol, was the first widely used multipurpose language; it was suitable for both scientific and business applications. In the early 1960's Lisp appeared and so did the notation (but not an implementation) of APL.

The rapid proliferation of languages troubled many observers. After all, most mathematics is done with a single, universally accepted notation. Implementing a new language is a major undertaking, and becoming comfortable as a programmer in it also takes time. Soon several projects were launched to design a new language so complete and versatile that it could serve as the universal argot of programming. All such endeavors have failed. The partial success of PL/I, developed under the sponsorship of IBM in 1965, made it clear that a language for all purposes is likely to be both hard to learn and hard to implement. Moreover, as the techniques of computing became more diverse, people realized that new languages would continue to be needed to address special application areas.

In a sense, all programming-language research since 1957 has been motivated by attempts to correct flaws in FORTRAN. Indeed, FORTRAN itself has been re-

```

▽SUM ← SUMODDS TERMS
[1] ▽SUM ← +(2 | TERMS)/TERMS

```

```
SUMODDS 23 34 7 9
```

TERMS ← 23	34	7	9	Initial value assignment.
(2 TERMS) ← 1	0	1	1	Array of remainders.
(2 TERMS) / TERMS ← 23		7	9	Compression of two arrays.
+/(2 TERMS) / TERMS ← 23	+	7	+	Reduction by addition.
SUM ← 39				Assignment of result.

APL PROGRAM calculates the sum of the odd elements in an array with a function whose operation is specified in a single line. The function has one parameter, *TERMS*, an array that "knows" how many elements it has, so that *N* need not appear in the program. An APL statement is executed from right to left except where parentheses alter the order of evaluation. In this example the expression $(2 | TERMS)$ is evaluated first; it calculates the remainder left after dividing each element of *TERMS* by 2 and creates an array of the same size as *TERMS* to hold the remainders. The symbol "/" can indicate two different operations, both of which appear in the example. In the expression $(2 | TERMS) / TERMS$, "/" is a "compression" operator that creates a new array in which each element of *TERMS* appears only if the corresponding element of $(2 | TERMS)$ is nonzero. In the symbol "+/," "/" is a "reduction" operator that reduces the array to a single number by inserting a "+" between each pair of elements.

```

(DEFUN SUMODDS
 (LAMBDA (TERMS)
 (COND
 ((NULL TERMS) 0)
 ((ODD (CAR TERMS)) (PLUS (CAR TERMS) (SUMODDS (CDR TERMS))))
 (T (SUMODDS (CDR TERMS)))))

```

```
(SUMODDS '(23 34 7 9))
```

```

(SUMODDS '(23 34 7 9))
= (PLUS 23 (SUMODDS '(34 7 9)))
= (PLUS 23 (SUMODDS '(7 9)))
= (PLUS 23 (PLUS 7 (SUMODDS '(9))))
= (PLUS 23 (PLUS 7 (PLUS 9 (SUMODDS ' ( )))))
= (PLUS 23 (PLUS 7 (PLUS 9 0)))
= (PLUS 23 (PLUS 7 9))
= (PLUS 23 16)
= 39

```

LISP PROGRAM calculates the odd-element sum by means of a function that calls on itself recursively. A Lisp function is a list, where the first element (called the *CAR*) is the name of the function and the remainder of the list (the *CDR*) gives the parameters. *DEFUN* is a function-defining function and *LAMBDA* precedes the names of the parameters; here the only parameter is the list of numbers *TERMS*. *COND* is a conditional function that evaluates the *CAR* of the lists that form its parameters. If the result is *T*, or true, the *CDR* of the list is evaluated; otherwise *COND* goes on to the next list. Here there are three possibilities. If *TERMS* is an empty list, *NULL* is true and *SUMODDS* returns a value of zero. If the *CAR* of *TERMS* is odd, the *CAR* is added to the running total and *SUMODDS* is called to evaluate the *CDR* of *TERMS*. If neither of these conditions is true, the *T* clause (which must be true) is reached; it simply calls *SUMODDS* with $(CDR (TERMS))$ as its parameter. Calculations are left pending during each call.

MACHINE CODE		ASSEMBLY CODE		
		LABELS	INSTRUCTIONS	COMMENTS
00100100	01011111	SUMODDS	MOVE.L (A7)+,A2	Pop return address from the stack into A2.
00100010	01011111		MOVE.L (A7)+,A1	Pop address of first term into A1.
00110010	00011111		MOVE.W (A7)+,D2	Pop n into D1.
01000010	01000010		CLR.W D2	Assign a value of 0 to the sum in D2.
01001110	11111010		JMP COUNT	Jump to the end of the loop to test if n=0.
00001000	00101001	LOOP	BTST 0,1(A1)	If the term addressed by A1 is even...
00000000	00000000		BEQ.S NEXT	...then go to NEXT
01100111	00000010		ADD.W (A1),D2	...otherwise add the term to the sum in D2.
11010100	01010001		ADDQ.W #2,A1	Set A1 to the address of the next term.
01010100	01001001	NEXT COUNT	DBF D1,LOOP	Decrement D1; unless it is -1, go to LOOP.
01010001	11001001		MOVE.W D2,-(A7)	Push the sum from D2 onto the stack.
00111110	10000010		JMP (A2)	Go to the return address.
01001110	11010010			

MACHINE CODE AND ASSEMBLY CODE specify the steps of the odd-element calculation in terms of the hardware resources of the computer. The code is necessarily specific to a particular machine, in this case the Motorola 68000 microprocessor. The algorithm employed is much like that of the Pascal procedure *SumOdds*, although it is more compact than the code that would be generated by

a Pascal compiler. Parameters are passed to the procedure on a stack and the result is returned on the stack; the address at which execution is to resume when the procedure is finished is also on the stack. The assembly-code version of the program, in which instructions take the form of "mnemonic" abbreviations, can be translated directly into the binary machine code executed by the microprocessor.

vised several times. The original version put certain arbitrary constraints on the programmer—for example, a variable name could be no longer than six characters—and offered only limited capabilities for defining data structures. Perhaps the most serious deficiencies were in the facilities for controlling program

flow. All branch points had to be defined by line numbers, and unless care was taken the function of a program could be made quite obscure by a tangle of *GOTO* statements. Later versions introduced control structures that encourage a more readable programming style.

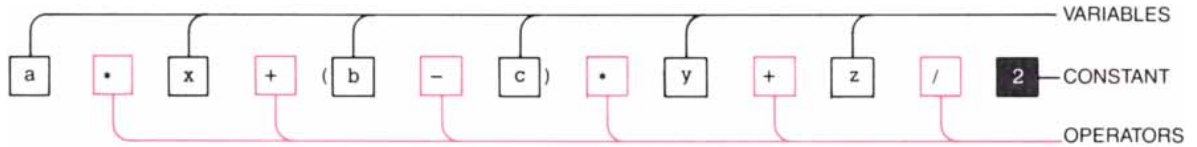
All the languages I have discussed so

far can be classified as procedural, or prescriptive. A program written in such a language tells how to get a result; it says first do this, then do that, and so on. There are also nonprocedural, or descriptive, languages, and they are becoming increasingly important. A descriptive program states what result is

ORIGINAL
EXPRESSION

$$a * x + (b - c) * y + z / 2$$

LEXICAL
ANALYSIS



PARSING

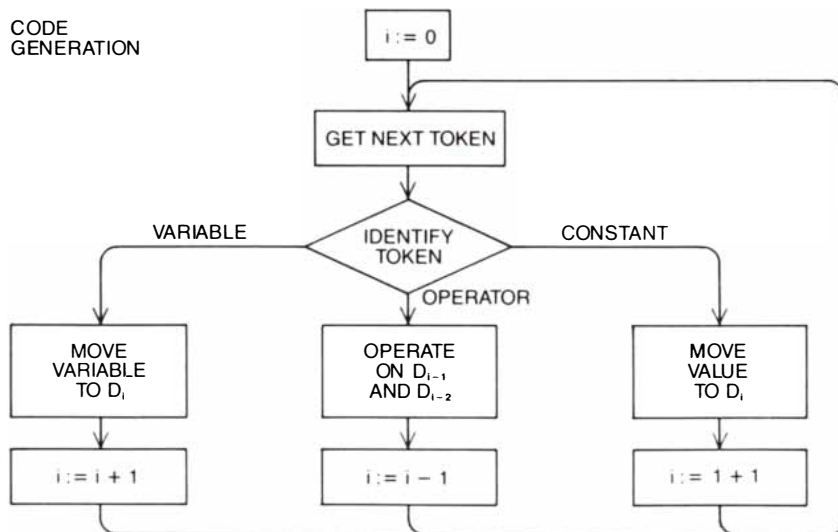
$$a * x + (b - c) * y + z / 2$$

$$(((a * x) + ((b - c) * y)) + z / 2)$$

$$(((a * x) * (b - c) * y) + (z / 2))$$

$$a * x * b - y * + z / 2 / +$$

CODE
GENERATION



i	INSTRUCTIONS	COMMENTS	i
0	MOVE.W A, D0	D0 := a	1
1	MOVE.W X, D1	D1 := x	2
2	MUL.W D1, D0	D0 := D0 * D1	1
1	MOVE.W B, D1	D1 := b	2
2	MOVE.W C, D2	D2 := c	3
3	SUB.W D2, D1	D1 := D1 - D2	2
2	MOVE.W Y, D2	D2 := y	3
3	MUL.W D2, D1	D1 := D1 * D2	2
2	ADD.W D1, D0	D0 := D0 + D1	1
1	MOVE.W Z, D1	D1 := z	2
2	MOVEQ.W #2, D2	D2 := 2	3
3	DIV.W D2, D1	D1 := D1 / D2	2
2	ADD.W D1, D0	D0 := D0 + D1	1

OPERATION OF A COMPILER, or translator, for a programming language has at least three stages. They are shown here for the particularly simple case of an arithmetic expression in Pascal. In lexical analysis the tokens, or symbols, that make up the program are identified and categorized. Parsing defines the semantic relations among the tokens. In an arithmetic expression the major task of the parser is to determine which operands are associated with each operator. It is done here by comparing the precedence of adjacent operators (as-

suming that multiplication precedes division, which precedes addition, and so on); parentheses are added around the operation of higher precedence. The expression is converted into "postfix" notation by exchanging the operator and the second operand in each sub-expression. The parentheses are then removed, yielding an expression that can be evaluated from left to right. Code generation transforms the parsed expression into machine instructions, employing a simple algorithm for assigning each variable to a hardware register.

wanted without specifying how to get it. The program sets forth relations rather than the flow of control, and so the programmer is relieved of responsibility for working out the steps of an algorithm and specifying their order.

The most conspicuous nonprocedural languages are the spreadsheet programs, such as VisiCalc and MultiPlan, that have become popular with the rise of personal computers. In MultiPlan a calculation is specified by writing formulas, much as in BASIC or FORTRAN. The order in which the formulas are to be evaluated, however, is determined by the implementation rather than by the programmer. To some extent temporal relations are replaced by spatial ones. In a conventional language the output of one procedure might serve as input to the next procedure; the analogous concept in a spreadsheet makes the value of one cell depend on the value of another.

There is even less sense of defining procedures in the language Prolog, a derivative of Lisp that has lately attracted the attention of many workers in artificial intelligence. In Prolog no formulas are written; instead relations between objects and quantities are defined. The language consists of declarations only and has no statements. Thus the relation (*product height width area*) describes the equality $area = height \times width$, but it does not specify that the height and the width are the given quantities or that the area is to be computed. The same relation can serve to find the height when the width and the area are known.

Another trend in the evolution of programming languages is the growth of interest in notational systems called object-oriented languages. As mentioned above, the statements of most programming languages are imperatives: the entity being addressed is not named, simply because there is only one possibility, an abstract embodiment of the computer as a whole. In an object-oriented language the computer is conceptually divided into objects that can

be addressed individually. Furthermore, the objects can communicate with one another by sending messages.

The notion of software objects was introduced by Ole-Johan Dahl and Kristen Nygaard of the Norwegian Computing Center in Oslo in Simula 67, a language derived from Algol 60. The idea did not attract widespread attention, however, until the development of the language Smalltalk in the 1970's by Alan Kay and a group of colleagues (of whom I was one) at the Xerox Palo Alto Research Center. Smalltalk consists exclusively of object-oriented constructs, which makes the language specification small and very general; on the other hand, because everything in the language is an object, some important data-structuring mechanisms cannot be implemented efficiently.

A software object consists of both data structures and algorithms. Each object "knows" how to carry out operations on its own data, but to the rest of the program the object can be treated as a black box whose internal workings are immaterial. Indeed, various objects may employ quite different algorithms to accomplish tasks the programmer identifies by the same key word. Just as penguins, horses and centipedes clearly have different methods for the activity identified generically as walking, so objects whose data consist of integers, arrays and complex numbers would employ different methods for the operation of addition.

My colleagues and I at Apple Computer, Inc., have developed a language called Clascal that adds the concept of classes of objects to the underlying structure of Pascal. Clascal, Smalltalk, Simula and some other object-oriented languages allow the objects in a class to inherit properties from a superclass to which they belong, so that each class does not have to be built up from scratch. Only those traits that distinguish the individual class need to be specified. Thus penguins, horses and centipedes share the concept of legs;

they differ in the number of legs and the details of the method of locomotion. Inheritance is another abstraction mechanism, allowing the properties of a class to be reused by many subclasses.

Inheritance turns out to be particularly useful in the design of interactive graphics software, another realm where there is much current activity. Entire programming languages can be built out of graphic images. Indeed, even certain computer games that rely heavily on graphics have some of the characteristics of a programming language. A notable example is a game called Robot Odyssey I, recently introduced by the Learning Company; "robots" programmed by connecting electronic logic gates and other components on a video screen can incorporate the concepts of conditional execution and procedure definition. A complete visual programming language tentatively named Mandala is now under development by Jaron Z. Lanier and his colleagues at VPL Research in Palo Alto. An example of what a Mandala program might look like is shown on the cover of this issue.

Another direction in which programming languages are expanding is the exploitation of parallel computation in computer systems made up of multiple processors. It would seem that 100 processing units ought to be able to solve a problem 100 times faster than a single processor of the same intrinsic speed, but the gain can be realized only if the software is able to break the problem into many pieces that can be worked on simultaneously.

Some languages provide an explicit mechanism for designating tasks that can be done in parallel; an example is the language called Occam, developed by the British semiconductor manufacturer Inmos. Other languages leave it to the compiler to analyze the program and discover opportunities for parallel execution. One such language is COMPEL (for "compute parallel"), on which I collaborated with Horace J. Enea in 1969. A COMPEL program consists entirely of assignment statements, which are not necessarily executed in the sequence in which they are written; the compiler is expected to deduce which calculations must be completed first. No compiler for COMPEL programs was ever written, but languages with a similar mechanism (called data-flow languages) have since been implemented.

The great diversity of programming languages makes it impossible to rank them on any single scale. There is no best programming language any more than there is a best natural language. "I speak Spanish to God, Italian to women, French to men and German to my horse," said Charles V (presumably in French). A programming language too must be chosen according to the purpose intended.

```
add (Adam is-parent-of Cain)
add (Adam is-parent-of Abel)
add (Eve is-parent-of Cain)
add (Eve is-parent-of Abel)
add (Cain is-parent-of Enoch)
which (x : x is-parent-of Abel)
Adam
Adam
Eve
No (more) answers
which (x : Eve is-parent-of x)
Cain
Abel
No (more) answers
```

```
add (x is-ancestor-of y if x is-parent-of y)
add (x is-ancestor-of y if z is-parent-of y and x is-ancestor-of z)
which (x : x is-ancestor-of Enoch)
Cain
Adam
Eve
No (more) answers
which (x : Adam is-ancestor-of x)
Cain
Abel
Enoch
No (more) answers
```

NONPROCEDURAL LANGUAGE called Prolog has no statements but consists entirely of declarations. In other words, a Prolog program gives no explicit instructions to carry out an operation; it merely states relations and makes inferences based on them. The illustration shows a program in a dialect called Micro-Prolog. The first five declarations set forth certain parent-child relations. The system can then answer questions about the established facts, for example identifying the parents of Abel and the children of Eve. Two rules of logical inference are then introduced to define the relation "ancestor of" in terms of the relation "parent of." The system can apply the rules to find all the ancestors or all the descendants of an individual.

Strong medicine for feverish health-care costs.

A new blood analysis system which embodies Kodak's expertise in chemistry, optics, physics, and electronics is helping clinical labs control costs. Its flexible software design is bringing new ease and reliability to the process of blood analysis. It can improve laboratory efficiency, increase productivity, and help to put health-care costs on the road to recovery.

The new system incorporates numerous technological inventions with such intricate design and engineering that it can perform a full range of routine tests—including kinetic enzyme, as well as potentiometric and colorimetric analysis—in operator-preferred sequence. If this sounds like quite an accomplishment, it is.

The Kodak Ektachem 700 analyzer incorporates 20 megabytes of hard-disk memory, relies upon Kodak's dry layer-coated Ektachem clinical chemistry slides like the one shown,* and

it produces hard-copy results for physicians' evaluation. In just 5 minutes!

But its big advantage is selective testing. At the touch of a CRT screen, this analyzer performs any combination of one to 26 assays and has the potential to report up to 7 additional calculated results on a single patient sample. That's a big plus, because it helps to eliminate wasted tests and wasted time, and contributes to operating economy.

Surprised that we're so committed to health care? You shouldn't be. We've been serving diagnostic medicine with radiography products for more than 80 years.

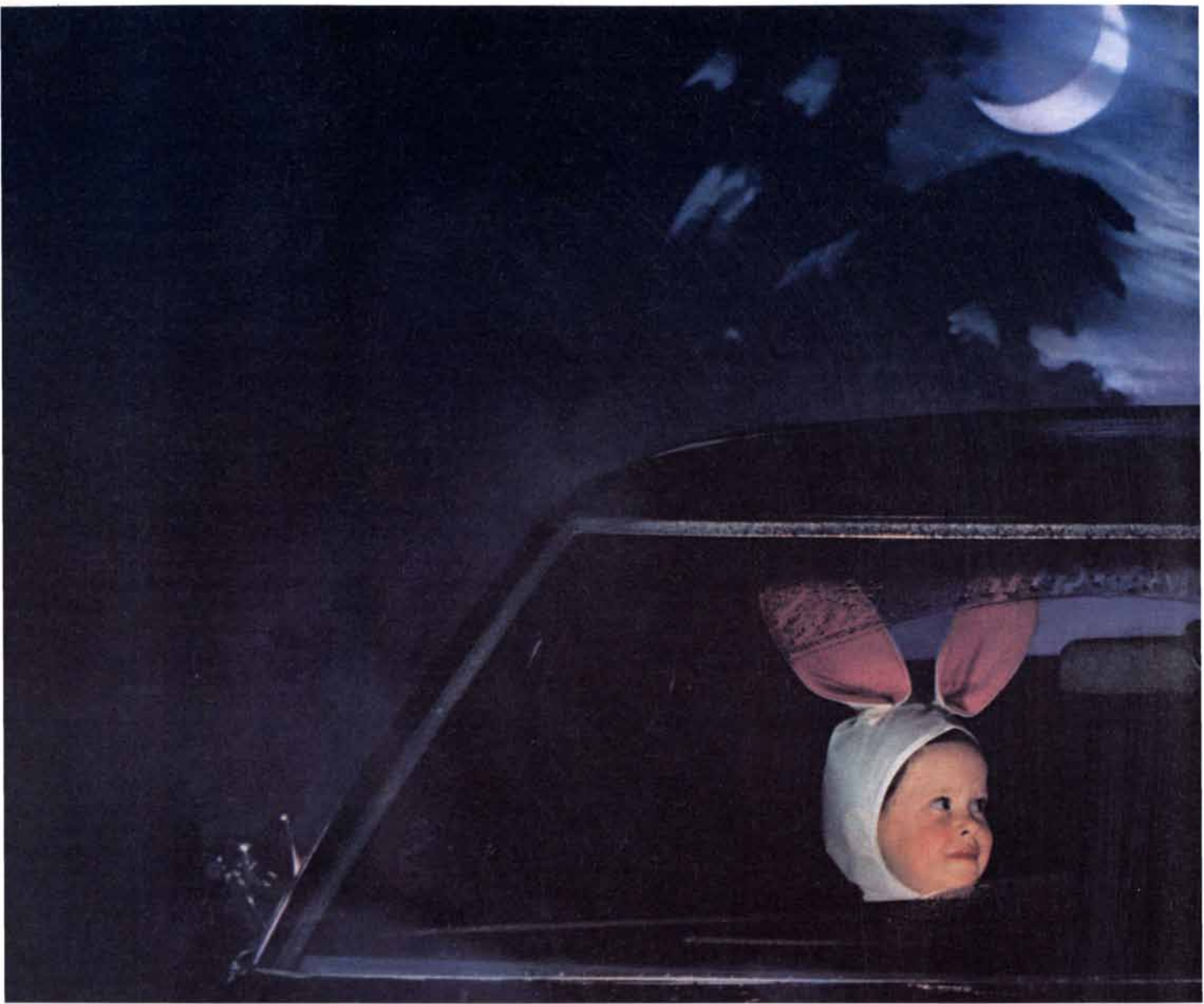
If you'd like technical papers dealing with the technology involved in Kodak Ektachem products, write: Eastman Kodak Company, Dept. GBSA-10, 343 State Street, Rochester, NY 14650.

Kodak. Where technology anticipates need.



*With Ektachem slides, complex sequential reactions can be carried out in ways not possible in solution chemistry. Multilayer coatings offer domains for multiple reactions within the single-use slide. In some layers, interfering substances can be trapped or altered; in other layers, reactions can be run which produce measurable signals.

A REASSURI



Every car maker has to meet the Federal requirements for safety and damageability. But Ford goes beyond those requirements in many important areas.

For example, all our cars have bumpers that are four times as strong as the government demands. (A fact GM, Chrysler, even Mercedes

and Volvo can't claim.)

We were the first major U.S. manufacturer to put halogen headlamps and steel-belted radials as standard equipment on all our cars. And we were also the first to offer an optional package of occupant protection features, including seatbelts and padded instrument panels.

You're going

*Based on a survey of owner reported problems during the first three months of ownership of 1983 vehicles.

**Offered by participating dealers.

ING QUALITY.

Every new Ford, Mercury and Lincoln comes
with 39 Lifeguard Design Safety Features.
To help protect your precious cargo.

Get it together — buckle up.

All of this, however, is just a small part of our commitment to build safe, reliable cars and trucks.

We also live up to that commitment by building those cars and trucks very, very carefully.

The fact is, a survey of thousands of new car and truck owners has shown that Ford is building the highest quality vehicles of any major U.S.

manufacturer.*

To learn more about all our 1984 models, and to get the details on the Lifetime Service Guarantee,** talk to your Ford or Lincoln-Mercury dealer.

We think that what you find out will be very reassuring.

to love the quality.



FORD • LINCOLN • MERCURY •
FORD TRUCKS • FORD TRACTORS

Quality is Job 1.

SCIENCE AND THE CITIZEN

And the Poor Get Sicker

Although a severe recession, such as the one that took place in the U.S. in 1981 and 1982, is generally thought of in economic terms, some of its direst consequences are not economic but medical. After each economic setback the health of individuals and of society deteriorates.

According to a study by M. Harvey Brenner of the Johns Hopkins University School of Hygiene and Public Health, in recent decades each increase of one-tenth in the fraction of the population that is unemployed (for example, a rise in unemployment from 10 to 11 percent) has been followed by an increase of 1.2 percent in overall mortality.

Brenner's work, which was done for the Joint Economic Committee of the U.S. Congress, draws on economic and health data from 1950 through 1980. Brenner and his colleagues examined the changes in the unemployment rate and other economic indicators, including the business-failure rate and per capita income, that occurred during the three decades. The economic fluctuations were compared with such changes in measures of health as overall mortality, mortality from cardiovascular and renal disease, the rate of first admissions to mental hospitals, the suicide rate, the rate of reported crime and the homicide rate.

The investigators found that the unemployment rate correlated strongly with each indicator of medical or social pathology. In addition to being associated with an increase in overall mortality, a rise of 10 percent in the unemployment rate corresponds to a 1.7 percent increase in the number of deaths from cardiovascular and renal disease (about 17,000 deaths based on the population of the U.S. in 1980), a .7 percent increase in the number of suicides (about 200 additional suicides), a 4.2 percent increase in the population of mental hospitals (about 6,000 new mental patients) and a 4 percent increase in the number of arrests (about 400,000 arrests).

The medical and social consequences of a recession ripple out for a considerable period after the initial economic shock. Most of the pathologies show three periods of peak incidence: one peak during the initial stage of economic contraction, just before the deepest point of the recession, the second from two to three years after the deepest point of the recession and the third from seven to 15 years after the deepest point.

In the case of the medical conditions the first peak could be due to the deaths of people with chronic syndromes made

acute by an economic setback. The later peaks could be the result of the development of chronic syndromes related to economic loss.

If the results of Brenner's study are applied to the sharp rise in unemployment that occurred in the 1981-82 recession, it appears that this event could be associated with as many as 75,000 additional deaths. Furthermore, the adverse health consequences are concentrated among those least able to cope with them: the members of low-income and minority groups.

Even during periods of prosperity members of low-income groups suffer a combination of stress, poor diet and inadequate medical care, which Brenner notes is associated with a heightened incidence of disease. Economic disturbances often intensify the disparity between the living conditions of the rich and those of the poor. Hence it is not surprising that increasing rates of some of the most severe pathologies correspond to an increase in social inequality.

For example, the economic measure that has the strongest association with the imprisonment rate is the ratio of unemployment among black males to unemployment among white males; the greater the disparity, the higher the overall imprisonment rate. The measure that has the strongest association with the homicide rate is the ratio of unemployment among males 15 through 24 years old to the overall rate.

The reduction of social services carried out by the Reagan Administration, the study suggests, may have exacerbated the impact of the recent recession on the health of the poor. It was found that decreases in AFDC payments were associated with an immediate increase in the infant mortality rate.

Rest Easy, Luddites

What will happen to the demand for human labor as computer-based automation pervades the service and manufacturing sectors of the U.S. economy?

A study, recently completed by the Institute for Economic Analysis (IEA) at New York University, suggests that latter-day Luddites can rest easy. Automation will not erode the total number of jobs. It will, however, swell the ranks of professional and technical workers while thinning the ranks of clerical workers. Manufacturing employment, the study predicts, will retain its proportionate share of the total labor force.

In order to make this forecast the IEA investigators, led by Wassily Leontief and Faye Duchin, employed an input-

output model of the U.S. economy: a detailed description of the flows of goods and services among industries. Within each of the 89 industrial sectors comprised by the model those quantities of labor, goods and services needed to sustain current production were distinguished from resources needed for capital expansion.

As data for the model, the investigators used input-output tables of the economy prepared for 1963, 1967, 1972 and 1977 by the Bureau of Economic Analysis in the Department of Commerce. They supplemented this information with data on prices, capital stocks and flows, as well as figures on employment from the Bureau of Labor Statistics.

The team subjected the model to three scenarios of technological change, assessing the impacts of each one on the demand for labor—divided for purposes of the study into 53 occupations—in each industry. The reference scenario, S1, assumed no further introduction of new technology after 1980; S2 specified a moderate pace of modernization, and S3 was an optimistic projection in which, for instance, electronic typewriters and work stations completely replace conventional typewriters by 1985. In each case modernization was assumed to take place within a context of steadily rising household and government demand for goods and services.

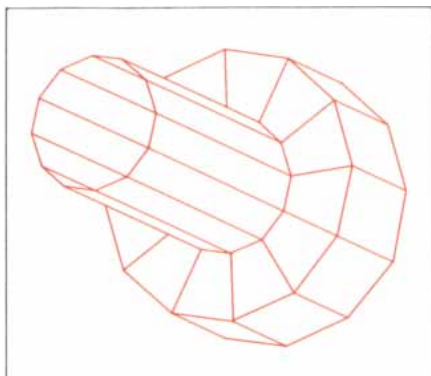
The results, say Leontief, "prove that you can use detailed information and get a detailed result." A rapid introduction of automation (S3) would enable the economy at the end of the century to produce the same quantity of goods using 10 percent less labor than is needed with today's technologies. The total number of jobs would remain high, however, because the rise in consumption assumed in the study would offset the laborsaving effects of automation.

The distribution of jobs, Leontief and Duchin report, will change dramatically. The share of professionals in total employment will rise from the 1978 figure of 15.6 percent to nearly 20 percent in the year 2000. Engineers and computer scientists will account for most of the increase. Clerical work, in contrast, will fall from the 1978 level of 17.8 percent of the labor force to 11.5 percent at the end of the century. Computerized office systems and telephone switchboards, automated teller machines and similar devices will account for the change.

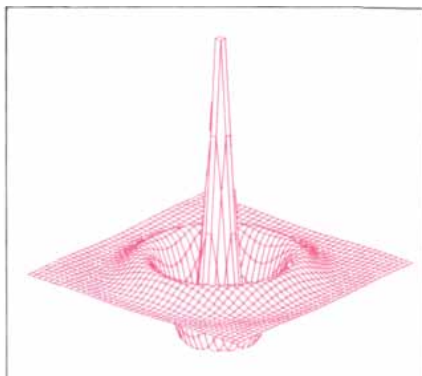
Middle managers also will be displaced as office computers take over their function of organizing and processing information. In factories, robots and computerized machine tools will

UPDATE THE SCIENTIFIC METHOD! ENERGRAPHICS ON THE IBM-PC*

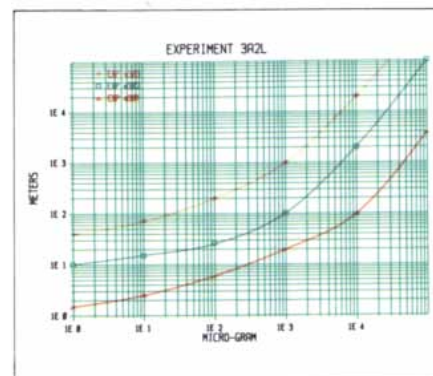
3-DIMENSIONAL OBJECT DRAWING ZOOM, ROTATE, HIDDEN LINE



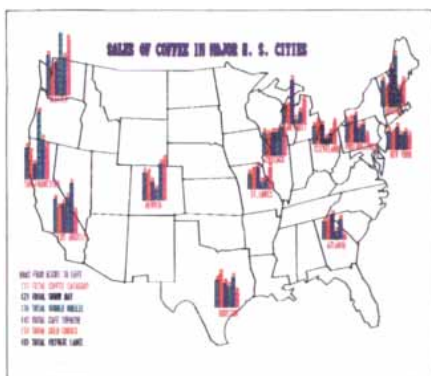
3-DIMENSIONAL SURFACE DRAWINGS



LOG-LOG SCALE GRAPHS



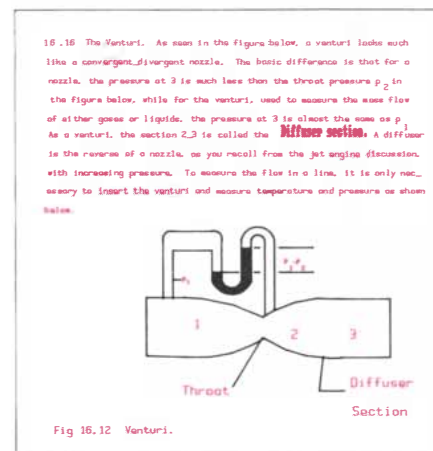
2-DIMENSIONAL DRAWING CAPABILITY



FLOWCHARTS ORGANIZATION CHARTS



DRAWING AND TEXT CAPABILITY



* **HARDWARE CONFIGURATION:**
IBM-PC; 128K; Dual Disk Drive; Graphics Adapter;
Dot Matrix Printer; and/or XY Plotter.

VisiCalc is a registered trademark of VisiCorp.
Lotus 123 is a trademark of Lotus Development Corp.
Multiplan is a trademark of Microsoft Corporation.

THE FIRST COMPLETE GRAPHICS PROGRAM DESIGNED FOR SCIENTISTS USING THE IBM-PC*

ENERGRAPHICS™

Dynamic and so easy-to-use, ENERGRAPHICS is already helping scientists and engineers in scientific research laboratories and universities around the country. ENERGRAPHICS can plot, analyze, research, graph, project and report with the best looking professional graphics available on the IBM-PC and other compatibles. ENERGRAPHICS interfaces with VisiCalc®, Multiplan™ and Lotus 123™ for maximum integration with spreadsheet programs.

Call 800-325-0174
(except in Missouri)
for our detailed
ENERGRAPHICS brochure

ENERTRONICS

Enertronics Research, Inc.
150 N. Meramec • Suite 207 • St. Louis, MO 63105 • (314) 725-5566

EVOLUTION

UNIX™ System V from AT&T is setting new standards—and solving old problems for the MIS manager. It's the software system that allows your company to take advantage of new technology. Without sacrificing your investment in computers and applications software.

It's another reason why good business decisions are based on UNIX System V.

How does today's MIS manager develop and implement a long-term information management plan—with minimal disruption and expense? By choosing software and hardware

products based on UNIX System V from AT&T.

Its unique capabilities help your data processing system evolve smoothly.

The profits of portability

UNIX System V software is portable from micros to mainframes. That's of critical financial and technological importance to you. Your software investment will soon be greater than your hardware investment.

Your software library can grow, too. Because software that is based on UNIX System V is hardware independent, you won't have to start your software library from scratch when you invest in new machines.

Portability also means you won't have to retrain your office staff every

time you buy a new computer. Your people can continue to use the same familiar software.

Gaining hardware independence

UNIX System V executes on a wide variety of hardware. That means greater leverage with your vendors. And greater potential for system growth. Even if the machines you're buying are of different generations.

You'll be able to add hardware more technologically advanced than your installed base—without disrupting that base.

UNIX System V is fully supported by AT&T and backed by a multimillion-dollar research and development program at AT&T Bell Laboratories. You can be assured that UNIX System V is

SYSTEM V

NONARY

a stable, fully documented, fully serviced platform that will continue to provide software portability.

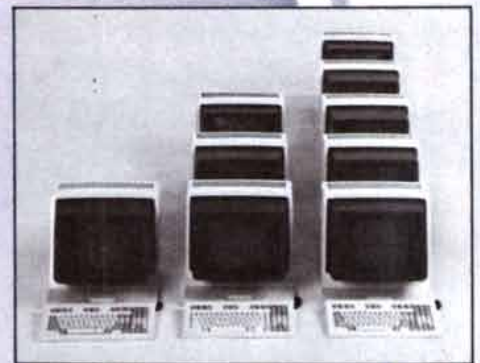
“Is it based on UNIX System V?”

A generation of computer science graduates is ready, willing, and able to work with UNIX Systems and the C language. Their familiarity will make turnover in your technical staff far less disruptive and less expensive.

It's another indication of the bottom line significance of UNIX System V—something that's becoming more and more obvious even to non-technical management. That's why so many are taking the time to ask, “Is it based on UNIX System V?” when it's time to invest in new software or hardware products for their company.

It's why you should be able to answer “yes” when they do. For more information, send for our free booklet, “Why Good Business Decisions are Based on UNIX System V.”

**UNIX System V. From AT&T.
From now on, consider it standard.**



AT&T

Please send me “Why Good Business Decisions are Based on UNIX System V.” SA0900BB

Mail to: AT&T, P.O. Box 967,
Madison Square Station, N.Y., N.Y. 10159

Name _____

Title _____

Department _____

Company _____

Address _____

City _____ State _____ Zip _____

Phone _____

UNIX System Licensee Yes No Don't know

eliminate some production-line jobs, and the machines' superior accuracy will reduce the need for checkers and inspectors.

Increased automation will, the model predicts, retard a major trend that has characterized the postwar decades: the shift of jobs from the manufacturing to the service sector of the economy. The rising demand for computers and computerized tools will create new manufacturing jobs even as automated equipment eliminates other jobs elsewhere in industry.

The report's optimism on the question of technological unemployment is guarded. The investigators measured the effects of incremental improvements in present technology; they did not take into account the introduction of such devices as voice-input typewriters. Moreover, the study was confined to the impact of computer-based automation; advances in other technologies such as agriculture or materials could affect the availability of jobs. Even if the number of jobs remains high, their changing distribution may mean that considerable training efforts will be necessary if the U.S. economy is to avoid a surfeit of labor in traditional jobs and a dearth of labor in newer ones.

Master Builder

How is it that a porpoise, a frog, a human being or a fruit fly achieves its characteristic appearance? What genetic mechanisms guide the development of a fertilized ovum so that it dif-

ferentiates into the internal organs and physical structures that give an individual member of a species form and function?

Some significant answers have begun to emerge from studies of the DNA of the fruit fly *Drosophila melanogaster*. Two clusters of genes figure importantly in the work. One is the bithorax complex, first identified by Edward B. Lewis of the California Institute of Technology. This complex controls the development of the thoracic and abdominal segments. Another cluster, the Antennapedia complex, identified by Thomas Kaufman and his colleagues at Indiana University, shares in the control of thorax development and promotes the development of the head.

Lewis, Kaufman and their colleagues discovered the role of each complex by inducing mutations with X rays and by observing natural mutations. Mutations in the two gene clusters produce a condition termed homeosis, in which one body part is substituted for another. For example, a mutation in the Antennapedia complex can give rise to an individual that sports legs in the place of antennae.

How is such an effect produced? Lewis hypothesized that the genes of the bithorax complex, working as a group to construct each segment of the developing insect, are expressed sequentially. The combination of genes expressed in the development of each segment specifies its form. A homeotic mutation alters the sequence that is expressed, causing the segment to assume the form

corresponding to the changed sequence; for example, legs appear instead of antennae.

By sequencing portions of the bithorax and Antennapedia complexes, William McGinnis, Walter J. Gehring and their colleagues at the University of Basel and Matthew P. Scott and Amy J. Weiner at Indiana found a common feature that may be a key to the way homeotic genes work. It is a section of DNA, 180 base pairs in length, that is present in at least six of the homeotic genes. Gehring has named the sequence, which differs by no more than 25 percent from gene to gene, the homeo box.

The homeo box is also found at another locus, the site of the *ftz* gene (so called because of the mutation that occurs there: *fushi tarazu*, Japanese for "not enough segments"). The *ftz* gene, part of the Antennapedia complex, seems to specify the number of segments in the developing embryo.

What role does the homeo box play? The Basel group and Allen Laughon of the University of Colorado at Boulder, working with Scott (who recently moved there from Indiana), have determined the amino acid sequence encoded by the homeo box. It closely resembles DNA-binding proteins, synthesized by species of yeast and bacteria, which can regulate the expression of genes. The finding raises the possibility that homeotic genes act as developmental master switches, using the homeo box to produce DNA-binding proteins that control other genes, which in turn specify the diversity of tissues within each segment.

This mode of developmental regulation may operate in other species. The Basel group has found that the homeo box is also present in the genomes of earthworms, beetles, frogs, chickens, mice and men.

Topping Out

The top, or *t*, quark, long sought by physicists in order to complete a list of particles generally thought to be the elementary constituents of matter, has probably been found.

At CERN, the European laboratory for particle physics, a computational search has culled nine unusual events from among some two million high-energy interactions recorded by the UA1 detector at the Super Proton-Antiproton Synchrotron between November, 1982, and July, 1983. Each event appears to signal the decay of a bound system formed by a top quark and the antiquark of another quark called the bottom, or *b*, quark. If the evidence is confirmed, the top quark is the sixth to be detected. The six quarks, together with the electron, the muon, the tau particle, three kinds of neutrino and the antiparticles of each particle, bring the number of known elementary particles to 24. Moreover, the



In a mutation of development-regulating genes, leglike structures replace antennae on the head of a fruit fly.



THANKS TO COMPU SERVE'S CB SIMULATOR, "DIGITAL FOX" ACCESSED "DATA HARI" AND PROCEEDED TO AN "ALTARED" STATE.

The CB Simulator, where CompuServe subscribers can access friends and influence people on 72 different channels.

Just pick your handle and get on line. From math to matrimony, there's always someone out there who speaks your language. Friends from all over the U.S. and Canada are at it 24 hours a day. Talking tech or just having fun. And if you've got a secret, just use the CB Scrambler.

That'll fool the "lurkers," those CB "see it all" who get their kicks by watching. Or you can always use the private talk mode for guaranteed one-to-one conversation.

The CB Simulator is just one of CompuServe's many electronic communications options that include a National Bulletin Board, Professional Forums and Electronic Mail. Plus, there's a world of on-line information and entertainment all for the price of a local phone call plus connect time.

You can access CompuServe with almost any computer and modem, terminal or communicating word processor.

To receive your illustrated guide to the CompuServe Information Service and learn how to subscribe, call or contact:

CompuServe

Consumer Information Service, P.O. Box 20212
5000 Arlington Centre Blvd., Columbus, OH 43220

800-848-8199

In Ohio call 614-457-0802

An H&R Block Company



Zeiss

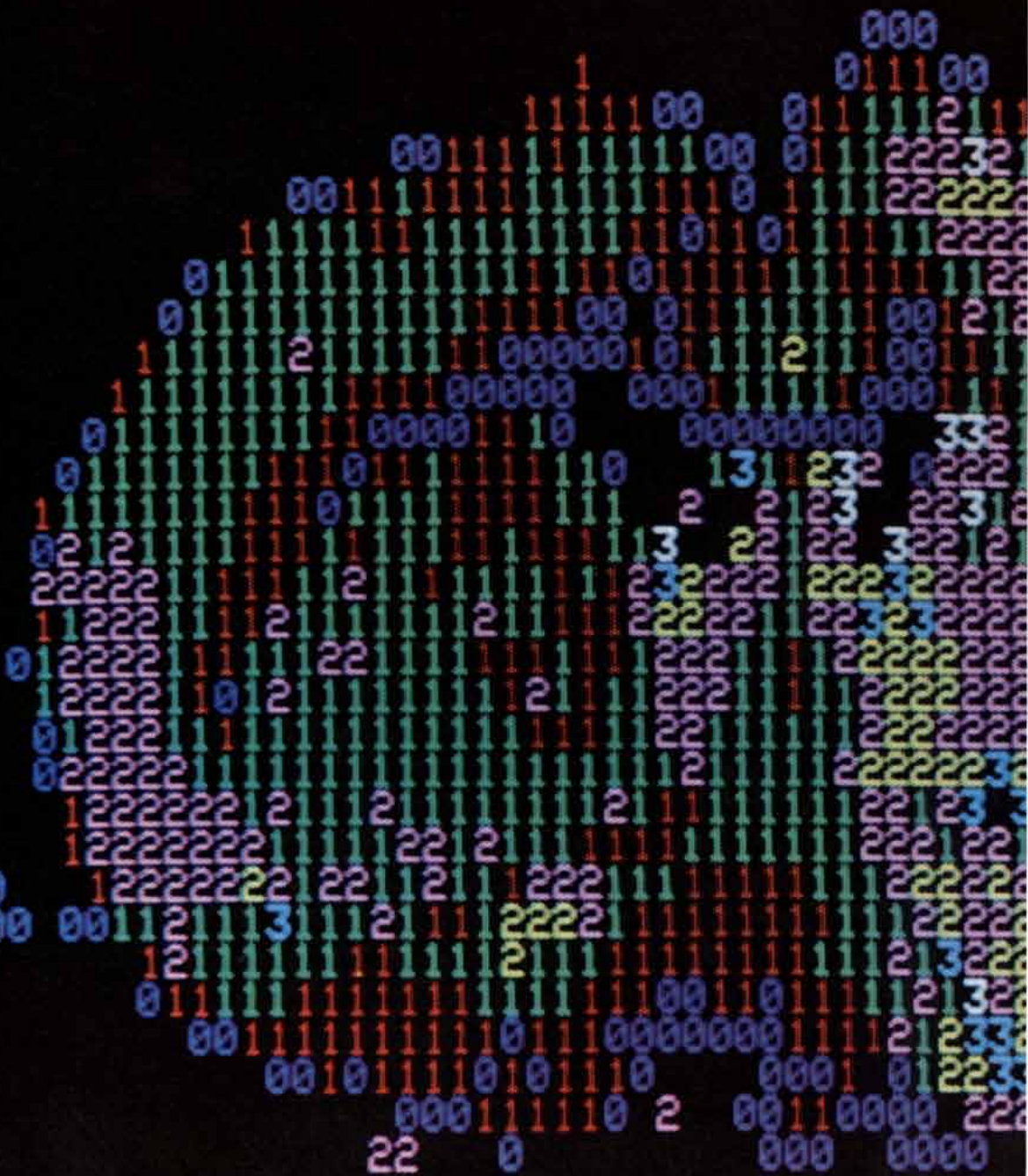
Carl Zeiss sold his first microscope in 1846. Twenty years and twenty employees later he had sold 1,000. Acknowledged as "... the most outstanding ones made in Germany ..." they were, nonetheless, produced by empirical, trial and error methods.

But it was also in 1866 that Zeiss entered into collaboration with Dr. Ernst Abbe. By 1872, Dr. Abbe had published the mathematical formulae that established for the first time – and for all time – the basis for microscope design.

In 1879, the two enlisted Otto Schott, an indefatigable experimenter in raw materials and melting processes for glass; and in 1884, the three founded a company for manufacturing optical glass to predictable standards – a science theretofore unknown.

Small wonder that Zeiss quickly became known as the "great name in optics," an identity it enjoys to this day.

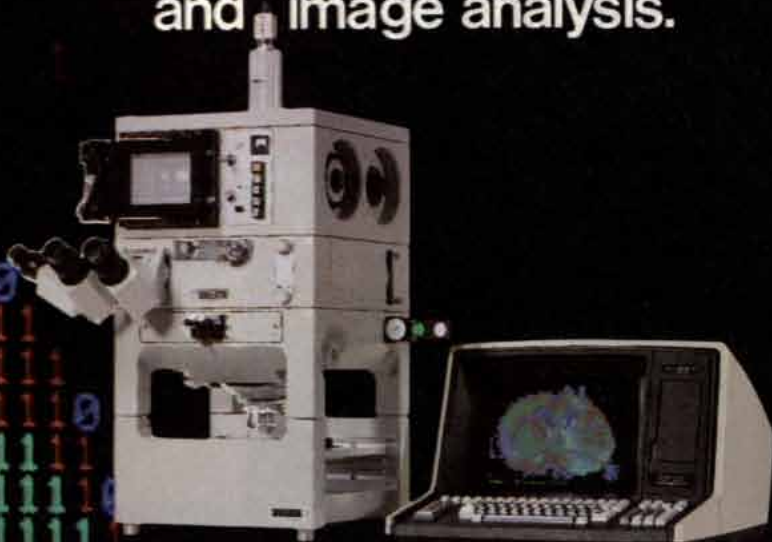
To superb optics and precision mechanics, the company has added excellence in advanced electronics to create a triple skill that, even in this age of high technology, is uniquely Zeiss.



DEPRESS 'C' KEY TO CONTINUE 'P' TO PRINT_

Today:

The great name in
quantitative microscopy
and image analysis.



It was with microscopes that Zeiss began, and it is upon microscopes that its most universal recognition is built.

Zeiss currently manufactures instruments for every technique in microscopy. They range in size and capability from the small, compact Zeiss Standard Microscope, through the Photomicroscope with its fully integrated 35mm camera, to the Universal, the most versatile of all conventional microscopes.

At the very pinnacle of the technology is the unconventional Zeiss Axiomat — the ultimate microscope, in which the optical axis is coincident with the axis of symmetry. The Axiomat achieves an image quality and stability whose only limitations are the

very laws of physics.

An innovator in electron microscopes since 1949, Zeiss is the only manufacturer in the world who excels in the twin fields of optical and electron microscopy. The Zeiss EM-10 and EM-109 series of electron microscopes are renowned for their advanced features and

ease of operation.

All Zeiss microscopes produce superb images. This image quality assumes added value when combined with computer technology for quantitative microscopy and image analysis. Zeiss systems extract and statistically analyze image data that might otherwise require many painstaking man-hours to uncover or, very possibly, remain forever hidden.

The remarkable capabilities of these quantitative microscopy and image analysis systems are fast making them indispensable tools in medicine, semiconductor manufacture, geology, cartography, geodetics, and many other disciplines.

Zeiss triune technology — optics, mechanics, and electronics — finds ready outlet in science, business, and industry. Currently representative are: computer numerical controlled measuring; computerized mapping and surveying; optics and electronics for microsurgery; electro-optical engineering and design; and products for sight.

And if that's Zeiss today...

Who knows what tomorrow will bring?
Carl Zeiss, Inc.
One Zeiss Drive,
Thornwood, NY 10594
(914) 747-1800

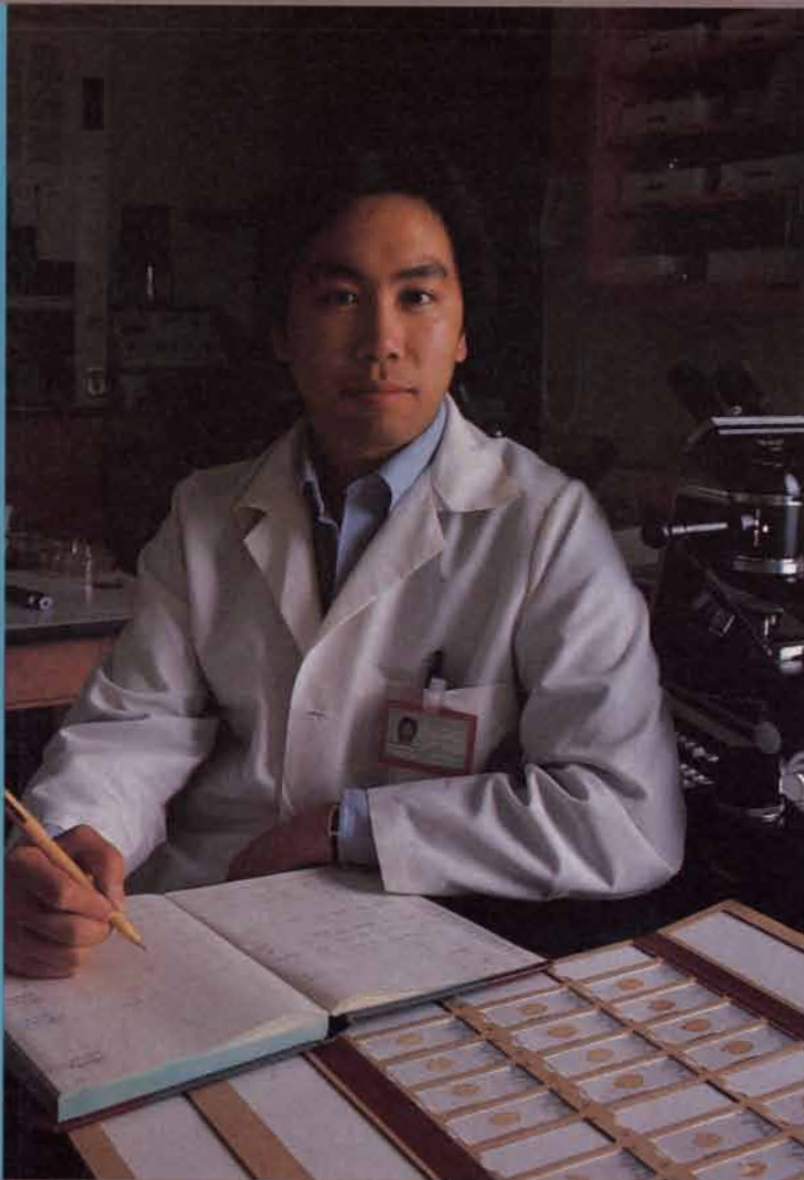
Who knows
what tomorrow
will bring?

ZEISS

Pascal-2™

PROVING GROUNDS

Pascal-2 Performs In Medical Neuroscience Lab at Scripps Clinic and Research Foundation. Division of Preclinical Neuroscience and Endocrinology.



“Pascal-2 is the language of our computerized imaging system. We hope to increase scientific objectivity, reduce labor costs during neuroanatomical analysis, and increase the quality of scientific research. Pascal-2 is the language of choice.”

Dr. Warren G. Young
Biosystems Analyst
Scripps Clinic and
Research Foundation
La Jolla, California

Dr. Young and his colleagues are investigating clinical disorders in the neural system. Pascal-2 is instrumental in maintaining a data base for the geometric attributes of neural cells.

Why is Pascal-2 Dr. Young's choice in software? "Structured programming, excellent records and file constructs, efficient coding, fast program execution, good documentation and customer support are some of the reasons why we chose Pascal-2."

To see how Pascal-2 meets your application needs call
1-800-874-8501.

Oregon Software

2340 S.W. Canyon Road
Portland, Oregon 97201
(503) 226-7760

The Pascal-2 system is available on Digital's PDP, VAX and Professional Computer systems running on the RSTS, RT-11, RSX and Unix Operating Systems.

Digital, PDP, VAX, RSTS, RT-11 and RSX are trademarks of Digital Equipment Corporation. Unix is a trademark of Bell Laboratories. Pascal-2 is a trademark of Oregon Software.

newly discovered top quark is one of the most massive particles known: the energy equivalent to its mass is between 30 and 50 GeV (billion electron volts).

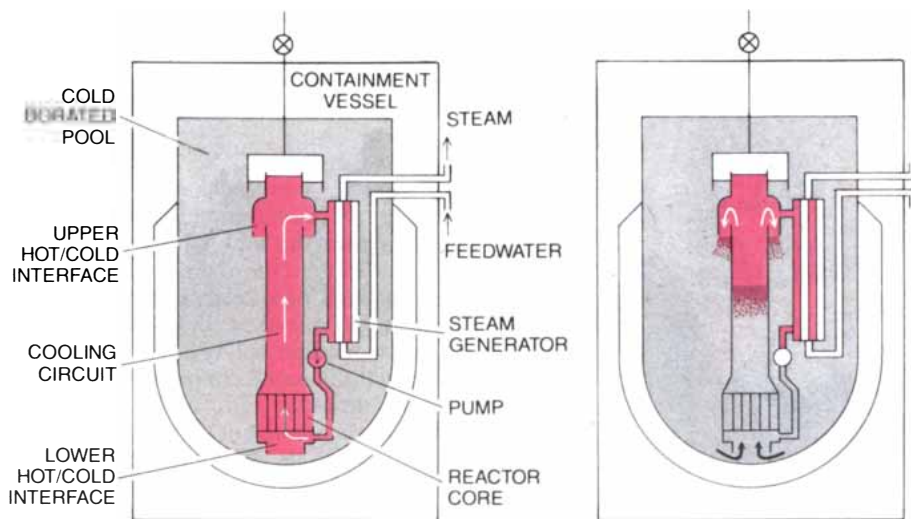
Quarks were proposed in 1964 by Murray Gell-Mann and, independently, by George Zweig, both of the California Institute of Technology. The proposal was put forward in part to account for the proliferation of hadrons seen among the by-products of collisions in high-energy accelerators. The proton, the neutron and more than 100 other particles are classified as hadrons. In the original version of the quark model only three quarks—the up, or *u*, the down, or *d*, and the strange, or *s*, quarks—and their anti-quarks were needed to explain the hadrons that were then catalogued. A major consequence of the quark model was therefore to restore parsimony to the fundamental understanding of matter. Since that time two new quarks, the charmed, or *c*, quark and the bottom quark, have been introduced in order to account for newly discovered hadrons. The discovery of a sixth quark suggests the quarks themselves are beginning to proliferate.

An Inherently Safe Reactor?

The safety of the light-water reactors that currently generate power in the U.S. is ensured by high-performance materials, redundant cooling systems, containment structures and carefully engineered controls. Such safeguards, numerous and costly, are needed to lessen the ever present danger of a core meltdown and the release of radioactivity into the environment. Is it possible to design an inherently safe reactor?

In a recent article in *Science*, Alvin M. Weinberg and Irving Spiewak of the Institute for Energy Analysis at Oak Ridge describe two reactor designs whose safety lies not in emergency backup systems but in the “immutable and well-understood laws of physics and chemistry” according to which the reactors operate. In such “inherently safe” reactors, say the authors, a core meltdown is essentially a physical impossibility.

The authors suggest that fundamentally safer reactor designs may be necessary if nuclear power is to emerge into a new period of growth. Since the Three Mile Island incident in 1979, numerous safeguards have been incorporated into U.S. reactors. Today, the authors report, the probability of a core meltdown stands at less than 10^{-4} per reactor-year. Yet in a future world heavily dependent on nuclear power, in which 5,000 or more reactors might be operating—more than 10 times today’s number—the probable frequency of a core meltdown somewhere in the world would be one every two years. Although only a fraction of such incidents would release substantial amounts of radiation, Weinberg



In the PIUS reactor coolant circulation normally excludes pool water from the core; during cooling-system failure, convection floods the core with pool water.

and Spiewak judge the rate to be unacceptably high.

One reactor design inherently resistant to such an accident is known as the modular High-Temperature Gas (HTG) reactor. Proposed, in slightly different forms, by American and German companies, it consists of a small graphite core cooled by helium gas circulating within the pressure vessel; a 1,000-megawatt generating station might consist of 10 such modules.

The HTG reactor’s safety is a consequence of both design and size. Because the gas does not circulate through external piping, the chance of a coolant loss is low. In the event of a cooling failure the core temperature would rise and fission would cease, as it would in a conventional reactor.

In a conventional reactor crippled by a loss of coolant, the heat released by the radioactive decay of fission products can melt the core even after the chain reaction comes to a halt. The very high surface-to-volume ratio of the small HTG core, in contrast, would allow the afterheat to radiate to the environment, stabilizing the core temperature at a safe level. Gas-cooled reactors incorporating some of the features the authors recommend have been in operation since 1956.

The other design to which the authors refer is untested. A Swedish concept, it is known as the Process Inherent Ultimately Safe (PIUS) reactor. Its novelty lies in the pressurized pool of boric acid solution that would fill the reactor vessel, immersing the core and its pressurized-water cooling system. The pool and the cooling system would be interconnected at several sets of baffles. Only the dynamic pressure developed by the coolant pumps would prevent the borated water from entering the cooling circuit; if the pumps or the cooling circuit were to fail, water from the pool would

flood the core. The boron in the solution, an efficient absorber of neutrons, would halt the chain reaction; the influx of water would carry off residual heat.

Ambiguous Protection

A new way to protect computer software from piracy has been suggested by Adi Shamir of the Weizmann Institute of Science in Israel and two of his students. Shamir’s software-protection method would enable a program to determine—as part of its normal function—whether it is on an illegally copied disk. An illegally copied program then either would cease to function or would self-destruct.

Many software manufacturers now protect their programs by selling them on unconventionally formatted diskettes; the format of a diskette defines the pattern in which the bits (binary units of information) representing various files are laid out on the disk surface. Arranging files in unconventional ways normally baffles a standard copying program, because the program must find an entire file before transferring the information onto a new diskette. Pirates have been able to circumvent this strategy by using programs that read each bit on the source disk and place the same bit at the exactly analogous location on the copy disk.

In an unpublished note Shamir suggests marking each legitimately sold diskette with ambiguous bits. An ambiguous bit can be formed by creating a region on a disk where the magnetic field strength is between the two strengths normally read as 0 and 1, the two binary bits. Owing to electrical noise and mechanical vibration in the reading device, such a region would appear to the user’s computer, entirely unpredictably and at random, sometimes as a 0 and sometimes as a 1. On being

INTESoft: POWERFUL, EASY, AND GREAT VALUE



Looking for powerful **integrated software** for your IBM PC—at a reasonable price? Have you noticed that powerful products are often complex and difficult? Not InteSoft. InteSoft products are **powerful and easy to use**, with a unique command structure that allows you to become productive immediately, plus **integration capability with other popular software**—all for a lower price than the competition. But don't take *our* word for it:

"The InteSoft programs are high-quality professional products and provide top-quality performance." *InfoWorld*, July 16, 1984.

"Although I'm a novice IBM PC user, InteWord is so well documented I generated my first page in less than 2 hours, *from scratch*." Dane Tong, Director of Purchasing, National Semiconductor/Datachecker DTS Division.

Software Digest* rated IntePlan number 1 in the Ease of Learning category and InteCalc one of the top 3 spreadsheets in the Overall Evaluation category. Call toll-free for a copy of InteMate, InteCalc, InteWord, IntePlan, or IntePert on a **30-day no-risk trial basis**. See the difference for yourself!

InteSoft

from Schuchardt Software Systems

800-421-1144
outside California

800-421-1145
inside California

Available at Software Centres International stores.
* *Software Digest*, July 1984 IBM PC Time and Project Management Programs Ratings Newsletter and April 1984 IBM PC Spreadsheet Programs Ratings Newsletter. IBM is a registered trademark of International Business Machines.

told to copy such a section of disk, the user's hardware will scan that section once, decide whether to call it a 0 or a 1 and make the corresponding copy, thereby losing the ambiguity.

Somewhere in the program itself there will be an instruction that tells the computer to read the section of the disk containing the ambiguous bits several times. If the computer reads the same value each time instead of the random values produced by ambiguous bits, the program either will refuse to run or will instruct the user's computer to erase the entire diskette.

Shamir points out that any protection method can be defeated. He says, however, that "the real problem is not to create a foolproof system but to make sure that for most users it makes more economic sense to rent or buy the software than to try to steal it."

The Stuff of Memories

How are memories stored in the brain? The mechanism remains elusive. Still, the storage must be accompanied by a lasting change in the brain: a change that affects the way the brain processes information. Such processing depends on the cell-to-cell transmission of signals across the intercellular contacts called synapses. Moreover, the ability to commit even fleeting events to memory suggests that the change can be induced by brief events in the brain.

Gary Lynch and Michel Baudry of the University of California at Irvine have a candidate for what they describe as "the biochemical processes involved in memory storage." They observed the results of the processes in the hippocampus.

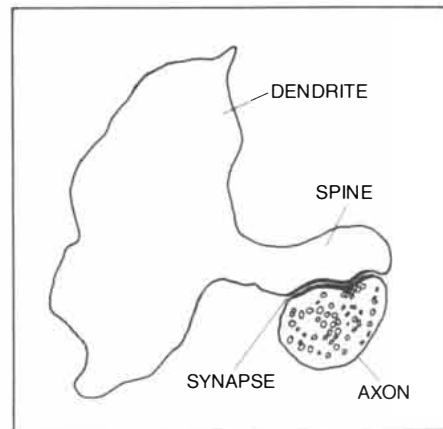
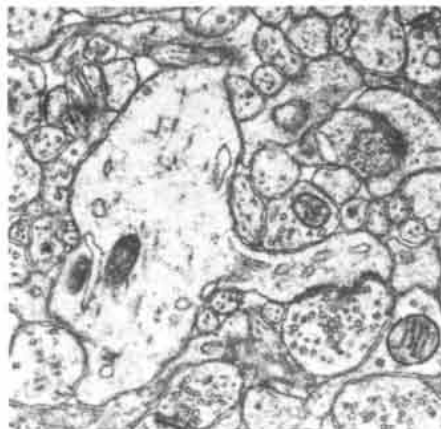
It was known that the excitation of certain circuits of nerve cells in the hippocampus can render the cells more sensitive to further signaling over periods of months; the phenomenon is called long-term potentiation, or LTP. Here, then, is a lasting functional change produced by a brief event. The question is

what biochemical and structural changes underlie the altered function.

One clue was that the hippocampal circuits exhibiting long-term potentiation employ an amino acid as their neurotransmitter: their synaptic messenger substance. The acid is thought to be glutamate. Accordingly Lynch and Baudry prepared slices of hippocampal tissue from rats, applied electrical stimulation to bring on long-term potentiation and then (five minutes to an hour later) assayed fractions of synaptic membrane. They found that the number of glutamate receptor sites had increased. Meanwhile electron microscopy of hippocampal tissue in which long-term potentiation had been induced revealed some structural changes. Dendritic spines, the thorn-shaped protrusions at which the extensions of a nerve cell receive synaptic signals from other cells, looked rounder. In addition the number of synapses on the main shaft of the nerve cell's dendritic extensions had increased by as much as 30 percent.

A further clue was that long-term potentiation seems to depend on the availability of calcium. The increase in the number of glutamate receptors proved to depend on calcium too. Hence Lynch and Baudry searched for—and found—an enzyme, bound to nerve-cell membrane and activated by calcium, that irreversibly "uncovers glutamate receptors." They found, moreover, that the enzyme (one of the class of enzymes called calpains) acts on the protein called fodrin, which lines the inner face of nerve-cell membrane. The disruption of the fodrin could conceivably account for the uncovering of receptors and also the change in the shape of dendritic spines.

Lynch and Baudry propose the following sequence. Bursts of neural signaling increase the flow of calcium ions into the nerve cells receiving the signals. There the calcium activates a membrane-bound calpain enzyme; the enzyme disrupts the membrane, and the disruption opens blocked glutamate re-



Dendritic spine after long-term potentiation had been induced

ceptors, making the synapse more responsive to future signals. On a broader scale, the disruption changes the shape of dendritic spines. The sequence seems to be independent of the biochemical mechanisms serving the "everyday" activity of the brain. Moreover, drugs are available that block calpain. Thus the hypothesis should be testable.

AIDS: In the Blood

To indict a microorganism as the cause of a disease it must be shown to be present in essentially all cases of the disease; it must be isolated and grown in culture; it must give rise to the disease when inoculated into a susceptible animal or human volunteer, and it must subsequently be recovered from the inoculated individual.

For the acquired immunodeficiency syndrome (AIDS), identification and isolation of a retrovirus (in France last year and in the U.S. four months ago) closely associated with the disease effectively fulfilled the first two postulates. A major step toward satisfying the other criteria has been reported in *Science* by a group of investigators. They state that a direct cause-and-effect relation between the virus and the disease has been demonstrated by medical accident.

Paul M. Feorino and his co-workers in a group headed by Donald P. Francis at the Centers for Disease Control in Atlanta have documented the first clear instance of a one-to-one transmission of AIDS between a donor and a previously uninfected recipient.

The recipient was a woman given a transfusion of packed red blood cells during an operation. Two months after surgery one of the donors of the red cells was found to be a homosexual male who had been hospitalized for AIDS. A year after surgery the woman, who had been exposed in no other way to the disease, was diagnosed as having AIDS. The diagnosis was confirmed when both she and the donor were shown to have antibodies to lymphadenopathy-associated virus (LAV), and the virus itself was isolated from the lymphocytes (immune-system cells) of both patients. LAV is the suspected AIDS agent discovered last year at the Pasteur Institute in Paris.

As HTLV-III (human T-leukemia/lymphoma virus) was not available to them the investigators were unable to test specifically for the presence of this organism, which was isolated by Robert C. Gallo's group at the National Cancer Institute (see "Science and the Citizen," July). Still, as Feorino and his colleagues point out, "the most likely explanation for the parallel evidence for HTLV-III and LAV being the cause of AIDS is that the two viruses are the same."

Before that likely proposition is proved, the first public-health benefits of the discovery of the AIDS agent should



"In Just A Few Days, I'll Show You How To Do

REAL MATH

On Your Calculator!"

$$\int_a^b f \quad \sum_{n=1}^x a_n \quad \frac{df}{dx} \quad \lim_{n \rightarrow \infty}$$

- Quick. •Guaranteed.
- Easy. •Fun, Too!

INTRIGUED BY CALCULATORS? Then you can step up your math skills fast! Use my new method in guidebook form. It's called **CALCULATOR CALCULUS**. This space-travel spinoff is sure-fire, so it has a **simple guarantee** — just return it for an immediate refund if you are not astounded at the problems you're solving with it!

But the point is — you won't want to send it back. For this is the **easiest, fastest shortcut** ever! The day you receive your copy in the mail you'll want to put it to work. It's that exciting and helpful.

My name is Dr. George McCarty. I teach math at the University of California. I wrote this guidebook to cut through the confusion. I guide you with **examples** you follow step-by-step on your calculator — you do simple **exercises** — then you solve practical **problems** with real precision!

POWER METHODS. Need to evaluate functions, areas, volumes — solve equations — use curves, trig, polar coordinates — find limits for sequences and series? It's all here!

If you're in the biological, social or physical sciences, you'll be doing Bessel functions, carbon dating, Gompertz' growth curves, half-life, future value, marginal costs, motion, cooling, probability, pressure — and plenty more (even differential equations).

Important numerical techniques? Those algorithms are here, too: rational and Padé approximation, bracketing, continued fractions, Euler's method, Heun's method, iteration functions, Newton's method, predictor-corrector, successive substitutions, Simpson's method and synthetic division.

LOOK AT WHAT USERS SAY: Samuel C. McCluney, Jr., of Philadelphia writes:

"**CALCULATOR CALCULUS IS GREAT!** For ten years I have been trying to get the theory of calculus through my head, using home-study courses. It was not until I had your book that it became clear what the calculus was all about. Now I can go through the other books and see what they are trying to do. With your book and a calculator the whole idea becomes clear in a moment, and is a MOST REFRESHING EXPERIENCE. I program some of the iterative prob-

lems you suggest and it always GIVES ME A THRILL to see it start out with a wild guess and then approach the limit and stop."

Professor John A. Ball of Harvard College (author of the book *Algorithms for RPN Calculators*) writes: "I wish I had had as good a calculus course."

Professor H. I. Freedman of the U. of Alberta, writing in *Soc. Ind. Appl. Math Review*, states: "There can be no question as to the usefulness of this book...lots of exercises...very clearly written and makes for easy reading."

Tektronix Engineer Bill Templeton says "**CALCULATOR CALCULUS** is the best, most clearly written book I have seen for improving your math skills."

I WANT YOU TO DO THIS. Get my complete kit, with a TI-35 calculator, plus its 200 p. Student Math Book, AND the guidebook, ALL for \$44.95 (for shipping to USA add \$2, or \$5 by AIR; Foreign \$5, or \$10 AIR; in Calif. add \$2.70 tax).

If you already have a scientific calculator, you can invest in the guidebook, **CALCULATOR CALCULUS**; for only U.S. \$19.95 (to USA or foreign: add \$1 for shipping, or \$4 by AIR; in Calif. add \$1.20 tax).

As pennywise Ben Franklin said, "An investment in knowledge pays the best dividends." GET STARTED NOW—Tax deductible for professionals.

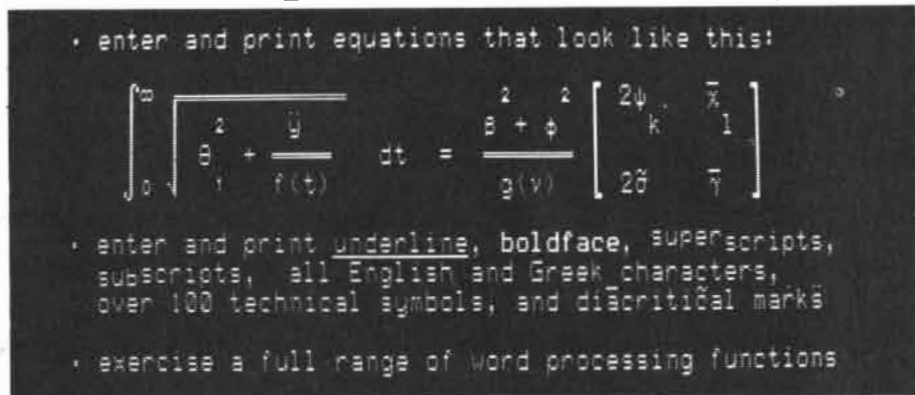
MONEY-BACK GUARANTEE! Send for it today. Be sure to give me your complete mailing address with your check or money order. If you want to charge it (Visa or MC), tell me your card no. and exp. date. Prompt shipment guaranteed.

George W. McCarty

Thank you! EduCALC Publications, Dept. A-9
27953 Cabot Road, Laguna Niguel, CA 92677

For fast service, phone MC or VISA orders
TOLL FREE to (800) 633-2252, Ext. 340.

Techwriter™ scientific WP lets your secretary



Actual unretouched photo of Techwriter screen presentation

- Available for: • IBM PC and XT
• DEC Rainbow • Apple II plus
• Apple IIe • Some IBM compatibles
• North Star Advantage • Visual 1050
• Seequa Chameleon *Registered trademarks

CMI Software

Originators of scientific word processing for PCs
1395 Main St., Waltham, MA 02154
617-899-7244

Please send information on Techwriter to:

Name _____

Company _____

Street _____

City _____

State _____ Zip _____

Phone _____

NEW FROM

BRIAN M. FAGAN

322 pages, 103 illustrations
Cloth: \$27.95; Paper: \$14.95



336 pages, 91 illustrations
Cloth: \$28.95; Paper: \$15.95

THE AZTECS

The Rape of the Nile author here constructs an irresistibly readable story of one of the world's most flamboyant, least understood early civilizations. Extensively illustrated, *The Aztecs* is unique in its chronological examination of the civilization as a smoothly functioning society at the height of its powers.

Fagan draws from a variety of research sources, including the codices compiled by sixteenth century friars, to convey the full flavor and detail of early Mexican life and culture. With its comprehensive maps, fine graphics, and wide-ranging photos, *The Aztecs* is certain to be the definitive book on the subject for many years.

CLASH OF CULTURES

This is a captivating journey through a momentous chapter of recent human history, the period in which Western civilization first came into contact with the full range of human biological and cultural diversity. Though the legacies of this chapter haunt us to this day, no book until now has fully explored the implicit consequences of this interaction.

Clash of Cultures is a book about myth and reality, about noble deeds and scurrilous dealings. Above all, it is a story of ordinary people pursuing their day-to-day goals, making decisions that were to have tragic and unimagined consequences in later generations.

Available at fine bookstores everywhere, or order from the publisher:



W. H. FREEMAN AND COMPANY
4419 West 1980 South
Salt Lake City, Utah 84104

If ordering by mail, please add \$1.50 for postage and handling; New York, California, and Utah residents add appropriate sales tax.

be apparent. A major group at risk are recipients of transfusions of blood and blood products. At least 100 individuals have contracted AIDS in this way. The need for a screening procedure is clear, and five groups have been licensed to develop test kits to detect antibodies to HTLV-III in blood samples. The first use of such kits will surely be to screen blood donations in an effort to prevent such cases as the one reported by Feorino and his colleagues.

A Fine Madness

The brown hare (*Lepus capensis*) enjoys an extended breeding season. During its course a single female may bear several litters, each comprising as many as seven young. Nevertheless, it has long been thought by biologists and lay people alike that males of the species are particularly randy in March, when they are said to chase one another frenetically and box in a mad competition for estrous females.

Diligent observation by a British amateur naturalist appears to put this interpretation of hare madness in error. Since 1977 Anthony J. F. Holley, a solicitor from Brent Knoll in Somerset, has spent more than 1,500 hours observing brown hares from the roof of his house, which he deliberately built for this purpose amidst fields well populated with the animals. Aided by a telescope, Holley has often been able to distinguish males from females and even to recognize individual hares by their scars and facial markings.

Writing in *Nature*, he and Paul J. Greenwood of the University of Durham report that male brown hares are apt to chase each other at any time during the January-through-August breeding season. Boxing, in which one hare typically rises on its hind legs and lunges at another with its forepaws, also takes place throughout that period.

Boxing among hares, Holley and Greenwood report, is not a male sport: in each of the 17 bouts in which Holley was able to discern the sex of the combatants, a male was pitted against a female. Apparently boxing is not a way for two males to decide which one will get a female but rather the means by which the larger and heavier female rejects an unwanted suitor.

Why then has the March male-madness myth persisted for so long? Holley and Greenwood point out that brown hares are nocturnal animals and in the winter generally do not emerge from their burrows until after sunset. On long summer evenings they do come out while it is still light but by then tall grass often conceals them (unless one watches from a rooftop). Their peculiarities are thus most apparent in early spring, when the days, but not the grass, have started to grow longer.

PC FORTH™
IBM PC & XT,
HP-150,
Macintosh,
Apple II,
CompuPro,
Sage & CP/M-68K,
Wang PC,
 All CP/M and
 MSDOS computers.

Try the professional language offering the utmost performance in the shortest development time. Transport your applications between any of our enhanced 83-Standard compilers or expanded 32-bit versions. Choose from our wide selection of programming tools including native code compilers, cross-compilers, math coprocessor support, and B-Tree file managers. All fully supported with hotline, updates, and newsletters.

Laboratory Microsystems Incorporated
 Post Office Box 10430, Marina del Rey, CA 90295
 Phone credit card orders to (213) 306-7412

**PROGRAMMING'S GREAT IN THEORY . . .
 . . .BUT LET'S GET PRACTICAL!**

Want to know what your computer is really thinking?

Ask **CodeSmith™-86** — the elegant Multi-Window "debugging" utility for MS-DOS, already in use world-wide by experienced & aspiring Programmers alike.

A powerful Machine-language learning aid to boot!

WATCH Your 8088's CPU at work as **CodeSmith™** vividly steps thru computer instructions with its scrolling symbolic display.

STOP & READOUT your computer with a single keystroke.

MODIFY registers and memory with new data — just type it in on the screen.

MARK your program code with comments & breakpoints.

COUNT how many times a given path is taken thru code.

TRACKDOWN & NAIL those hard-to-find bugs with STOP-on-DATA-COMPARE.

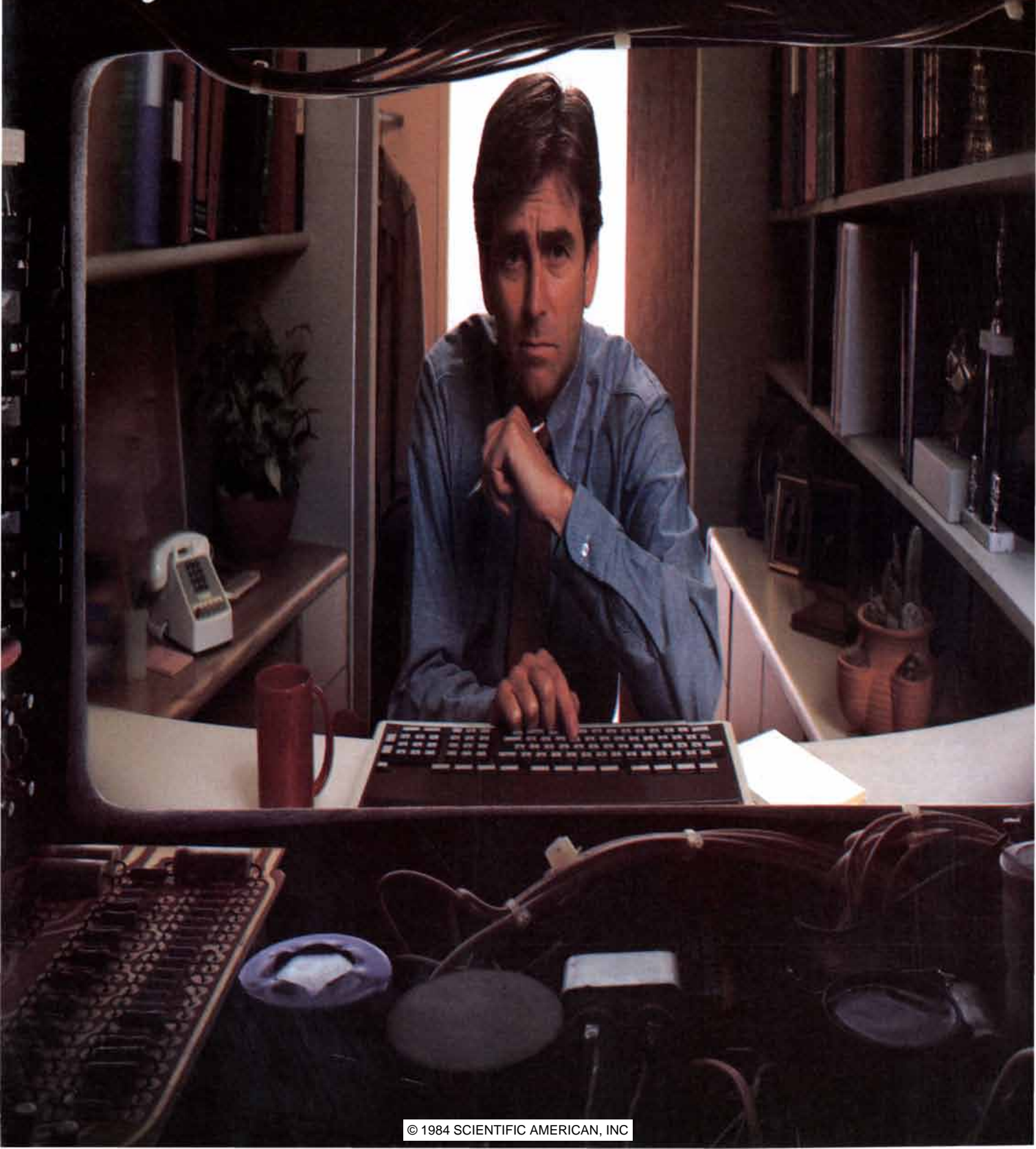
Requires 160K RAM
\$145⁰⁰

ORDER YOUR **CodeSmith™-86** TODAY.

VISUAL AGE
 642 N. Larchmont Blvd. • L.A., CA 90004 • (213) 439-2414
 CodeSmith™ International Arrangements, Inc. MS, TM Microsoft Corp.

COD/Blue Label

Ordinary computer systems
take a very narrow view
of your world.



The extraordinary MAPPER[®] System puts things in true perspective.

Ordinary computer systems see you and your special business problems the way their programs tell them to. Which means they can't see things quite right.

That's because computer programs are either off-the-shelf — designed for some mythical "typical" business. Or they're custom-designed — created by an outsider who's unlikely to have a complete picture of your business.

But meet the Sperry MAPPER System.

It's truly an extension of your mind. Instead of confining you to any fixed program, MAPPER gives you carte blanche to create your own programs. Without doing any programming, in the conventional sense.

As a result, you gain unheard-of power. To make the computer see the full scope of your real world, and deal with it realistically.

MAPPER adapts to the way you work.

You and your people don't have to be computer experts to use MAPPER expertly. Far from it. Simple but powerful English commands get you what you want. A word or two is usually enough.

As you insert and extract information, freed from rigid procedures, you create your program along the way. So you can make mistakes. Change your mind. Even alter your destination in mid-journey.

MAPPER does more than work the way you work. It actually follows the way you think.

MAPPER can serve a handful. Or handle a thousand.

The MAPPER System can be scaled for a major corporation, for a single department, or for a growing business. You can even timeshare through a Sperry service bureau.

So no matter what size your business, the power of

MAPPER is affordable. In fact, it can cost as little as a network of ordinary personal computers.

Before you invest in any ordinary computer system, invest a little time with Sperry. Come see MAPPER in action — hands-on. Phone toll-free **800-535-3232**. Or send us the coupon.

MAPPER is a trademark of Sperry Corporation. © Sperry Corporation 1984

Sperry Corporation, Computer Systems, Department 100, Box 500,
Blue Bell, PA 19424-0024

Please phone me to arrange a demonstration.

send me information on the MAPPER system.

NAME _____

TITLE _____

COMPANY _____

STREET _____

CITY _____

STATE _____

ZIP CODE _____

TELEPHONE _____



Operating Systems

A computer operating system spans multiple layers of complexity, from commands entered at a keyboard to the details of electronic switching. The system is organized as a hierarchy of abstractions

by Peter J. Denning and Robert L. Brown

At a terminal connected to a computer system you type the command *date* and press the key marked *Return*. Almost instantly the message *September 15, 1984*, appears on the display screen. Asking for the current date would seem to be among the simpler demands one might make of a computer, and yet it sets in motion a complex series of events calling into action many of the hardware and software resources of the system. Coordinating the events and managing the resources are among the responsibilities of the collection of programs called the computer operating system. The operating system provides facilities and services needed by almost all other software.

Consider what must happen in order to answer a request for the date. As each character of the command is typed, the keyboard transmits a code to the computer, where it is received by a circuit board charged with handling communication with the terminal. The board stores each character code in a reserved area of memory called a buffer and issues a signal that "interrupts" the central processing unit of the computer, activating a program called the terminal driver. The terminal driver echoes a copy of the code back to the terminal for display on the screen.

When the code for the *Return* key is received, signifying that the typing of the command is complete, the terminal driver activates another program called the listener (because it attends to requests from users). The listener reads the characters *d a t e* from the keyboard buffer, searches a magnetic-disk memory for a program called *date*, loads the program into main memory and starts its execution. The *date* program in turn consults a clock built into the hardware, which maintains a count of the milliseconds that have passed since some fixed starting date. From the count the program calculates the month, day and year and expresses the information as the string of characters *September 15, 1984*. The string is passed to the terminal-driver program, which transmits the binary

code for each character to the terminal, where it appears on the display screen.

Each of these events could be described in even finer detail. For example, before the listener can load the *date* program, it must first search a directory of commands to find out where on the disk the program is stored; indeed, the directory itself must be read from the disk. The disk is organized in concentric tracks, and each track is divided into sectors; hence instructions must be issued to position the disk head over the appropriate track and to read the binary data when the selected sectors pass under the head. The resulting stream of bits is stored temporarily in a buffer. When the program is loaded, space in main memory must be allocated to it; when it has finished executing, the space must be reclaimed. The sequence of events is still more complicated in a computer dealing with several programs at once. In that case one program may have to be suspended momentarily while the central processor attends to another; then the first program must resume exactly as if there had been no interruption.

It can be seen from this example that an operating system spans the entire range of complexity found in computer systems. Some parts of the operating system interact directly with the hardware of the computer, where events (such as the switching of individual logic gates) can have a time scale as brief as a few billionths of a second. At the other end of the spectrum, parts of the operating system communicate with the user, who issues commands at a much more leisurely pace, perhaps one every few seconds. A single keystroke at the terminal might result in 10 calls on the operating-system programs, in the execution of 1,000 machine instructions and in 10 million changes of state in logic gates.

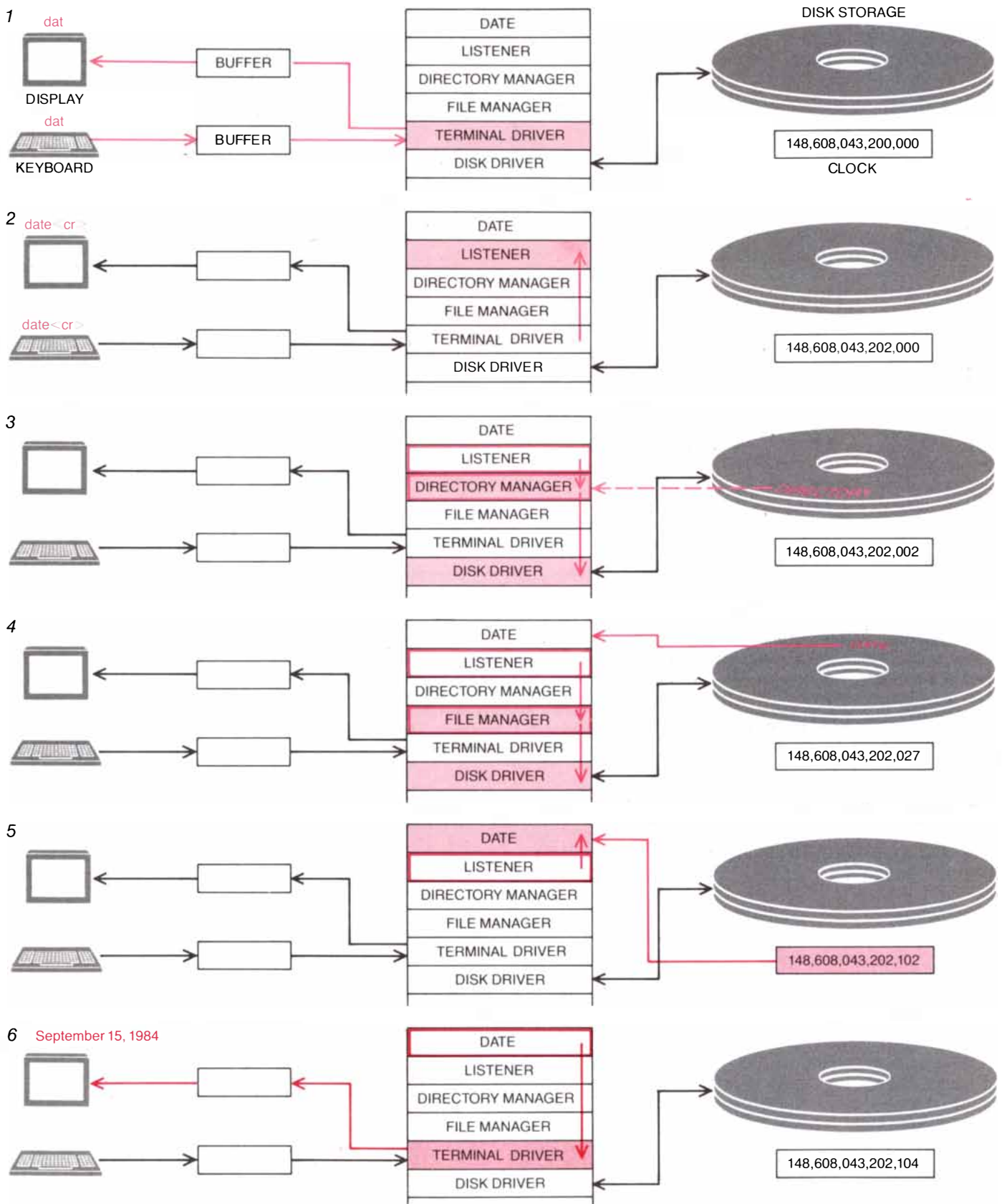
The strategy adopted for managing this complexity is one that has proved to be of crucial importance in virtually all areas of computer science. The basic idea is to create a hierarchy of levels of abstraction so that at any level one can

ignore the details of what is going on at all lower levels. Thus when the listener program loads a program from disk storage, the listener need not specify the positioning of the disk head; such mechanical operations are done by a program at a lower level in the hierarchy. At the highest level of all is the user of the system, who ideally is insulated from everything except what he aims to accomplish.

The first operating systems were created for the first electronic computers in the late 1940's. They were sets of simple routines for input and output, such as a program for storing binary codes read from a punched paper tape into successive memory locations. The entire operating system consisted of a few hundred machine instructions.

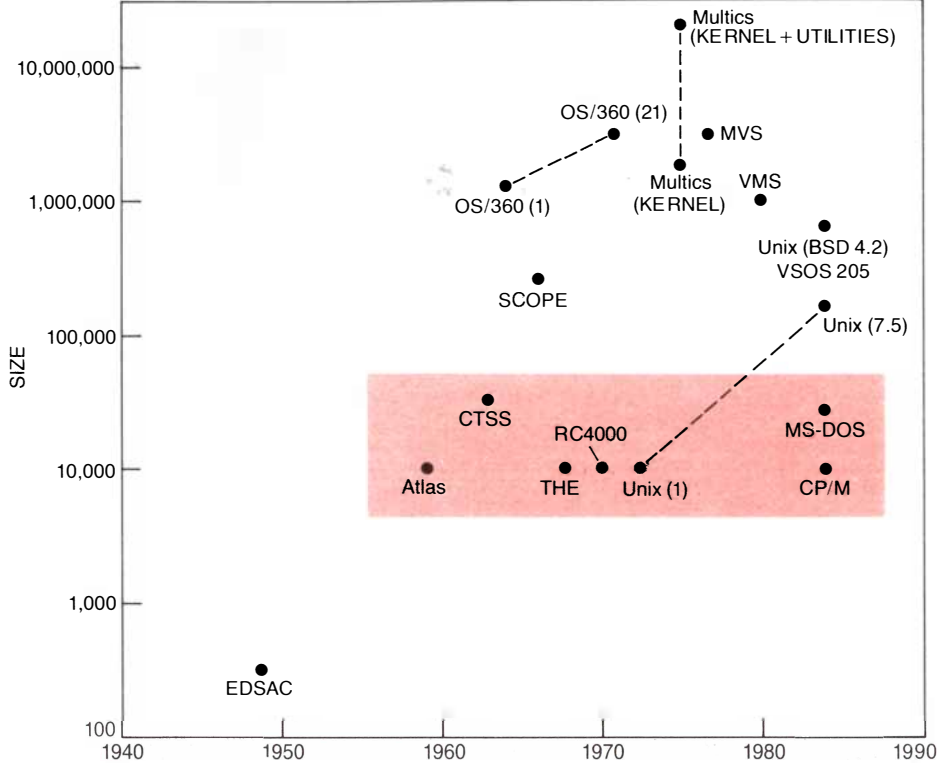
By the mid-1950's most computers were being run in "batch mode." An operating system collected programs submitted by many individuals and executed them in rapid succession, thereby eliminating the delays entailed in manually loading one program at a time. Operating systems of this kind were called supervisors or monitors. In addition to their primary function of program loading they managed secondary storage devices (such as magnetic disks, drums and tapes), allocated main memory and handled input and output. In most cases they also included a software "library" of commonly needed routines. For example, many computer applications call for the sorting of information; if a versatile sorting routine is part of the library, the operating system can load it along with each program that needs it.

By 1960 the first time-sharing systems were being designed. In this method of operating a computer the attention of the central processor is switched rapidly among several user programs, giving all the users the illusion that their programs are executing simultaneously. In constructing such systems the problems of sharing the processor, the memory and the various software resources had to be addressed. Solving those problems gave



EXECUTION OF A COMMAND sets in motion events at several levels in the hierarchy of programs that make up the operating system. The command is simply a request for the date. As each character is typed at a keyboard (1) it is received by a terminal-driver program, which echoes it to the display screen. When a carriage return is entered, the terminal driver passes the string of characters *d a t e* to the listener program (2), which interprets it as the name of a command. The listener asks the directory manager to search the directory of commands for *date*. The directory manager in turn asks the disk driver to copy the directory into a buffer in the directory manager's stor-

age space (3). When the command has been found, the listener directs the file manager to load the binary code for the *date* program into memory; to do this the file manager again uses the disk driver (4). The listener then activates the *date* program, which reads a "clock" (5), a hardware device that keeps a count of the milliseconds that have passed since some fixed starting time, in this case midnight of January 1, 1980. From this number the program calculates the current date and displays it through the terminal driver as September 15, 1984 (6). The listener and the various drivers and managers constitute part of the "kernel" of the operating system; *date* is a utility program.



EVOLUTION OF OPERATING SYSTEMS suggests a tendency toward exponential growth, but some smaller systems have been introduced recently for microcomputers. The size is given in units that correspond to “words” of machine storage or to assembly-language instructions. EDSAC was developed at the University of Cambridge, Atlas at the University of Manchester, CTSS at the Massachusetts Institute of Technology, THE at the Eindhoven University of Technology and RC4000 at the University of Denmark. Scope is a product of the Control Data Corporation, and so is VSOS 205; OS/360, MVS and VMS are products of the International Business Machines Corporation. Multics was developed jointly by M.I.T. and Bell Laboratories, Unix by Bell Laboratories alone. CP/M and MS-DOS are microcomputer operating systems introduced respectively by Digital Research, Inc., and the Microsoft Corporation. The systems within the band of color are single-machine kernels, probably of the minimum possible size.

LEVEL	NAME	OBJECTS	EXAMPLE OPERATIONS
13	SHELL	USER PROGRAMMING ENVIRONMENT	STATEMENTS IN SHELL LANGUAGE
12	USER PROCESSES	USER PROCESSES	QUIT, KILL, SUSPEND, RESUME
11	DIRECTORIES	DIRECTORIES	CREATE, DESTROY, ATTACH, DETACH, SEARCH, LIST
10	DEVICES	EXTERNAL DEVICES SUCH AS PRINTERS, DISPLAYS AND KEYBOARDS	CREATE, DESTROY, OPEN, CLOSE, READ, WRITE
9	FILE SYSTEM	FILES	CREATE, DESTROY, OPEN, CLOSE, READ, WRITE
8	COMMUNICATIONS	PIPES	CREATE, DESTROY, OPEN, CLOSE, READ, WRITE
7	VIRTUAL MEMORY	MEMORY SEGMENTS	READ, WRITE, FETCH
6	LOCAL SECONDARY STORAGE	BLOCKS OF DATA, DEVICE CHANNELS	READ, WRITE, ALLOCATE, FREE
5	PRIMITIVE PROCESSES	PRIMITIVE PROCESSES, SEMAPHORES, READY LIST	WAIT, SIGNAL, SUSPEND, RESUME
4	INTERRUPTS	FAULT-HANDLING PROGRAMS	INVOKE, MASK, UNMASK, RETRY
3	PROCEDURES	PROCEDURE SEGMENTS, CALL STACK, DISPLAY	MARK STACK, CALL, RETURN
2	INSTRUCTION SET	EVALUATION STACK, MICROPROGRAM INTERPRETER, SCALAR DATA, ARRAY DATA	LOAD, STORE, BRANCH, ADD, SUBTRACT
1	ELECTRONIC CIRCUITS	REGISTERS, GATES, BUSES, ETC.	CLEAR, TRANSFER, ACTIVATE, COMPLEMENT

HIERARCHY OF ABSTRACTIONS is the essential organizing principle of an operating system. Each level is the manager of certain “objects,” which can be hardware or software. A program at a given level has access only to operations defined at lower levels; furthermore, the internal details of those operations are hidden. The first seven levels concern operations within a single machine; higher levels can draw on the resources of multiple computers in a network.

rise to a number of important conceptual advances, including parallel-process synchronization, virtual memory, device-independent input and output and interactive command languages, all of which we shall discuss below:

As operating systems became more elaborate they also grew larger. The Compatible Time Sharing System, put into operation at the Massachusetts Institute of Technology in 1963, consisted of approximately 32,000 36-bit words of storage. OS/360, introduced a year later by the International Business Machines Corporation, had more than a million machine instructions. By 1975 the Multics system, developed by M.I.T. and Bell Laboratories, had grown to more than 20 million instructions.

By then, however, a countervailing influence was being felt: minicomputers had entered the marketplace and microcomputers (including personal computers) were beginning to appear. These machines were slower and had a smaller memory capacity than the mainframe machines of the time, but they extended access to computing to a much broader range of potential users. In order to squeeze operating systems into the smaller accommodations of mini- and microcomputers the functions of the system were divided. Services needed by almost all programs, such as input and output routines, were put in a “kernel” that remains in the main memory of the computer whenever it is running. Other programs, called system utilities, are stored on disk and read into main memory only when they are needed. Judging from the operating systems introduced in the past several years, it appears the minimum kernel needed to manage the resources of a single computer consists of a few tens of thousands of instructions. The available utilities and libraries of software are continuing to grow almost exponentially, straining the capacity of secondary-storage facilities.

The evolution of operating systems has not ended. A new population of users, including many who do not make computing a full-time occupation, has placed new demands on software. One response has been the development of interactive graphic displays. With such a display one might delete a file not by typing the command *delete* but by pointing to a drawing of a trash can. New ways of organizing a computer system have also evolved. Instead of having a single large computer connected to many terminals, each user can be given a work station that has its own processor and communicates with other work stations by means of a high-speed network. It is the operating-system software that must coordinate the actions of the various computers in such a distributed-processing network.

The hierarchical structure of a mod-



computer/electronics book club

The best of both worlds!

hands-on interfacing projects
the latest electronic applications
all from *one easy-to-use source!*

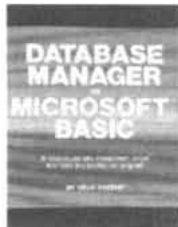
Select 5 Books for Only \$2⁹⁵



1466
List \$18.95



1665
List \$21.95



1567
List \$18.95



1553
List \$15.95



1604
List \$24.95



1295
List \$16.95



1650
List \$19.95



1673
List \$13.95



1556
List \$21.95



1341
List \$13.50 (paper)



1474
List \$16.95



1394
List \$15.95



1537
List \$16.95



1679
List \$17.95



1389
List \$15.95



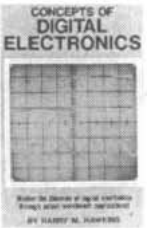
1682
List \$14.95



1536
List \$14.95



1195
List \$13.95



1531
List \$17.95



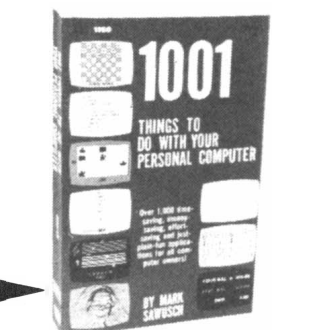
1541
List \$24.95



1539
List \$19.95



1409
List \$15.95



Plus FREE For Joining →

7 very good reasons to join the computer/electronics book club

- **Big Savings.** Save 20% to 75% on books sure to increase your computer and electronics know-how
- **No-Risk Guarantee.** All books returnable within 10 days without obligation
- **Club News Bulletins.** All about current selections—mains, alternates, extras—plus bonus offers. Comes 13 times a year with hundreds of up-to-the-minute titles you can pick from
- **"Automatic Order."** Do nothing, and the Main selection will be shipped automatically! But . . . if you want an Alternate selection—or no books at all—we'll follow the instructions you give on the reply form provided with every News Bulletin
- **Bonus Books.** Immediately get a Dividend Certificate with every book purchased and qualify for big discounts of 60% to 80%
- **Extra Bonuses.** Take advantage of added-value promotions, plus special discounts on software, games, and more
- **Exceptional Quality.** All books are first-rate publisher's editions selected by our Editorial Board and filled with useful, up-to-the-minute information

computer/electronics book club

P.O. Box 110, Blue Ridge Summit, PA 17214

Please accept my membership in the **computer/electronics book club** and send the 5 volumes circled below, *plus* my FREE copy of *1001 Things To Do With Your Personal Computer*, billing me \$2.95 plus shipping and handling charges. If not satisfied, I may return the books within ten days without obligation and have my membership canceled. I agree to purchase 3 or more books at reduced Club prices (plus shipping/handling) during the next 12 months, and may resign any time thereafter.

- 1195 1218 1295 1341 1389 1394 1409 1449 1466
1474 1531 1536 1537 1539 1541 1553 1556 1567 1604
1625 1650 1665 1671 1673 1679 1682 1712 1748

Name _____ Phone _____
Address _____
City _____
State _____ Zip _____

Valid for new members only. Foreign applicants will receive special ordering instructions. Canada must remit in U.S. currency. This order subject to acceptance by the computer/electronics book club.

XSA-984

ern operating system separates its functions according to their complexity, their characteristic time scale and their level of abstraction. The bottom illustration on page 96 shows an organization spanning 13 levels. It is not a model of any particular operating system but rather incorporates ideas from several systems; facilities for distributed processing are included. Each level is the manager of a set of "objects," which can be hardware or software and whose nature varies greatly from level to level. Each level also defines the operations that can be carried out on those objects.

The lowest levels include the hardware of the system. Level 1 is that of electronic circuits, where the objects are registers, memory cells, logic gates and so on. The operations defined on these objects are actions such as clearing a register or reading a memory location. Level 2 is that of the processor's instruction set, which can deal with somewhat

more abstract entities, such as an evaluation stack (a sequence of registers or memory cells where numerical values are held pending some operation to be carried out on them). The operations at this level are the instructions the processor itself can execute, such as *add*, *subtract*, *load* and *store*.

Level 3 adds the concept of a procedure, or subroutine, a self-contained program fragment that can be called on within a larger program and that returns control to the point at which it was called. Level 4 introduces interrupts, which cause the processor to save a record of its current state and then turn to a new task. Events that trigger an interrupt include error conditions, such as the overflow of an arithmetic register, and more commonplace events, such as the receipt of a character code from a terminal.

The first four levels together correspond roughly to the basic machine as it

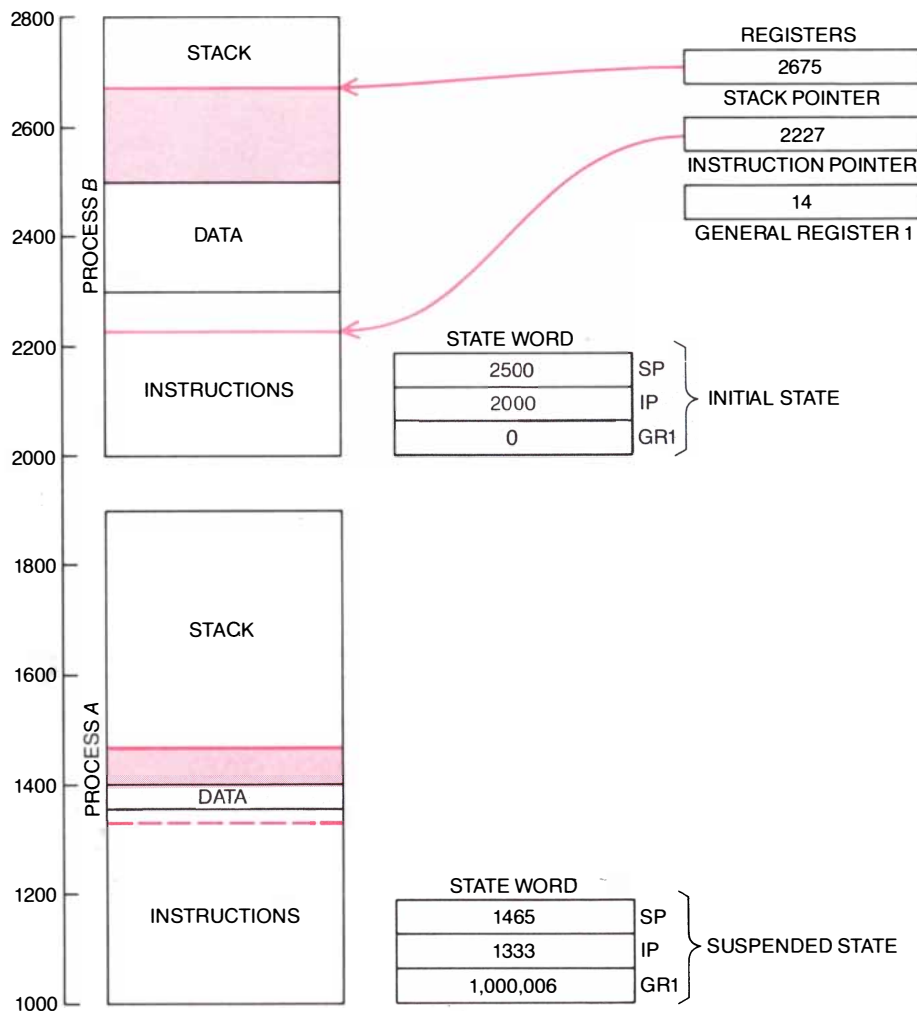
is provided by the manufacturer, although there are close interactions with some elements of the operating-system kernel. For example, interrupts are generated by a hardware component, but the routines invoked when the processor is interrupted are part of the kernel.

Concepts associated explicitly with the coordination of multiple tasks first appear at level 5, which is identified as the level of "primitive processes," or single programs in the course of execution. Because a primitive process may be interrupted at any time, a mechanism is needed to suspend a process and then resume it. The mechanism consists of the "state word," a data structure that can hold the contents of all the registers in the central processor, and a "context switch" operation. When a process is to be suspended, the context-switch operation copies the register values into the state word for that process; on resumption it restores the registers to their former values.

If all the activities going on within a computer were completely independent of one another, little more than the notion of a state word would be needed to create a multiple-process operating system. Actually one process often depends on results from another, so that the processes must be synchronized. A program that requires data from a file on disk, for example, cannot proceed until the data have been read and made available in main memory. It is not possible for the programmer to know beforehand how long the disk-reading operation will take, and so there must be a way to make one program wait until another signals it is ready.

The concept that provides the key to synchronization is the semaphore; it occupies a central place in the theory of operating systems. In the simplest case it is helpful to think of a semaphore as being directly analogous to a railroad signal, with green and red lights that indicate whether or not it is "safe" for a process to continue. At the point where a process must be synchronized with some external routine, the programmer inserts an instruction such as *wait(semaphore A)*. Each time that point in the program is reached the semaphore is inspected. If it is in the red state, execution of the process is suspended; if it is green, the process continues but the semaphore is set to red. When the second process issues a *signal(semaphore A)* instruction, the semaphore is reset to green and the first process resumes execution if it has been waiting.

Actually, because multiple processes may be controlled by the same semaphore, the implementation of the semaphore must be somewhat more complicated: it must maintain a counter and a queue of waiting processes. For each *wait* instruction the counter is dec-



PRIMITIVE PROCESS represents a single program in the course of execution. Here two primitive processes are shown loaded into a segment of main memory. Process B is executing. The instruction pointer, one of the computer's hardware registers, indicates the address of the next instruction; the stack pointer indicates the top item in the temporary storage area called the stack. A real computer would have many more general-purpose registers, but here only one register is shown. Process A has been suspended, but the contents of all the registers at the moment of suspension were recorded in a reserved area called the state word. The operating system can readily switch between processes. The current contents of the registers are put in the state word of process B, and the registers are reloaded with the values from process A's state word.

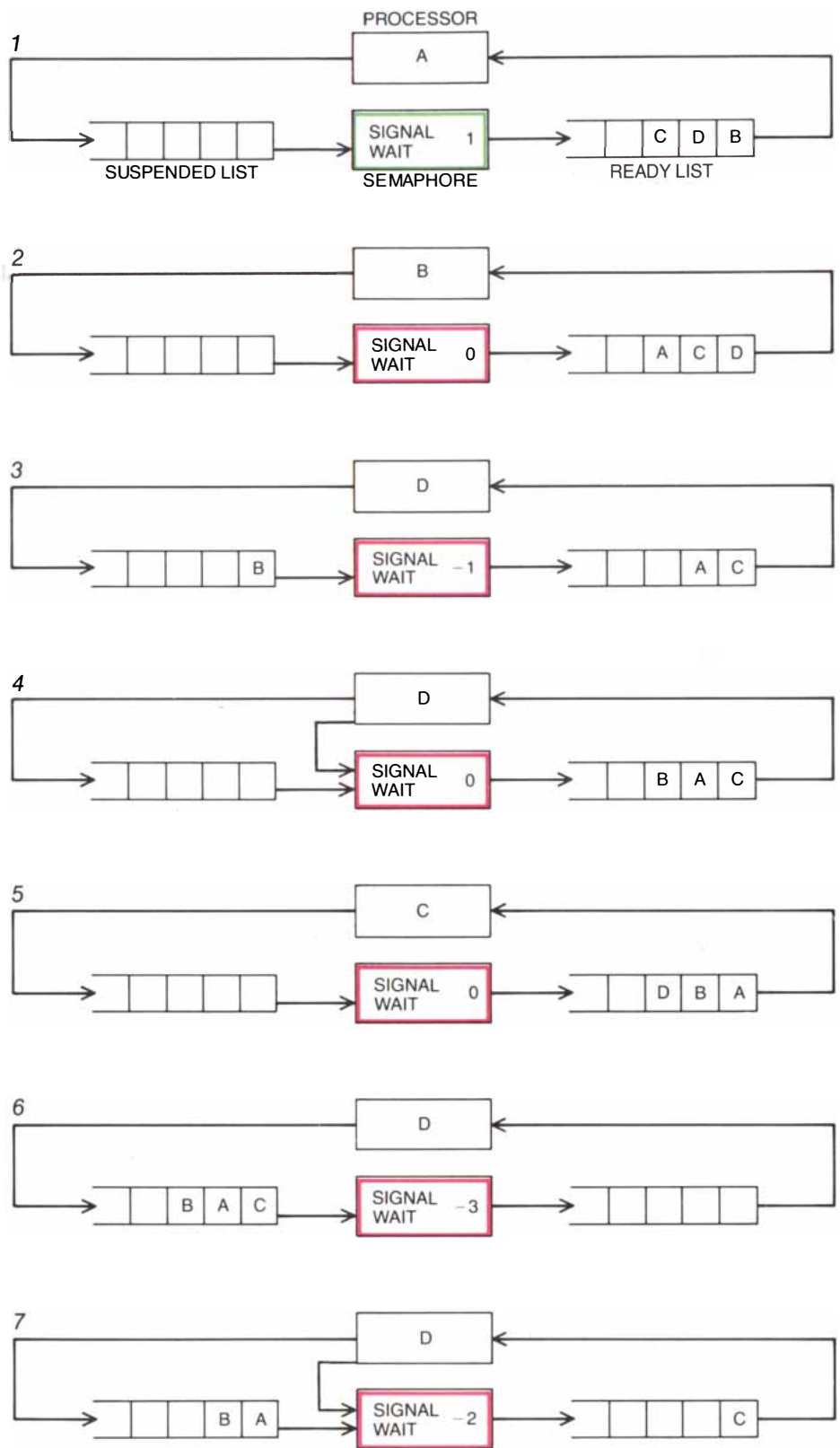
remented and for each *signal* instruction it is incremented. If the value of the counter is negative, any process issuing a *wait* instruction is put in the queue; when the next *signal* instruction is received, the first process in the queue is transferred to the "ready list" of processes available for execution. The two operations defined on semaphores are powerful enough to synchronize parallel processes in a variety of contexts, from the need to stop a process when an input buffer is empty or an output buffer is full to the need to allow only one process at a time to manipulate shared data.

Level 6 in the operating-system hierarchy handles access to the secondary-storage devices of a particular machine. The programs at this level are responsible for operations such as positioning the head of a disk drive and reading a block of data. Software at a higher level merely determines the position of the data on the disk and places a request for it in the device's queue of pending work. The requesting process then waits at a semaphore until the transfer has been completed.

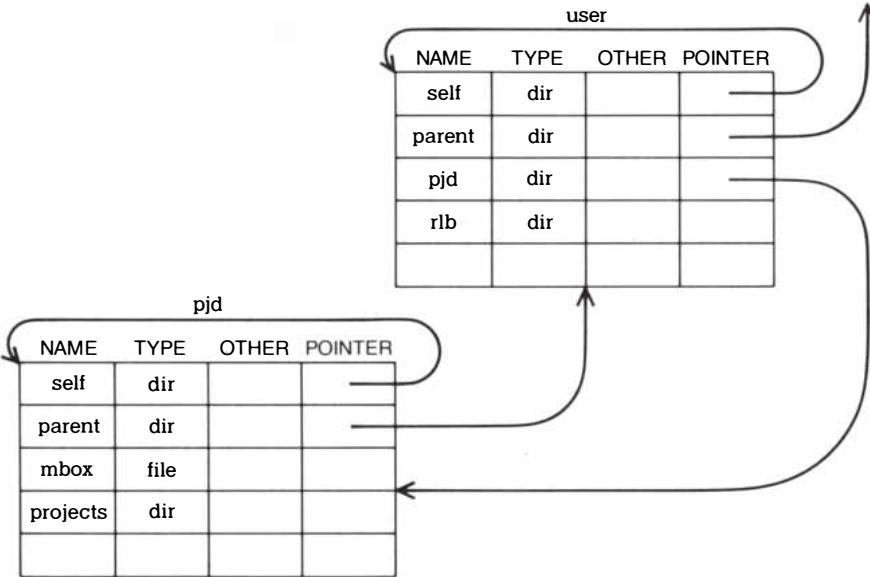
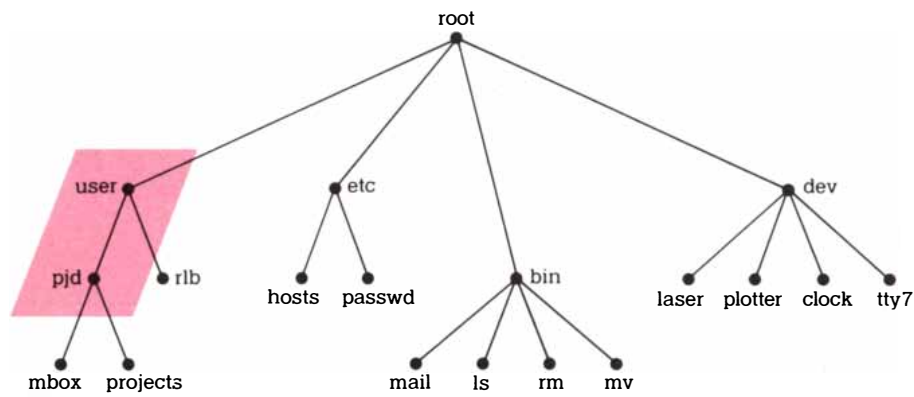
The function of level 7 is virtual memory, a scheme for managing the computer's main and secondary memories that gives the programmer the illusion of having a main memory large enough to hold a program and all its data even if the available capacity of main memory is much smaller. Addresses can be arbitrarily large, and programs running concurrently can employ the same addresses without conflict; the operating system translates each virtual address into a hardware address. If an attempted translation fails because the information called for is not in main memory, the virtual-memory manager automatically fetches it from disk storage. Before doing so it may have to make room in main memory by removing other data. As in other similar circumstances the requesting process is interrupted until the needed information is made available.

Up to level 7 the operating system deals exclusively with the resources of a single machine. Beginning with the next level the programs of the operating system encompass a larger world including peripheral devices such as terminals and printers and also other computers attached to the network.

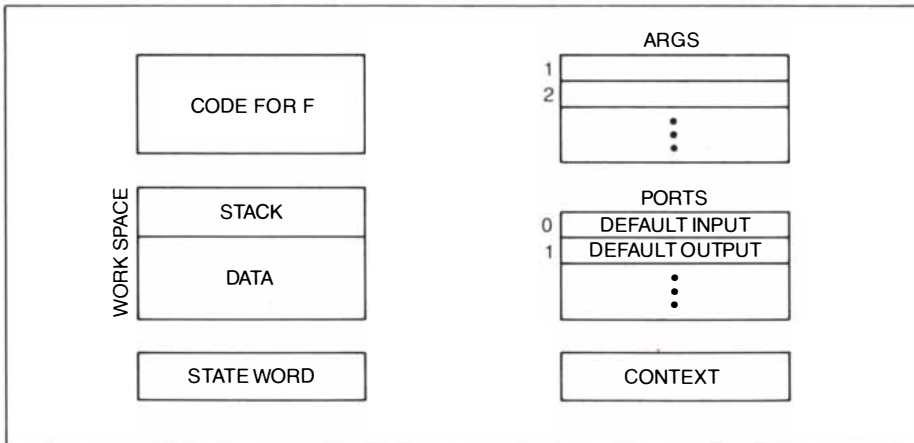
Level 8 deals explicitly with communication between processes, which can be arranged through a single mechanism called a pipe. A pipe is a one-way channel: a stream of data flows into one end and out of the other. The stream has a write pointer, which keeps track of the number of items written into the pipe, and a read pointer, which records the number of items read at the other end. A request to read more items is delayed until they are actually present. A pipe can connect two processes



SOFTWARE SEMAPHORE is a mechanism for controlling primitive processes that must be synchronized. Here processes A, B and C all depend on a result from process D. For each *wait* instruction the semaphore decrements a counter and for each *signal* instruction it increments it. A waiting process is allowed to pass the semaphore only if the value of the counter is greater than zero. Initially (1) process A is running, B, D and C are ready and the semaphore count of +1 indicates that one of D's results is available. When A issues a *wait* instruction, it therefore immediately passes the semaphore and rejoins the ready list. Then B runs (2), eventually issues a *wait* instruction and is suspended, allowing D to run (3). When D completes a new result, it issues a *signal* instruction, which allows B to move to the ready list (4). D rejoins the ready list and C begins to run (5) but is suspended when it issues a *wait* instruction; in the same way A and C run and are suspended, allowing D to resume execution (6). When D has a result, it issues a *signal*, transferring C to the ready list (7); later cycles of D will release A and B from suspension.



TREE OF FILES AND DIRECTORIES organizes the resources of a computer system. The root of the tree and the intermediate nodes are directories that can list either files or subordinate directories. The “bin” directory, for example, holds the binary code of system utility programs, such as programs for electronic mail and for listing, moving or removing files. Similarly, the “dev” directory lists devices and the “etc” directory holds miscellaneous information such as the host computers available and the encrypted passwords of users. The structure of the “user” directory, where each user of the computer system keeps his own files, is shown in somewhat greater detail in the lower part of the illustration. Each directory has a pointer to itself and to its parent. The directory structure represented here is based on that of the Unix operating system.



USER PROCESS is a virtual computer: a simulated machine that appears to be dedicated to executing a single program. A user process incorporates the elements of a primitive process (the executable code, the work space and the state word) as well as a list of arguments supplied when the program was started, a list of ports for input and output and a description of the program’s context. The arguments are parameters typed after the command name; they are entered into successive slots of the **ARGS** array. The ports include two for “default” input and output, which serve unless other ports are specified. The context lists such items as the working directory.

executing on a single machine, as when the output of one program is designated the input of another. A pipe can equally well transmit information between computers; indeed, a set of pipes linking processes in all the machines of a network can serve as a broadcast facility, which is useful for finding resources that might be anywhere in the network.

The file system, which provides for the long-term storage of named files, is implemented at level 9. Whereas level 6 deals with disk storage in terms of tracks and sectors—the fixed-size divisions of the hardware itself—level 9 addresses more abstract entities of variable length, whose boundaries do not necessarily correspond to those of physical tracks and sectors. Indeed, a file may be scattered over many noncontiguous sectors.

The *create* and *destroy* operations set up a new file and discard an old one; *open* and *close* make and break the connection between a file and a process. For the content of a file to be examined it must be copied into an area of virtual memory, and for information to be saved it must be copied from virtual memory into a file; the copying is done by *read* and *write* operations. If a file is kept in a different machine, level-9 software can, by using level 8, create a pipe to the file’s home machine. (The best way of accomplishing this is still an open question.)

Level 10 provides access to external input and output devices, including the time-of-day clock, printers, plotters and the keyboards and display screens of terminals. The operations defined on these objects are again *create* and *destroy*, *open* and *close*, *read* and *write*; again a pipe can be created to gain access to a device attached to another machine.

Level 11 manages a hierarchy of directories that catalogue the hardware and software objects to which access must be controlled: pipes, files, devices and the directories themselves. The central element of a directory is a table that matches the external name of an object (that is, the name known to and supplied by the user, such as “addresslist”) to an internal name employed by the operating system to find the object. A hierarchy arises because a directory can include among its entries the names of subordinate directories.

Each directory is a list of entries giving an object’s external name (stored as a string of characters), the internal name (stored as a binary code), an indicator of its type (file, device and so on) and certain other information. For example, the directory entry commonly reveals whether the object can be read from or written to or, in the case of program files, executed; each kind of access might be allowed for some users but not for others.

The directory level is responsible only

Step into Toyota's 1984 Cressida, and let the luxury begin.

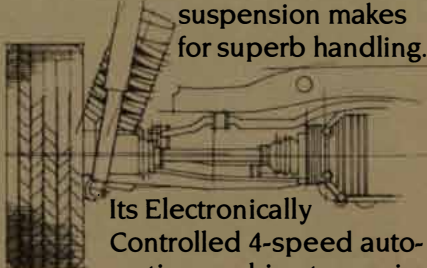
Inside: seats with a rich velour fabric, or a soft full-grain leather.



Cruise control for relaxed driving. And now, as standard equipment, power windows, power door locks, an electronic AM/FM/MPX stereo with 4 speakers, cassette, seven band graphic equalizer, and automatic power antenna.

Outside, more luxury. Classic, elegant lines. A sloping hood that reduces aerodynamic interference for quieter motoring. Roof height precisely scaled to allow plentiful headroom.

Cressida's luxury soothes you, but its performance excites you. Its 2.8 liter electronically fuel-injected Twin Cam engine produces 143 horsepower at your command. Its independent rear suspension makes for superb handling.



Its Electronically Controlled 4-speed automatic overdrive transmis-

OH WHAT A FEELING!
TOYOTA

sion with lock-up torque converter lets you choose at the touch of a button a "Normal," "Power" or "Economy" mode of driving. This "thinking" transmission adjusts to suit changing driving conditions.

And on

and on

and on.

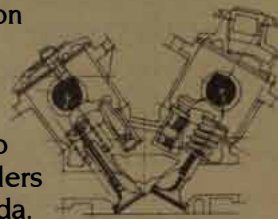
There is

no end to

the wonders

of Cressida.

Ahhh what a feeling. The Cressida Luxury Sedan.



BUCKLE UP—IT'S A GOOD FEELING!

**CRESSIDA. LUXURY THAT SOOTHES THE SOUL.
PERFORMANCE THAT LIFTS THE SPIRIT.**



AHHH!

Exxon research is today's cars by keeping

New fuels and lubricants must anticipate advancing technology and increasing performance demands.

Turbochargers, electronics, advanced transmissions, knock sensors, fuel injectors and other sophisticated devices are being rapidly incorporated into today's automobiles. While they permit more precisely controlled and optimized vehicle operation, they also place greater demands on fuels and lubricants. Keeping ahead of the changing requirements has challenged scientists and engineers at Exxon Research and Engineering Company (ER&E), and their Exxon colleagues at three major affiliated laboratories outside the U.S.

Hotter Engines

Today's engines are smaller. They operate at temperatures some 50° to 75°F hotter than their older V-8 counterparts. Turbocharged engines run even

hotter. Higher temperatures can cause motor oils to oxidize faster, producing sludge and varnish deposits which thicken the oil. This in turn can lead to greater friction and increased engine wear.

In the 1970's, ER&E scientists and engineers discovered an additive technology which resulted in the first fuel-saving motor oil using oil-soluble friction modifiers. Today, they are creating new oils for the hotter engines and subjecting the most promising formulations to grueling tests.

For example, a fleet of New York taxi cabs runs on *Uniflo*® motor oil test formulations for 50,000 miles, using oil drain intervals more than twice those recommended. The taxi engines are dismantled before and after each test to measure wear on critical parts in microns, and to examine engine deposits.

MOFT

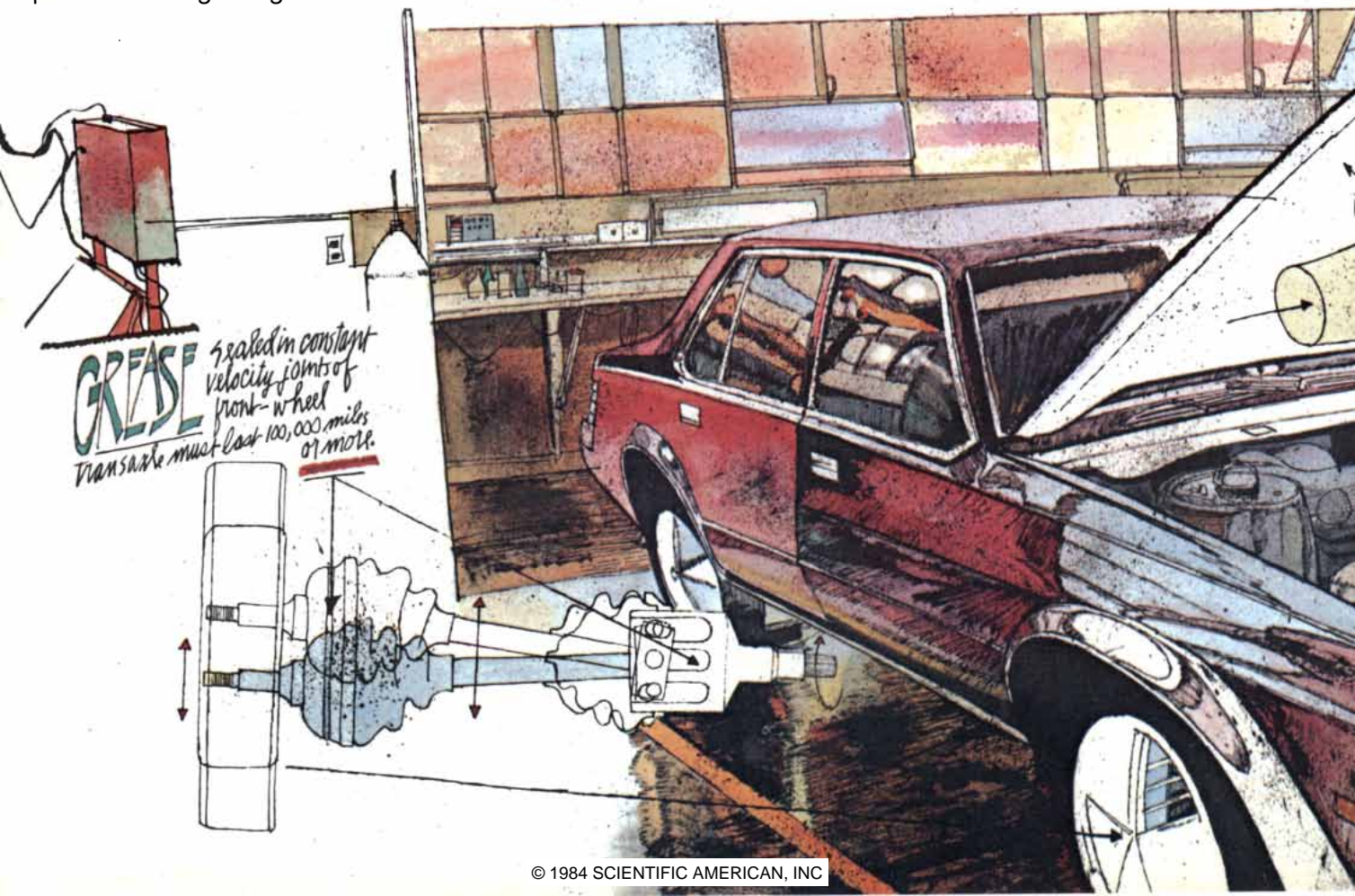
Balancing friction and antiwear properties of an oil is a delicate task. Lower viscosity reduces friction, improves fuel

efficiency and aids startability, but too low viscosity may result in excessive engine wear.

Minimum Oil Film Thickness (MOFT), a patented technique developed at ER&E, electronically measures the protective lubricant film between the bearings and crankshaft of a running engine—sometimes only a fraction of a micron thick. These measurements permit researchers to compare different oil and additive formulations in their search for better wear protection and fuel mileage.

FWD Transaxles

The heat and wear demands of "sealed for life" constant velocity joints in front-wheel transaxles posed other challenges for ER&E scientists in lubricant research. Their response, a lithium-based grease, 5191, can be found in many U.S. front-wheel-drive vehicles,



Keeping ahead of an eye on tomorrow's.

withstanding temperatures up to 300°F and lasting for over 100,000 miles.

Electronic Knock Sensors

Pioneering research at ER&E demonstrated the concept of electronic knock sensors which are now being installed in many of today's cars. These sensors detect engine knock and feed the information to an on-board computer which corrects spark timing to match gasoline octane. This makes higher compression ratios feasible at any given octane level, resulting in more efficient engines.

Changing Fuels for Changing Engines

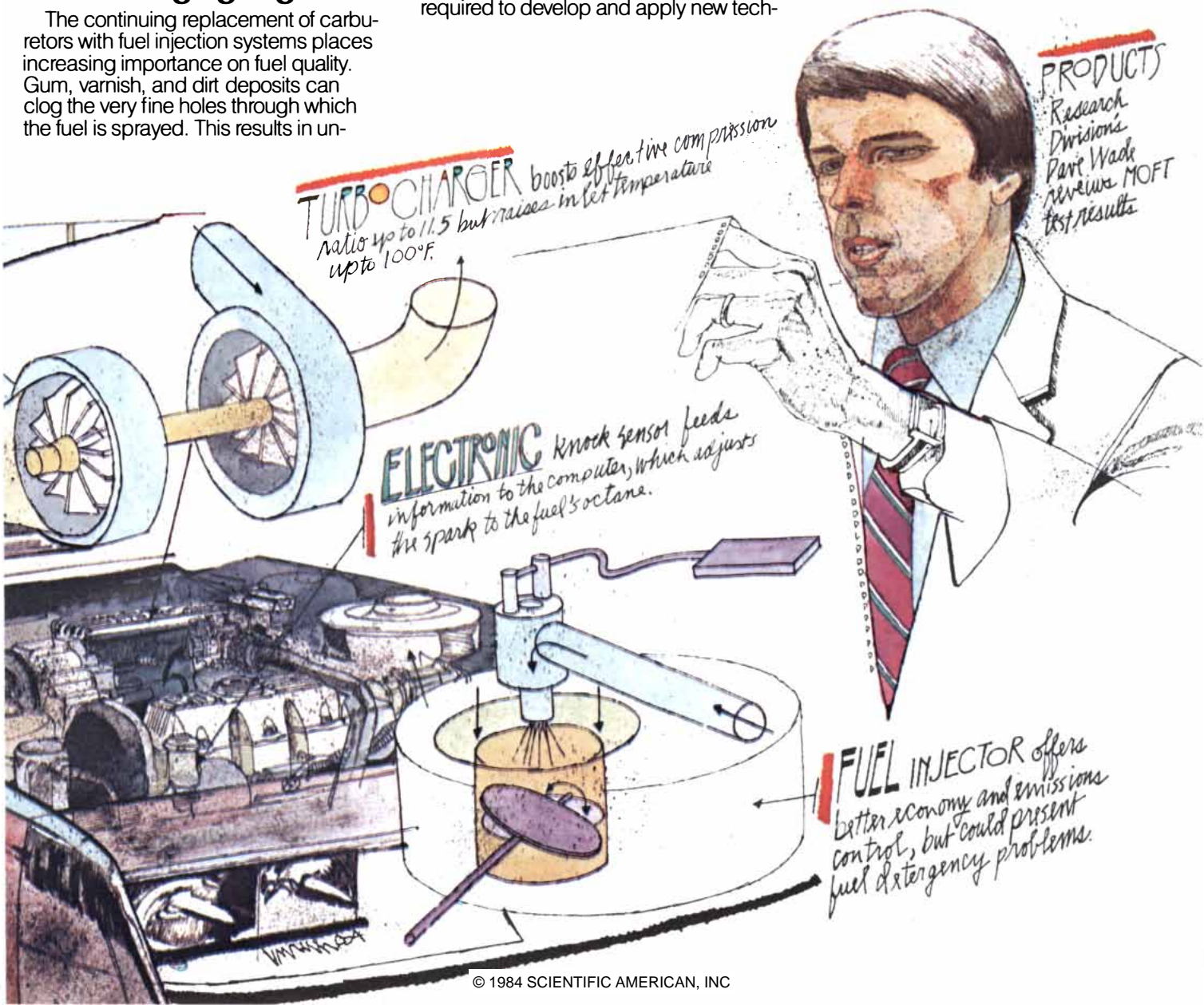
The continuing replacement of carburetors with fuel injection systems places increasing importance on fuel quality. Gum, varnish, and dirt deposits can clog the very fine holes through which the fuel is sprayed. This results in un-

even distribution to some cylinders and reduced engine performance. Ongoing work at ER&E is defining the cleanliness needs of these and future systems and, in parallel, developing fuel quality features to meet those requirements.

Exxon Research and Engineering Company

Improving products for the transportation industry is just one example of the research programs under way at ER&E. A wholly owned subsidiary of Exxon Corporation, ER&E employs more than 2,000 scientists and engineers working on petroleum products and processing, pioneering science and the engineering required to develop and apply new tech-

niques to the manufacture of fuels and other products. For more information on automotive products research or ER&E, write Dr. E. E. David, Jr., President, Exxon Research and Engineering Company, Room 200, P.O. Box 101, Florham Park, New Jersey 07932.



Fellowships in Science Broadcast Journalism

WGBH-Boston, a major producer of radio and television programs for public broadcasting, including NOVA, seeks indications of interest from proven science writers who want to learn the skills and move into science broadcast reporting. We are creating six year-long fellowships for experienced science journalists willing to risk their talents on a major career change.

Fellows will undergo an intensive training program in the art and craft of producing and writing science programs for radio and television, in a working professional environment.

If you have a track record in writing about science in a clear and vital way for a general audience, please send a letter and resume for further information to:

David Kuhn
 Director, Science Fellowships
 WGBH
 125 Western Ave.
 Boston, MA 02134



Aha...



GAMES FOR THINKERS

An exciting, new way to learn creative problem solving! Games designed by university professors improve thinking skills using fun, strategy and challenging competition. Like chess, each game can be played at many levels from young children to intelligent adults. Fascinating for everyone!

Write for free catalog and studies that show how WFF 'N PROOF Games can:

- double math achievement
- cut school absenteeism by 2/3 and
- raise I.Q. scores by 20 points

ORDER YOUR GAMES FOR THINKERS TODAY!

WFF 'N PROOF (logic)	\$16.00
QUERIES 'N THEORIES (sci. method)	16.00
EQUATIONS (creative mathematics)	13.00
ON-SETS (set theory)	13.00
ON-WORDS (word structures)	13.00
PROPAGANDA (social studies)	13.00
CONFIGURATIONS (geometry)	7.75

Complete 7-game Special **79.95**

All prices include postage and handling. Satisfaction Guaranteed

Order from **WFF 'N PROOF**

1490-FE South Blvd., Ann Arbor, MI 48104

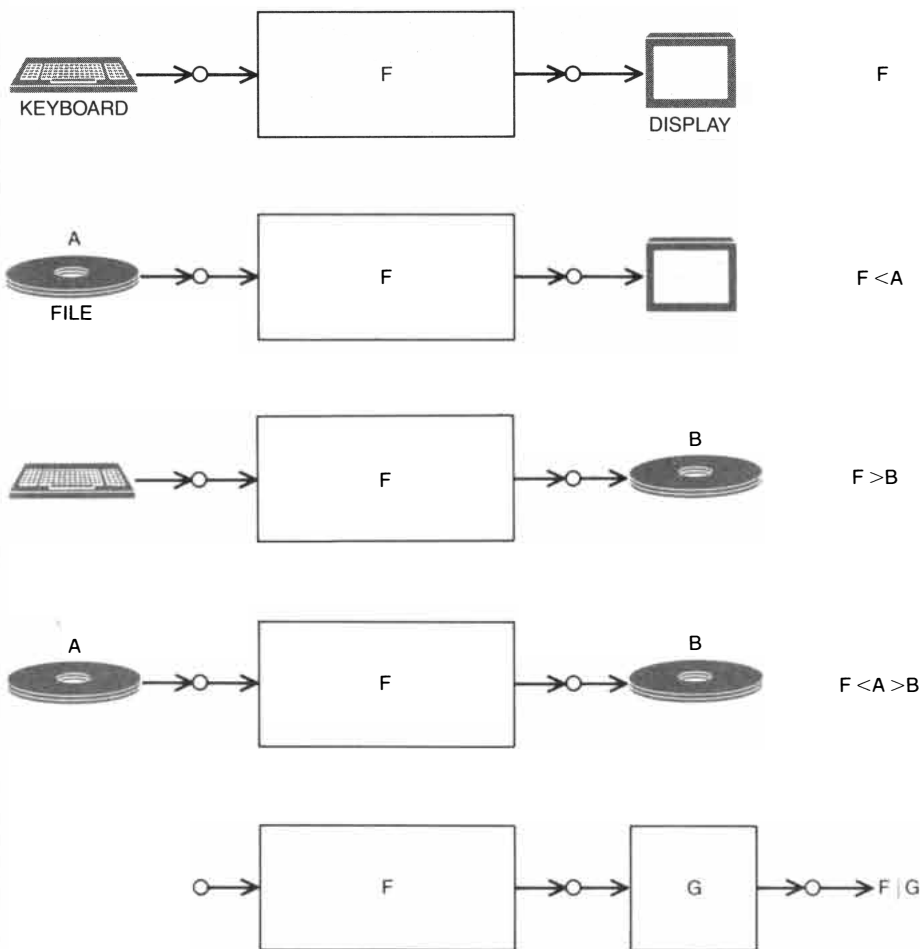
for recording the association between the external and the internal names of objects; other levels manage the objects themselves. Hence when a directory of devices is searched for the string "clock," the result returned to the calling program is merely the internal name of the time-of-day clock. The internal name is then passed to a program at level 10, which does the actual reading of the clock.

Level 12 implements user processes, which are entire virtual machines executing programs. It is important to distinguish the user process from the primitive process of level 5. All the information needed to define a primitive process can be expressed in the state word that records the contents of registers in the central processing unit. A user process includes a primitive process, but it encompasses much else as well: a virtual memory containing the program and its work space, information supplied by the user when the program was started and a list of other objects with which the process can communicate. A user process is much more powerful than a primitive process.

Level 13 is the "shell," so called be-

cause it separates the user from the rest of the operating system. It is the interpreter of a high-level command language, through which the user gives instructions to the system. The shell incorporates the listener program that responds to the terminal keyboard; it parses each line of input to identify program names and other information; it creates and invokes a user process for each program and connects it as needed to pipes, files and devices.

An important principle adopted in the hypothetical operating system we are describing here is input-output independence. At levels 8, 9 and 10 the same fundamental operations (namely *create*, *destroy*, *open*, *close*, *read* and *write*) are defined for pipes, files and devices. Writing a block of data into a disk file calls for a sequence of events quite different from the one needed to transmit the same data to a printer or to supply it to the input of another program, but neither the author nor the user of the program needs to be concerned with those differences. All *read* and *write* statements in the program can refer to input and output "ports." The ports are at-



SOFTWARE PART, or component, is a program with a single input stream and a single output stream, a structure that helps in combining programs and devices in various ways. If no other source and destination are specified, a program is connected by default to the user's keyboard and display. The "<" sign designates a source of input and the ">" sign a destination for output. The "|" sign creates a "pipe" linking the output of one program to the input of another.

Introducing Filevision™ for Macintosh.

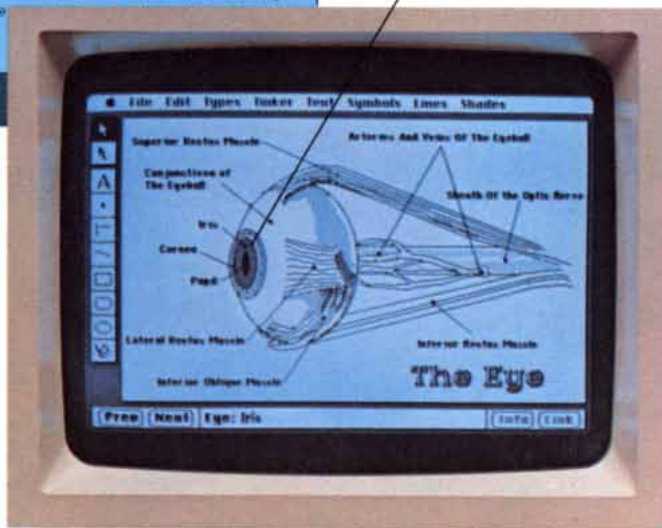
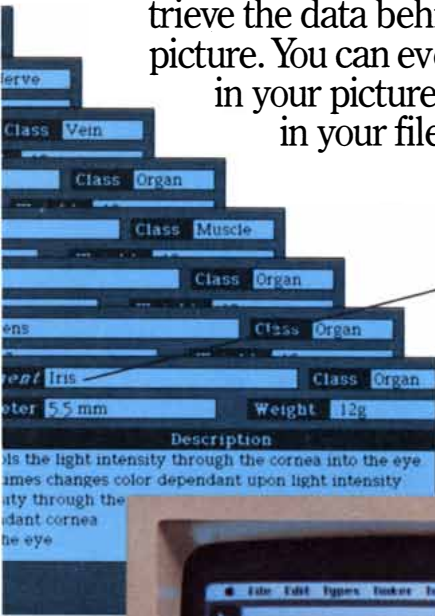
The fine art of filing by pictures.

Filevision. The first software that combines a practical filing system with a simple-to-use, object-oriented drawing system. It lets you file things the way you see them. And quickly visualize your data in pictures. Instead of sorting through tedious line-by-line listings.

To use Filevision, you simply place objects in a picture, or choose from the ready-made symbol menu to represent the pieces of information you wish to file.

In the click of a mouse, you can retrieve the data behind each object in your picture. You can even select the objects in your pictures based on the data in your files.

Each object in your picture is automatically connected to the information about that object.



Illustrate anatomy and study physiology. Map sales territories and track volume. Seat a convention and highlight non-smokers. Filevision pops requests right off the screen. In the click of a mouse.

Filevision is a trademark of Telos Software Products.
Telos is a trademark of Telos Corporation.
Macintosh is a trademark licensed to Apple Computer Inc.

Each object is automatically connected to a data form. Which you custom design, quick as a click.

For a change, it's simple to modify my files.

Updating your files is just as easy. Whenever the best-laid plans of mouse and man

need a little replanning, you're just a click or two away from reperfecting your files.

Create new symbols, and add them to your picture.

Make a data form for any new object, and all objects of that type will have the same form. Automatically.

Modify a symbol, and all matching symbols in your picture will be modified. Automatically.

Change an existing form, and all forms of that type will change. Automatically.

The possibilities are endless.

Filevision can study anatomy while fleshing out the physiology, peek at the population behind a map, show a business person just what's in stock, or lay out an office and see who's got what. In fact, if you can see it, you can file it visually with Filevision. And retrieve it visually, too.

Filevision. The unique filing system for Macintosh that lets you store and work with information in pictures, as well as numbers and text.



TELOS™
SOFTWARE PRODUCTS
Software for the real world.

In the battle between the IBM PC, there can be

Hear the guns?

It's a battle for your desktop. Apple® versus IBM®. The easy-to-use Macintosh against the serious business computer from Big Blue.

And the winner? Epson®. That's right, Epson. Because for the person who simply wants to buy one relatively perfect personal computer, the Epson offers an opportunity for peace in our time.

A computer that is easy to use, like the Mac, but also runs all sorts of business software, like the PC.

And aren't those two computers exactly the one you need?

**An easier way to be easy...
A more serious way to be serious.**

The Epson is easy because its keyboard works in English, not computerese. And only

the Epson comes with *Valdocs*™, a powerful integrated software system that takes you step-by-step through the five most important business functions: word processing, business graphics, telecommunications, electronic filing and daily scheduler.

As a result, while IBM owners are still pondering their manuals, and Macintosh owners are still drawing sneakers, Epson owners are churning out productive work with electronic speed and accuracy.

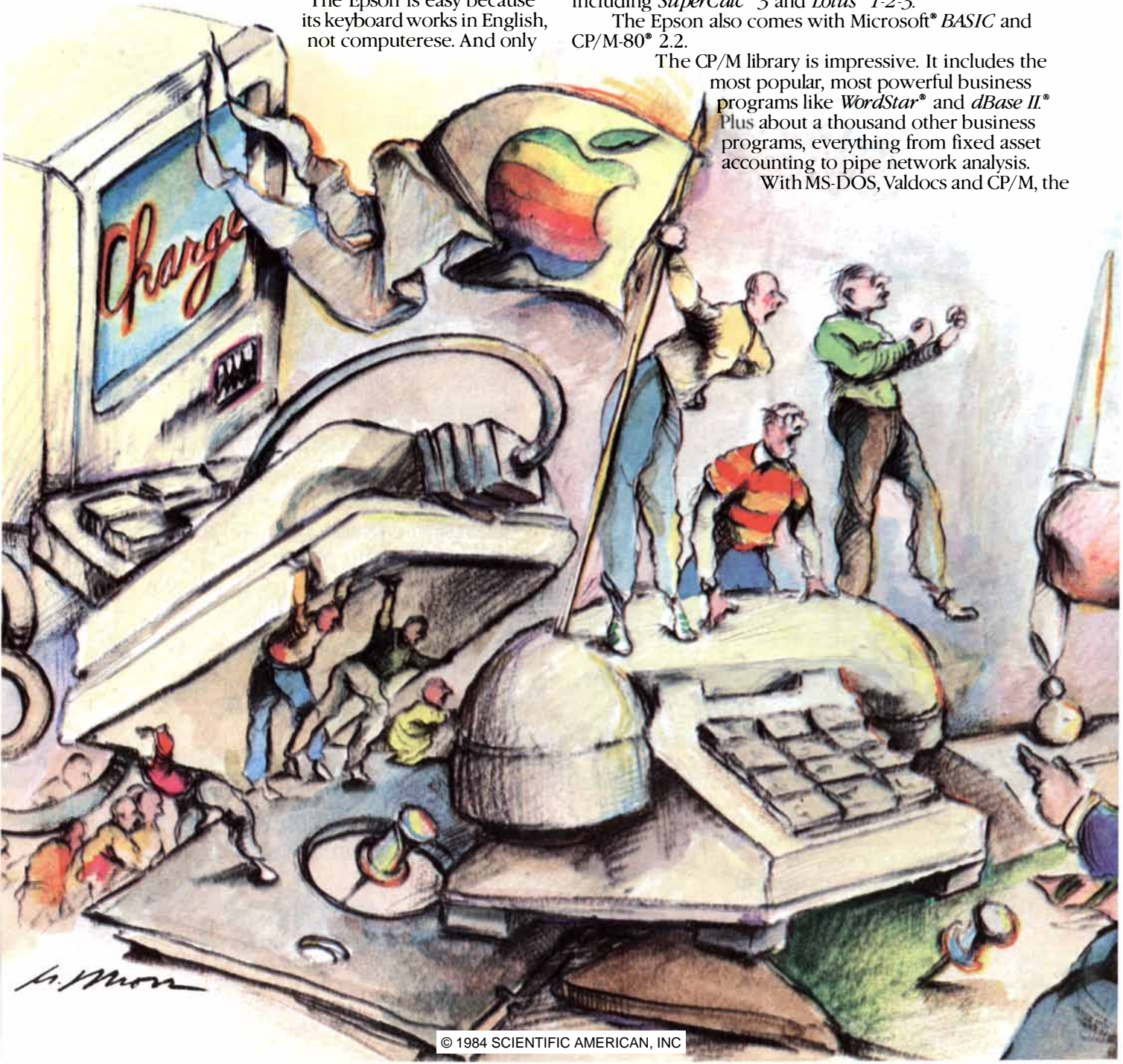
The Epson also opens the doors of your disk drives to the largest collection of software in captivity. In fact, the Epson runs *more* business programs than the IBM PC.*

To start, the Epson is available with an optional 16 bit co-processor so you can use almost any MS™-DOS program, including *SuperCalc*® 3 and *Lotus*® 1-2-3.®

The Epson also comes with Microsoft® *BASIC* and CP/M-80® 2.2.

The CP/M library is impressive. It includes the most popular, most powerful business programs like *WordStar*® and *dBase II*®. Plus about a thousand other business programs, everything from fixed asset accounting to pipe network analysis.

With MS-DOS, Valdocs and CP/M, the



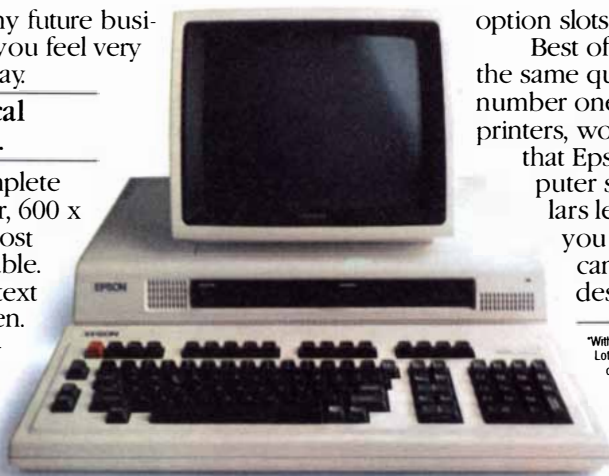
the Apple Macintosh and only one winner. The Epson.

Epson should be able to handle any future business need. And that should make you feel very good about siding with Epson today.

The ultimate technical specification: value.

The Epson QX-10 comes complete with a 12" high resolution monitor, 600 x 400 Pixels, driven by one of the most powerful graphic processors available. With screen resolution this good, text and graphics will leap off the screen. And when you add a graphics program, like *Q-plotter*,™ you can produce presentation graphics of the highest order.

Standard issue on the Epson also includes 256K memory, plus 128K resident video memory, dual 380K Epson-made disk drives, a CMOS Realtime Clock/Calendar with battery backup, a 1-year warranty, an RS-232C port and a parallel port; thus freeing the five—that's right, five—



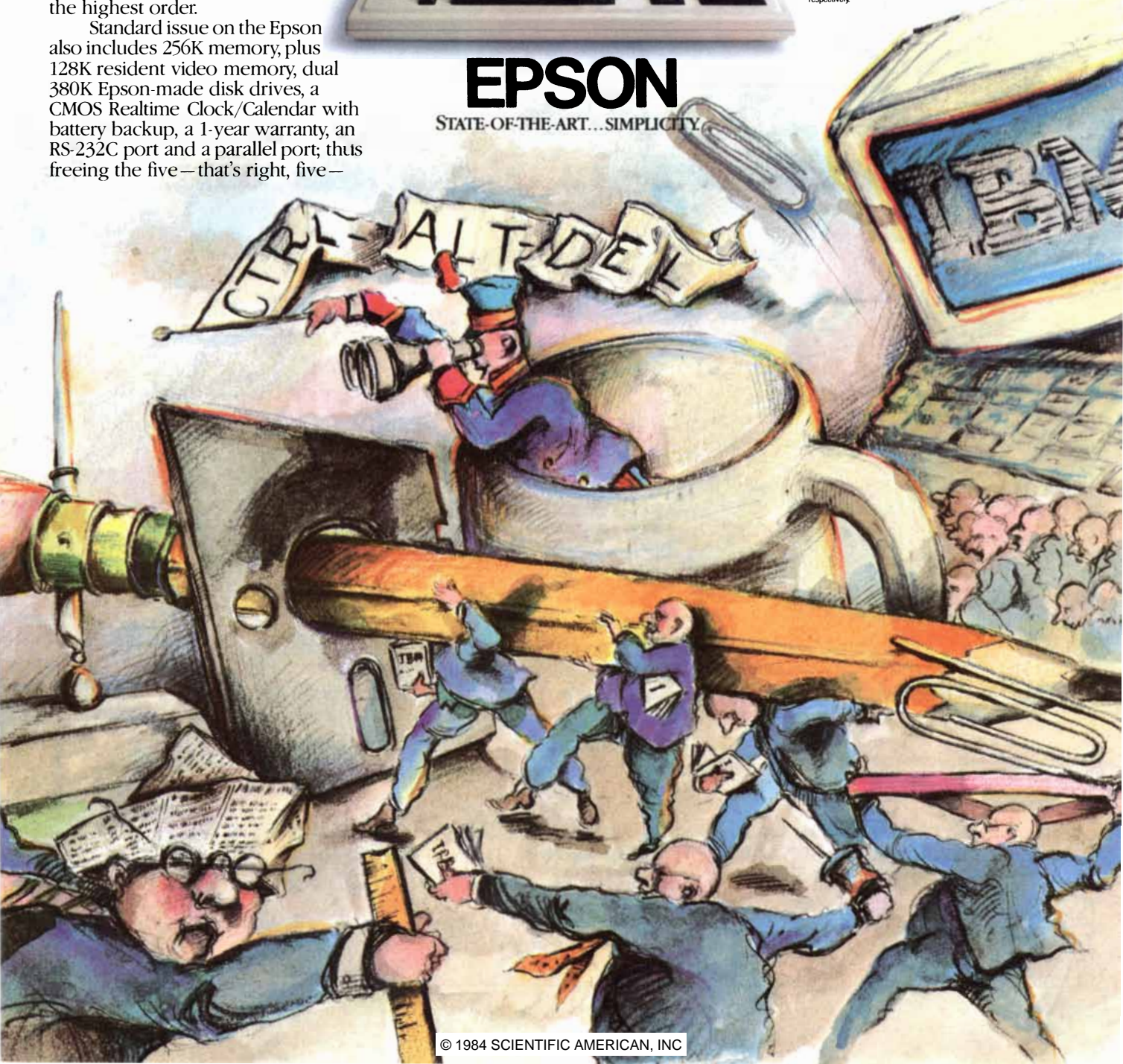
option slots for some real options.

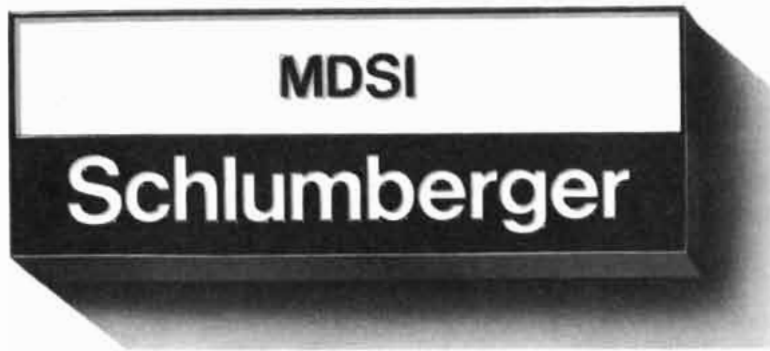
Best of all, everything is Epson quality, the same quality that has made Epson the number one manufacturer of computer printers, worldwide. And when you consider that Epson gives you a complete computer system at a price a thousand dollars less than either Apple or IBM, you understand why this computer can not only bring peace to your desktop, but to your budget, as well.

*With optional MS-DOS board. Apple, the Apple logo, IBM, Epson, SuperCalc 3, Lotus 1-2-3, CP/M-80, Microsoft, WordStar and dBase II are registered trademarks of Apple Computers, Inc., IBM, Epson Corp., Sorcim, Lotus, Digital Research, Microsoft, Micropro and Ashton-Tate respectively. ValTools, MS, Q-plotter are trademarks of Rising Star, Microsoft and MetroSoftware respectively.

EPSON

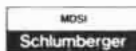
STATE-OF-THE-ART...SIMPLICITY.





**Memorize this symbol.
It might help keep you from making
a million dollar mistake
in Computer-Integrated Manufacturing.**

If you want to find out more about the intelligent approach to CIM,
write us at our North American Headquarters.
Or see us at Booth #2646 at the International Machine Tool Show September 5-13.



MANUFACTURING DATA SYSTEMS INCORPORATED

North American Headquarters

4251 Plymouth Road, P.O. Box 986
Ann Arbor, Michigan 48106
U.S.A.
Telephone: 313-995-6000
TWX: 810/2236039

European Headquarters

Hahnstrasse 70
Lyoner Stern
D-6000 Frankfurt/Main 71
West Germany
Telephone: 49.611.66.41.420
Telex: 841/413591

Asian Headquarters

Koito Building 8F
12-4 Nishi-Shinjuku, 6-Chome
Shinjuku-ku, Tokyo 160
Japan
Telephone: 81.3.348.45.01
Telex: 781/02324413

Software development... another good reason to think of Westico.

tached to particular files, pipes or devices only when the program is executed.

This strategy, called delayed binding, can greatly increase the versatility of a program. A sorting routine, for example, could take its input either from a file or directly from a terminal and could send its output to another file, to a terminal or to a printer. Without delayed binding a separate routine might be needed for each possible combination of source and destination.

Another principle observed in the construction of the operating system has the enigmatic name "information hiding." Each level builds on the levels below and hides all the internal details of its operations from the levels above. For example, the primitive-process manager at level 5 creates the illusion that all the primitive processes on the ready list are executing in parallel; the details of queuing, interrupts and so on are invisible to higher levels. A program that makes use of primitive processes deals with only a small set of external commands for creating and destroying processes, suspending and resuming their execution and sending and receiving messages. Similarly, the user-process manager at level 12 gives the illusion that each program operates in its own machine; the creation of the primitive process, the work space and connections to input and output ports are hidden.

Just as no level has access to the internal workings of lower levels, no level depends on assumptions about higher levels. The virtual-memory manager (level 7) must have access to the interrupt system (level 4) and the secondary-storage system (level 6), but it knows nothing of the file structure (level 9). The stratification of the operating system aids in its construction because the levels can be installed and tested one at a time from the bottom up.

The description of an operating system given so far is static: the parts have been listed, but how they work has not been demonstrated. The state of the system changes as commands are executed. The following examples of system dynamics are based on the Unix operating system, which incorporates many features of the hypothetical system discussed above.

The user sees a computer as a large system with many useful resources. Some of the resources are programs stored as binary code that can be executed merely by giving a file name. In a Unix system this category might include the date program, compilers for high-level languages, and programs for preparing documents, including formatting programs for tables, equations and ordinary text. Other elements of the system are data files, perhaps holding documents of various kinds, including the "source code" of programs. Hardware

Westico is *the* place to get all the latest Digital Research programs. We carry a complete line of Digital Research language processors, development tools and operating systems, including Concurrent DOS™ with windows. In fact, we're the only source for their CB-68K™ Compiler and GSX-86™ Programmer's Toolkit.

And with Westico, you get more than just the technical expertise built into every Digital Research

program. You get all the support you need. Fast, reliable delivery. Expert technical assistance. Plus, outstanding dealer discounts to keep every independent software vendor smiling. Westico offers hundreds of other software programs in formats for over a hundred personal computers.

So when you're thinking development software, think Westico. For a Westico software catalog and complete information, call or write today.



WESTICO

The Software Express Service™

25 Van Zant Street, Norwalk, CT 06855
Telephone (203) 853-6880. Telex 64-3788.

Access Manager, CBASIC Compiler, CBASIC-86, CB-80, CB-86, CP/M-68K, CB-68K Compiler, Concurrent DOS, Concurrent PC DOS, Display Manager, Digital Research C, Dr. Logo, DR Graph, DR Draw, FORTRAN 77, GSX-80, GSX-68K, GSX-86, Pascal/MT+, Pascal/MT+ 86, Pascal/MT+ 68K, Personal BASIC, PL/I-80, PL/I-86, DR Assembler
Tools are trademarks of Digital Research Inc.

devices such as terminals, the clock and printers are also accessible.

The directories listing all these resources are arranged in an inverted tree, with the highest-level directory at the root. Some directories are reserved for the use of the system; they might include directories of devices, of commands and of files holding miscellaneous data such as the passwords of authorized users. A directory labeled "user" has a subdirectory for each person who has an account on the system, and each user in turn can create a tree of further subdirectories in which to store all the files, pipes and devices he has created.

Most users of the system spend most of their time employing existing programs, not writing new ones. The design of the operating system, and in particular the principle of input-output independence, encourages this style of computing. Most of the programs in the system libraries are "software parts" that can work interchangeably with various sources of input and destinations for output.

When a user "logs in," generally by giving a password, the operating system creates a user process that includes a copy of the shell program. The input to the shell is connected to the keyboard of the user's terminal and the output to the same terminal's display. The shell "listens," without taking any action, until a full line of input has been typed, signaled by the receipt of the carriage-return character. The line is then scanned to pick out the names of programs being invoked and the values to be passed as arguments to those programs. For each program called up in this way the shell creates a user process, including a copy of the executable code for the program and a work space. The processes

specified in the command line.

Operations of substantial complexity can be specified in the command language of the Unix shell. For example, a sequence of operations that formats a file named "text" could be set in motion by the command line

```
tbl <text|eqn|lptroff >output .
```

Here the first program invoked is *tbl*, whose function is to search a file for descriptions of tables of information and insert the formatting commands. The "<" symbol indicates that *tbl* is to take its input from the file "text." The output of *tbl* is directed by a pipe (the "|" symbol) to the input of *eqn*, which supplies formatting commands for any descriptions of equations. The output of *eqn* is then piped to *lptroff*, another formatting program that prepares the rest of the text for printing; the name of the program is shorthand for "laser-printer typesetter runoff." Finally, the ">" symbol indicates that the formatted document is written in a file named "output." It is ready to be sent to a laser printer, a high-quality printer that works much like a photocopying machine.

If the formatting and printing of documents is to be done often, typing such an elaborate command would soon become tedious. Unix encourages the user to store complicated commands in executable files called shell scripts that become simpler commands. A file named *lp* might be created with the contents

```
tbl <$1|eqn|lptroff >$2 .
```

Here the names of the input and output files have been replaced by the variables \$1 and \$2. When the command *lp* is in-

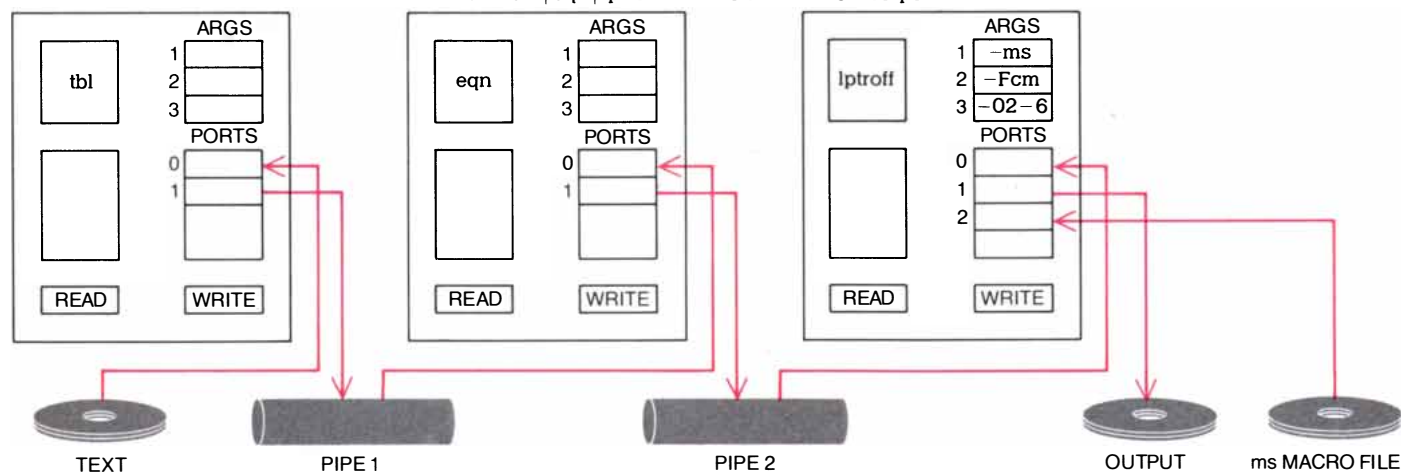
lp text output

would substitute "text" for \$1 and "output" for \$2 and so would have exactly the same effect as the longer command given above.

One more small but essential piece of the operating system needs to be discussed. Given that various components of the operating system have responsibility for loading all programs into the machine, the question naturally arises of how the operating system itself is loaded and started. The answer is a "bootstrap sequence." The sequence begins with a program of just two instructions permanently inscribed in read-only memory and hence present even when the power is off. This small program initiates the loading of a somewhat larger program from disk, which then takes control and loads the operating system itself.

The hierarchical principle we have applied here to the organization of an operating system is one that has proved to be of great utility throughout the natural sciences. After all, structures and events in the natural world span many orders of magnitude in space and time and cannot be grasped all at once: it is not possible to comprehend the evolution of a galaxy by plotting the trajectories of its constituent atoms. Of all man-made objects computer systems have the greatest disparity between the smallest and the largest components. The designers of operating systems have begun to cope with that vast range of scales by creating a hierarchy of abstractions.

```
tbl <text|eqn|lptroff -ms -Fcm -o2-6 >output
```

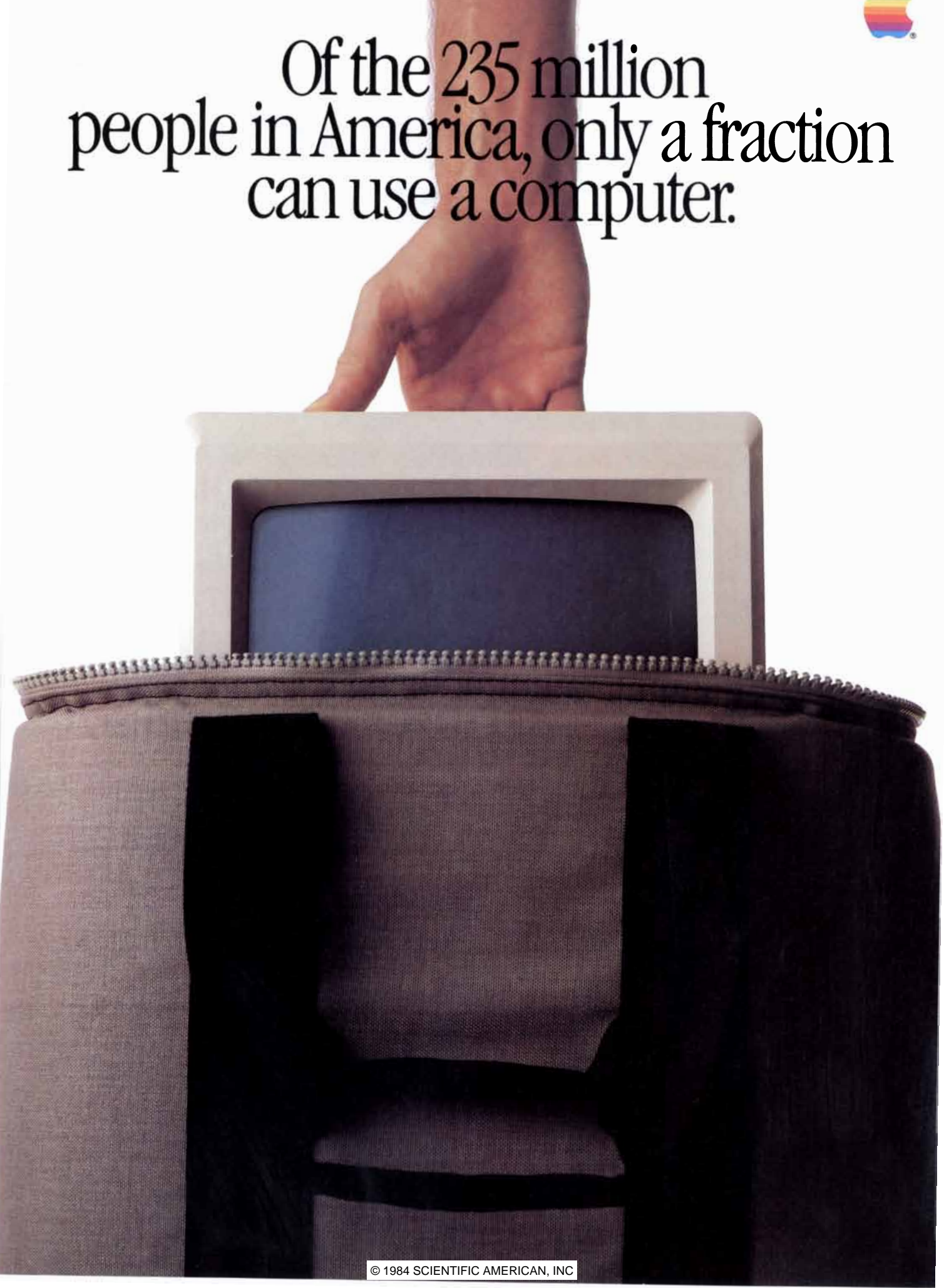


PROCESS PIPELINE brings together three programs, two pipes and three files to prepare a text for printing. The first program, *tbl*, takes its input from the file named "text" and inserts formatting commands for tabular matter. The output of *tbl* is piped to *eqn*, which does similar formatting of equations. The second pipe carries the text to *lptroff*, which completes the formatting; the name of the program stands for

"laser-printer typesetter runoff." Three options to *lptroff* are given as arguments in the command line: *-ms* instructs the program to open a "macro" file called "ms" in order to expand abbreviated format codes found in the text, *-Fcm* specifies a type font named Computer Modern and *-o2-6* indicates that only pages 2 through 6 of the output are to be generated. The output is directed to a file for later printing.



Of the 235 million
people in America, only a fraction
can use a computer.



Now there's Macintosh. For the rest of us.

In the olden days, before 1984, not very many people used computers. For a very good reason.



Some particularly bright engineers.

Not very many people knew how. And not very many people wanted to learn.

After all, in those days, it meant listening to your stomach growl through computer seminars. Falling asleep over computer manuals. And staying awake nights to memorize commands so

complicated you'd have to be a computer to understand them.

Then, on a particularly bright day in Cupertino, California, some particularly bright engineers had a particularly bright idea: since computers are so smart, wouldn't it make more sense to teach computers about people, instead of teaching people about computers?

So it was that those very engineers worked long days and late nights and a few legal holidays, teaching tiny silicon chips all about people. How they make mistakes and change their minds. How they refer to file folders and save old phone numbers. How they labor for their livelihoods, and doodle in their spare time.

For the first time in recorded computer history, hardware engineers

actually talked to software engineers in moderate tones of voice, and both were united by a common goal: to build the most powerful, most portable, most flexible, most versatile computer not-very-much-money could buy.

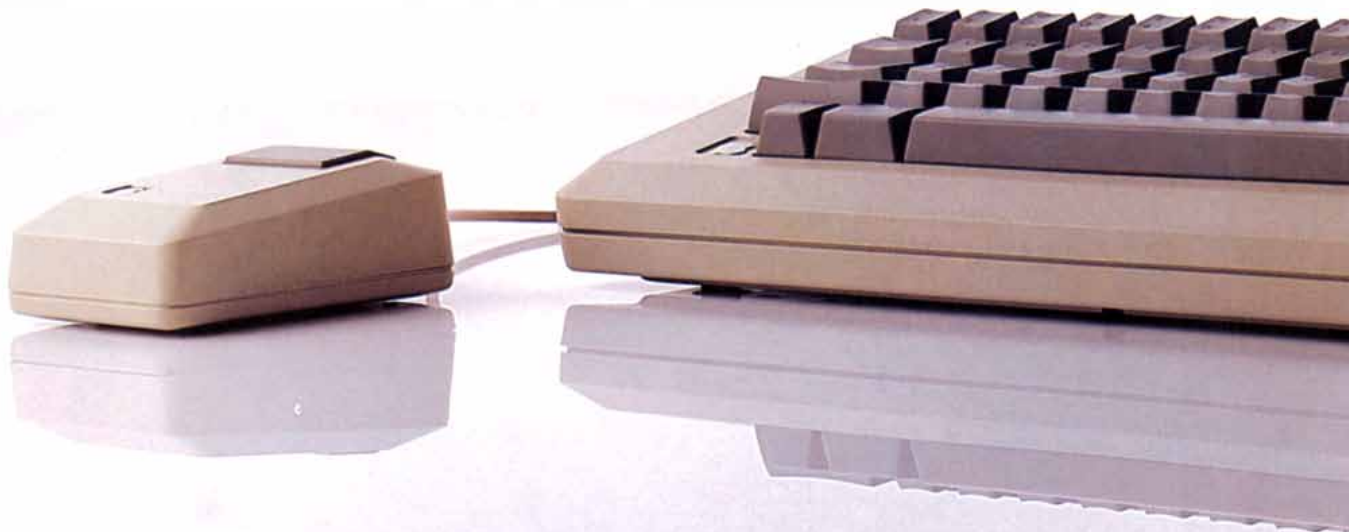
And when the engineers were finally finished, they showed us a personal computer so personable, it can practically shake hands.

And so easy to use, most people already know how.

They didn't call it the QZ190, or the Zipchip 5000.

They called it Macintosh.™

And now we'd like to show it to you.

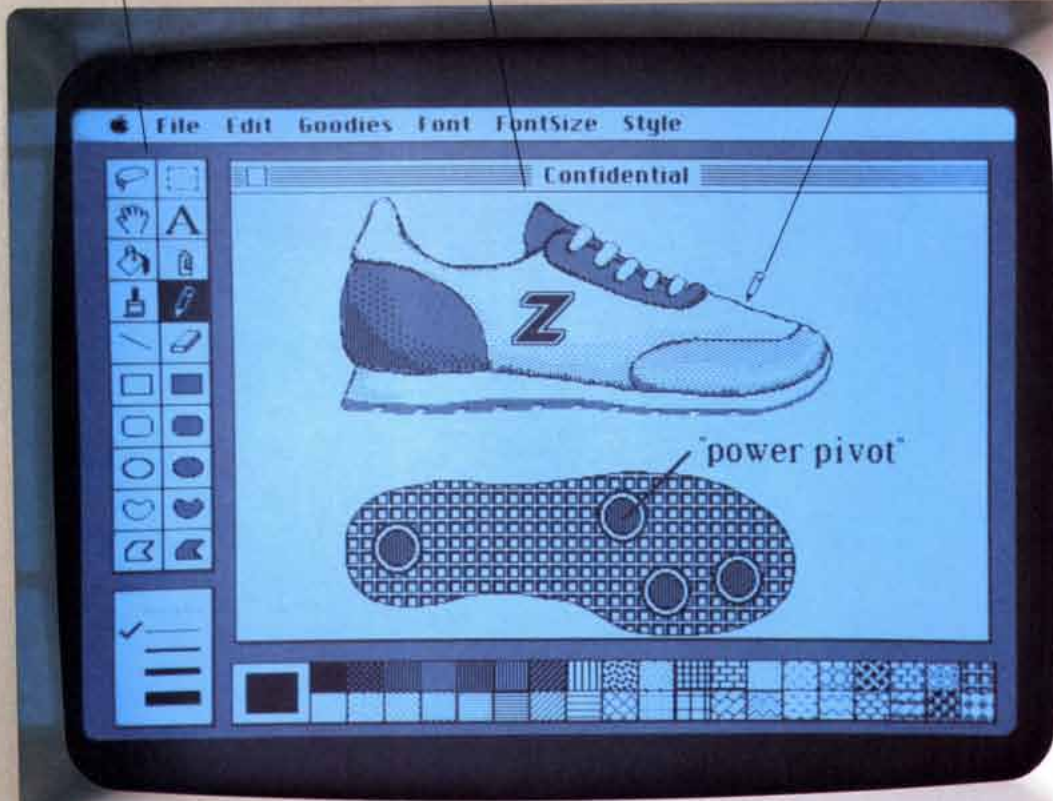




3 palettes display available tools, line widths, and patterns.

You're not limited to the work area you see here. You can scroll up and down, left and right.

The pointer becomes whatever tool you select to work with—in this case, a pencil.



Point. Click.

To tell Macintosh what you want to do, all you have to do is point and click.

You move the pointer on the screen by moving the mouse on your desktop. When you get to the item you want to use—click once, and you've selected that item to work with.

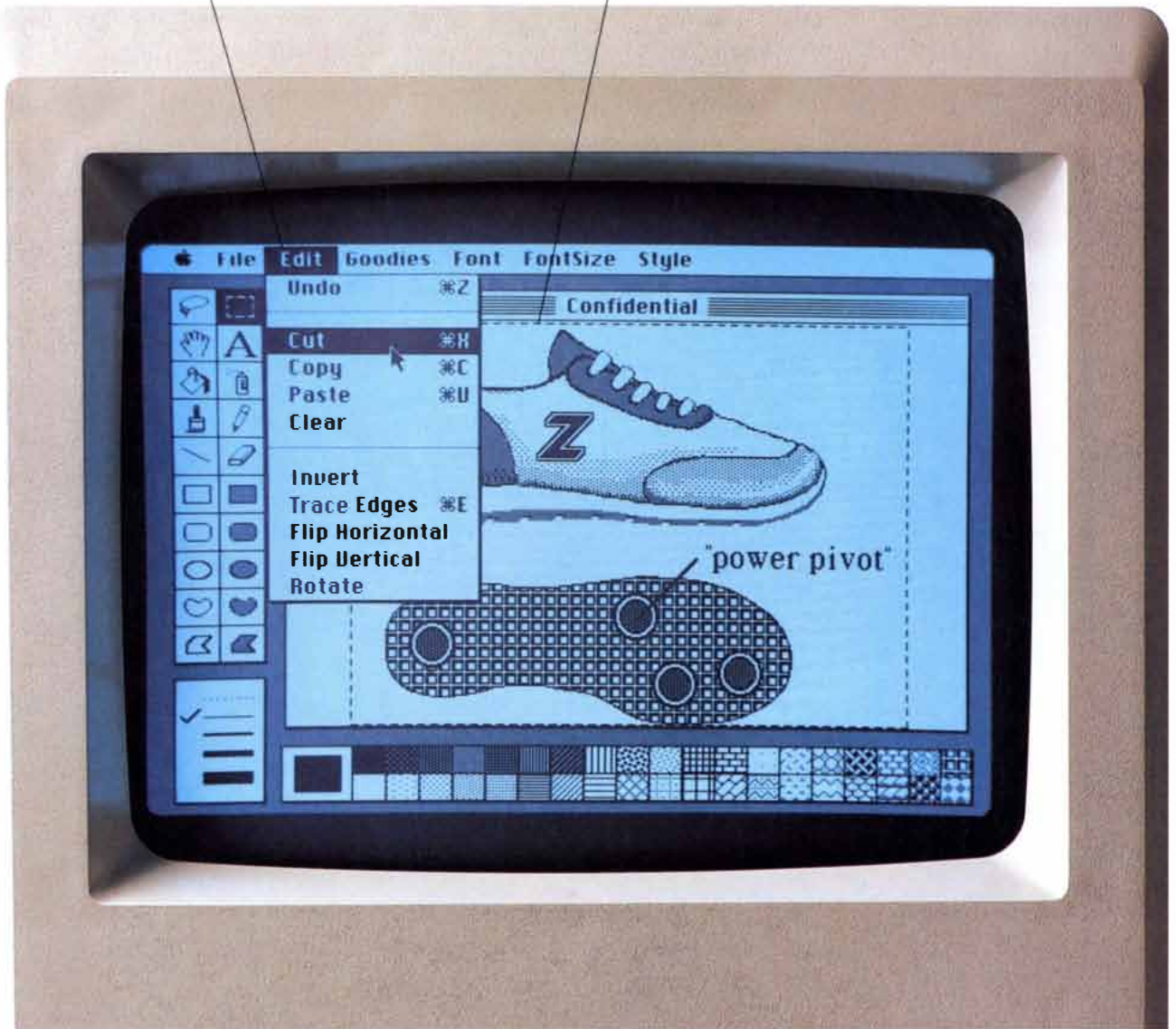
In this case, the pointer appears as the pencil you've selected to put some finishing touches on an illustration you'd like to include in a memo.



"Pull-down menu" displays all your options.

To select whatever you want "cut" from the screen, just put a rectangle around it.

Macintosh stores the image you've "cut" out on a "clipboard" in its memory.



Cut.

Once you've completed your illustration, you need to cut it out of the document you created it on, so that you can put it into the word processing program you used to write your memo.

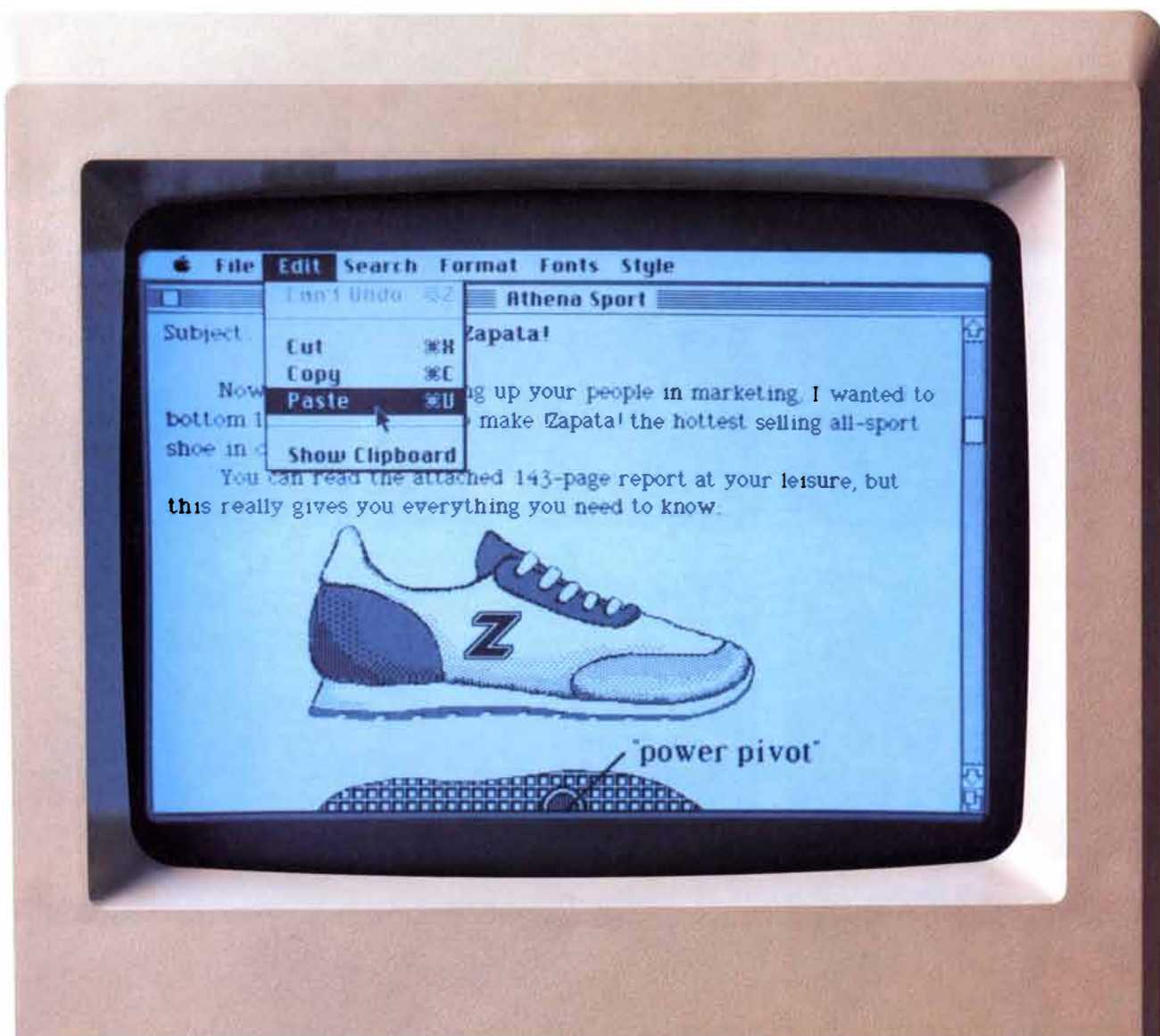
To do this, you simply use the mouse to draw a box around the illustration, which tells Macintosh this is the area you want to cut.

Then you move the pointer to the top of the screen where it says "Edit." Hold the mouse button down and Edit will then show you a list, or "pull-down menu" of all the editorial commands available. Then pull the pointer down this menu and point to the command, "Cut," highlighted by a black bar.

Release the mouse button and, zap, it's done.



*Macintosh automatically makes room
for your illustration in the text.*



Paste.

And now, to finish your memo, bring up MacWrite™, Macintosh's word processing program. Just pick a place for your illustration.

In the meantime, your illustration has been conveniently stored in another part of Macintosh's ample memory.

To paste the illustration into your memo, move the mouse pointer once again to the Edit menu at the top of the screen.

This time, you pull the mouse down until "Paste" is highlighted by a black bar. Release the mouse button and, once again, zap.



With Macintosh, you can print out your own office forms or stationery in addition to whatever you print on them.

Athena/Sport MEMO
CONFIDENTIAL

From: Nancy Aronson
To: Chris Jordan
Subject: Launching iZapata!

Now that you're gearing up your people in marketing, I wanted to bottom line what's going to make iZapata! the hottest selling all-sport shoe in our line.

You can read the attached 143-page report at your leisure, but this really gives you everything you need to know:

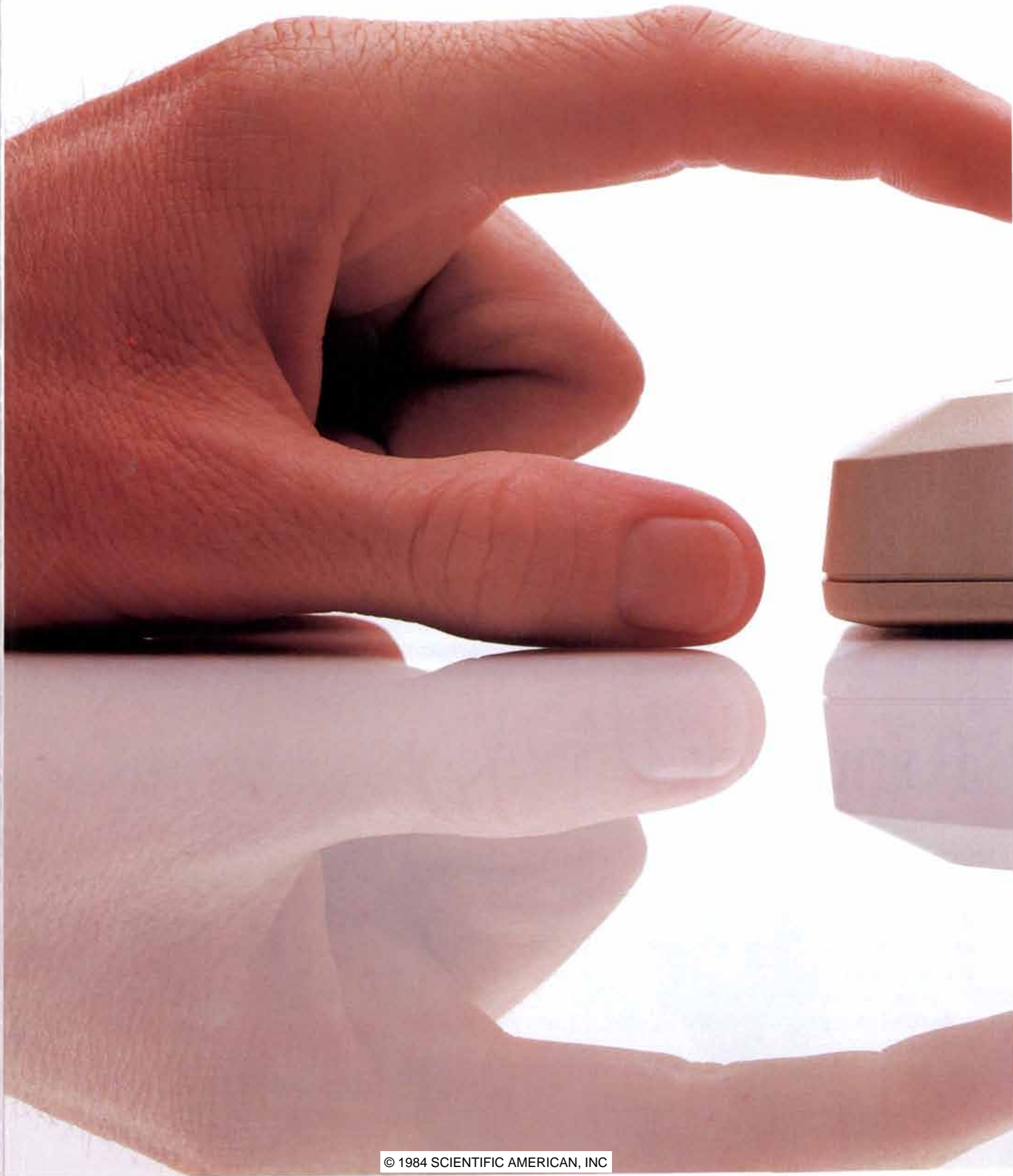


New styling, yes. But the 'power pivots' on the sole tell all -- they may not look like much, but it took about a megabuck in research to put them where they are. They make iZapata! the first all-sport sneaker of the '80s. Look forward to seeing the real thing you buy once up with.

And print.

You tell a Macintosh Personal Computer to print the same way you tell it to do everything else—move the mouse pointer to "File" and pull it down until "Print" is highlighted in a black bar. And, provided you have a printer, you'll immediately see your work in print.

Your work, all your work, and nothing but your work. Because with Macintosh's companion printer, Image-writer, you can print out everything you can put on a Macintosh's screen.



If you can point, you can use a Macintosh.

You do it at baseball games. At the counter in grocery stores. And every time you let your fingers do the walking.

By now, you should be pretty good at pointing.

And having mastered the oldest known method of making yourself understood, you've also mastered using the most sophisticated personal computer yet developed.

Macintosh. Designed on the simple

premise that a computer is a lot more useful if it's easy to use.

So, first of all, we made the screen

layout resemble a desktop, displaying pictures of objects you'll have no trouble recognizing. File folders. Clipboards. Even a trash can.

Then, we developed a natural way for you to pick up, hold,

and move these objects around.

We put a pointer on the screen,



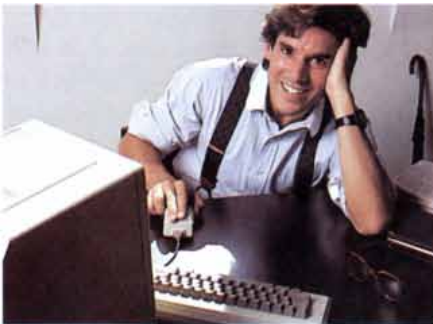
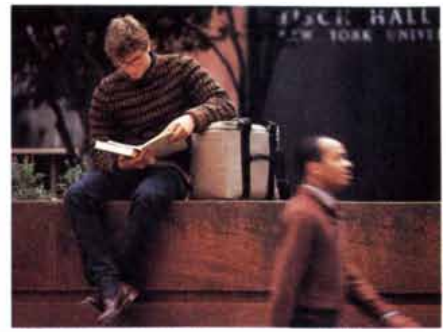
and attached the pointer to a small, rolling box called a "mouse." The mouse fits in your hand, and as you move the mouse around your desktop, you move the pointer on the screen.

To tell a Macintosh Personal Computer what you want to do, you simply move the mouse until you're pointing to the object or function you want. Then click the button on top of the mouse, and you instantly begin working with that object. Open a file folder. Review the papers inside. Read a

memo. Use a calculator. And so on.

And whether you're working with numbers, words or even pictures, Macintosh works the same basic way. In other words, once you've learned to use one Macintosh program, you've learned to use them all.

If Macintosh seems extraordinarily simple, it's probably because conventional computers are extraordinarily complicated.



If you have a desk, you need a Macintosh.

Macintosh was designed for anyone who handles, collects, distributes, interprets, organizes, files, comprehends, generates, duplicates, or otherwise futzes with information.

Any information. Whether it's words, numbers or pictures.

We've narrowed it down to anyone who sits at a desk.

If, for example, your desk is in a

dormitory, Macintosh isn't just a tool, but a learning tool. For doing everything from problem sets in Astrophysics 538 to term papers in Art Appreciation 101. Not to mention perfecting skills in programming languages like Macintosh BASIC and Macintosh Pascal. Which explains why colleges and universities across the country are ordering Macintoshes by the campus-full.

If you own your own business, owning your own Macintosh Personal Computer could mean the difference between getting home before dark, and getting home before Christmas. With software programs like MacWrite, MacProject, MacTerminal, MacDraw, MacPaint, data base managers, business graphics programs and other personal productivity tools available from leading software developers, you can spend more time running your business, and less time chasing after it.

And even if you work for a company big enough to have its own mainframe or minicomputer, Macintosh can fit right in. It's fluent in DEC VT100, VT52 and plain old TTY. With additional hardware, it can talk to IBM mainframes in their very own 3278 protocols.

If your company has a subsidiary abroad, your colleagues there can use all the same tools. Because Macintosh will be available in international versions with local conventions (alphabets, currencies, dates, etc.).

In other words, wherever there's a desk, there's a need for a Macintosh.

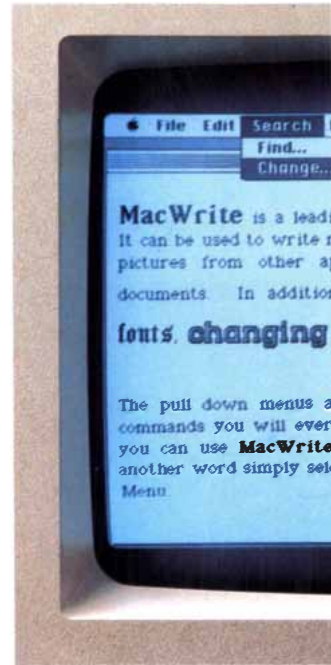
And the less you can see of your desktop, the more you could use one.



An ordinary personal computer makes Macintosh even easier to understand.



Word processing before Macintosh.



In 1977, Apple set the first standard for the personal computer industry with the first generation Apple II.

In 1981, IBM set the second standard with their PC.

And in 1984, Macintosh will set the third industry standard, redefining the term "personal computer."

To give you an idea just how far the technology has advanced over the past three years, we're going to compare, screen-to-screen, the way IBM's PC and Macintosh perform five typical personal computer functions.

Take word processing, for example.

Any computer worth its weight in silicon does an adequate job of shuffling words. Provided, of course, you know all the keystroke "command sequences" to make it happen. And the IBM PC is

no exception.

Macintosh, on the other hand, is quite an exception.

Using Macintosh's word processing program, MacWrite, anything and everything you might want to do with words can be done with a point-and-click of the mouse.

MacWrite not only shuffles words, it can shuffle them in many different type styles and sizes (not to mention boldface, italics and underlining). So you can create documents that look like they came from a typesetter, not a computer. For your foreign correspondence or scientific documents, the Macintosh keyboard gives you 217 characters including accented letters and mathematical symbols.

But what really separates Macintosh

from the blue suits is its extraordinary ability to mix text with graphics. You can actually illustrate your words, memos and letters with tables, charts and free-hand illustrations composed on other graphics programs. All by cutting and pasting with the mouse.

That capability alone makes Macintosh its very own form of communication. A new medium that allows you to supplement the power of the written word with the clarity of illustrations. In other words, if you can't make your point with a Macintosh, you may not have a point to make.

Actually, the difference between Macintosh and the IBM PC becomes obvious the minute you turn both of them on.

The two screens top right show you

precisely how each of them greets you. Notice the IBM presents you with a laundry list of files available for accessing. And multiple steps are required to "get at" the particular file you choose to work with.

Macintosh, on the other hand, shows you everything you've saved (charts, graphs, illustrations and documents), pretty much the same way you'd see them arranged on your desk. Choose one with the mouse, click, and you're ready to work.

Even comparing a program as

the additional cost to add the color card and separate color monitor required to make use of them.

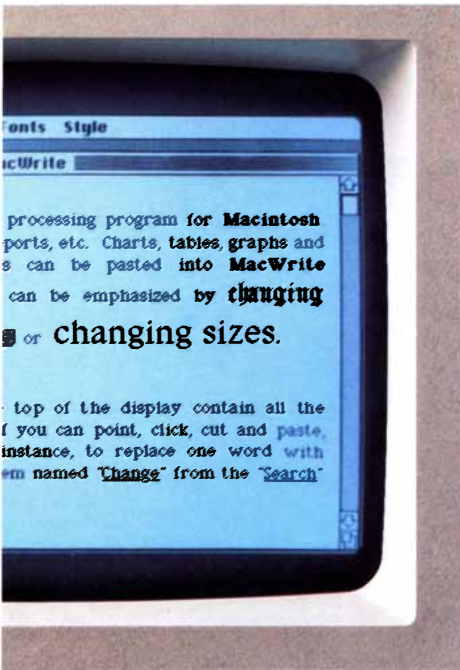
When you compare the actual unit you purchase initially with our Macintosh, the IBM PC not only comes up short a few bar and pie charts, it draws a complete blank.

Macintosh uses its graphics program, Microsoft's Chart, to turn numbers nobody understands into charts and graphs that everybody understands. With it, you can "cut" numbers you want charted from another Macintosh program and

paste them directly into Chart. Just choose the style of chart you want from a "pull-down" selection of pie and bar charts, line and scatter graphs. Then customize your graph with legends and labels in whatever type style your little chart requires.

There is one thing that the IBM PC manages to do as well as Macintosh: IBM 3278 terminal emulation, so you can communicate with heftier IBM's.

But with MacTerminal software, your Macintosh can also fully emulate all the popular DEC terminals.



MacWrite

commonplace as the electronic spreadsheet clearly shows you that Macintosh is anything but commonplace.

Microsoft's® Multiplan™ for Macintosh has been designed to take full advantage of Macintosh's built-in Lisa Technology—clumsy cursor keys are replaced by a point-and-click of the mouse.

Let's say you want to change the width of a column in your spreadsheet. On the IBM PC, that's a 4-key command sequence. On Macintosh, you simply move the pointer and click.

Should you need to make a few quick computations before entering new spreadsheet figures, you can use the built-in desk calculator, for example.

When it comes to business graphics, in all fairness, IBM has color and bar charts to spare. Provided you can spare



File listings before Macintosh.



Macintosh's Finder.



Spreadsheets before Macintosh.



Microsoft's Multiplan for Macintosh.



Business graphics before Macintosh.



Microsoft's Chart for Macintosh.



Terminal emulation before Macintosh.



MacTerminal.

Comparisons made using standard configuration Macintosh and IBM PC (5150 2-disk unit, 256K bytes RAM, 5151 monitor). November 5, 1983.

And here's where ordinary personal computers draw a blank.

You've just seen some of the logic, the technology, the engineering genius and the software wizardry that separates Macintosh from conventional computers.

virtually any image the human hand can create. Because the mouse allows the human hand to create it.

MacPaint gives you total freedom

able by enlarging MacPaint illustrations or making transparencies for overhead projection. Or clarify a memo or report by "cutting out" your illustration and "pasting" it into your text.

What MacPaint does for helping you visualize your wildest imaginings, MacProject does for helping you visualize the unforeseen.

You simply enter all the tasks and resources involved in a project—whether it's opening a new office or producing a brochure—and MacProject will chart the "critical path" to completion, calculating dates and deadlines. If there's a single change in any phase of the project, it will automatically recalculate every phase.

So with MacProject, you can generate business plans and status reports that reflect the realities



MacPaint produces virtually any image the human hand can create.

Now, we'd like to show you some of the magic.

First, there's MacPaint. A program that transforms Macintosh into a combination architect's drafting table, artist's easel and illustrator's sketch pad.

With MacPaint, for the first time, a personal computer can produce

to doodle. To cross-hatch. To spray paint. To fill-in. To erase.

And even if you're not a terrific artist, MacPaint includes special tools for designing everything from office forms to technical illustrations. Plus type styles to create captions, labels and headlines.

So you can have custom-designed graphics without hiring a design studio. Make your presentations more present-

of the job, not the limitations of your computer.

But more important than the practical benefits of programs like MacPaint and MacProject, they represent the very tangible difference an attitude can make.

An attitude that the only thing

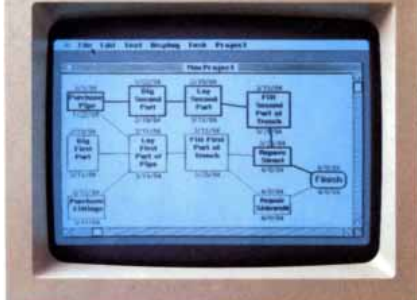
imagination of the people creating it.

Not just the engineers who design it, but software developers like Lotus® Development Corporation, currently developing a Macintosh version of their 1-2-3™ program.

And Software Publishing Corp., with a new pfs® filing program as easy to use as the Macintosh it was designed for.

And Microsoft, with Productivity-Tools, like Multiplan, Chart, File and Word.

If Macintosh has an extraordinary future ahead of it, it's because of the extraordinary people behind it.



MacProject does for project management what VisiCalc® did for spreadsheets.



MacPaint can create both freehand sketches and precise technical illustrations.



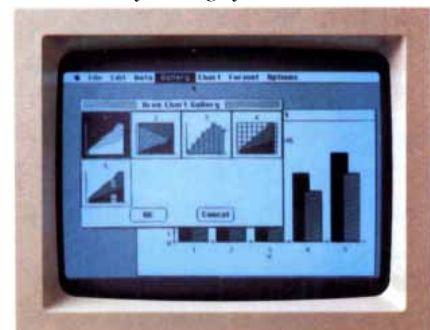
If you don't see a typeface you like here, Macintosh lets you design your own.

"To create a new standard takes something that's not just a little bit different. It takes something that's really new and captures people's imaginations. Macintosh meets that standard." — Bill Gates, Chairman of the Board & CEO Microsoft Corporation.

"Macintosh is much more natural, intuitive and in line with how people think and work... this is going to change the way people think about personal computers. Macintosh

sets a whole new standard, and we want our products to take advantage of this." — Mitch Kapor, President & CEO, Lotus Development Corporation.

"If you were to put machine 'x' on the table and a Macintosh on the table beside it, and then put pfs software on both machines... like a taste test... we think Macintosh's benefits would be pretty obvious." — Fred Gibbons, President, Software Publishing Corporation.



Microsoft's Chart displays a more graphic approach to business graphics.



Using insets with MacPaint, you can even illustrate your illustrations.



With Macintosh's unlimited graphics, there'll soon be no limit to the games it can play.



What makes Macintosh tick. And, someday, talk.

Macintosh has a lot in common with that most uncommon computer, the Lisa™ personal office system.

The garden variety 16-bit 8088 microprocessor.



Macintosh's 32-bit MC68000 microprocessor.



Its brain is the same blindingly-fast 32-bit MC68000 microprocessor—far more powerful than the 16-bit 8088 found in current generation computers.

Its heart is the same Lisa Technology of windows, icons, pull-down menus and mouse commands—all of which makes that 32-bit power far more useful by making Macintosh far easier to use than current generation computers.

And, thanks to its size, if you can't bring the problem to a Macintosh, you can always bring a Macintosh to



Standard 5-1/4" floppy disk.

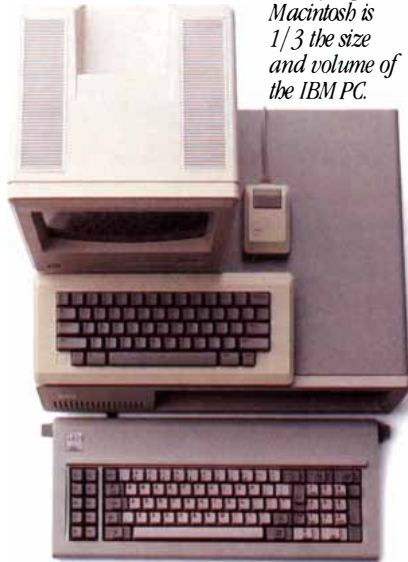


Macintosh's 400K 3-1/2" disk.



the problem. (Macintosh actually weighs 9 pounds less than the most popular "portable.")

Small footprint. Macintosh is 1/3 the size and volume of the IBM PC.



Another miracle of miniaturization is Macintosh's built-in 3½" microfloppy drive. Its 3½" disks store more than conventional 5¼" floppies—400K. So while they're big enough to hold a desk-full of work, they're small enough to fit in a shirt pocket.

And speaking of talking, Macintosh has a built-in polyphonic sound generator capable of producing high quality speech or music.

On the back of the machine, you'll find built-in RS232 and RS422 AppleBus serial communications ports. Which means you can connect printers,

modems and other peripherals without adding \$150 cards. It also means that Macintosh is ready to hook in to a local area network. (With AppleBus, you can interconnect up to 16 different Apple® computers and peripherals.)

Should you wish to double Macintosh's storage with an external disk drive, you can do so without paying extra for a disk controller card—that connector's built-in, too.

And, of course, there's a built-in connector for Macintosh's mouse, a feature that costs up to \$300 on computers that can't even run mouse-controlled software.

Of course, the real genius of Macintosh isn't its serial ports or its polyphonic sound generator.

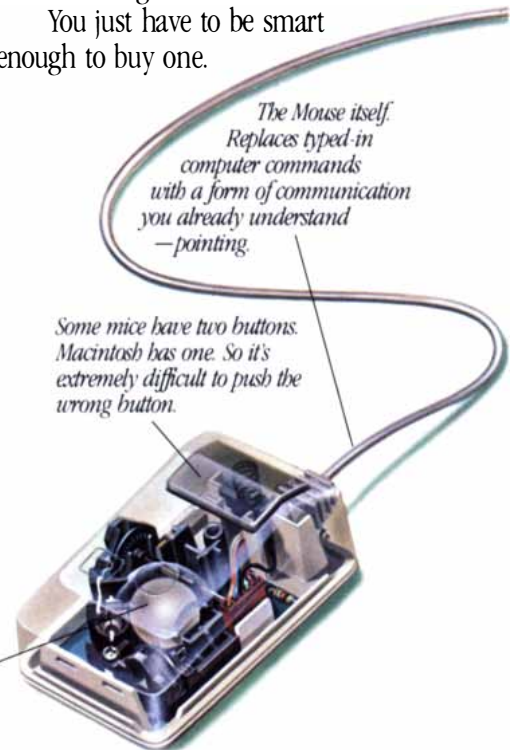
The real genius is that you don't have to be a genius to use a Macintosh.

You just have to be smart enough to buy one.

The Mouse itself. Replaces typed-in computer commands with a form of communication you already understand—pointing.

Some mice have two buttons. Macintosh has one. So it's extremely difficult to push the wrong button.

The inside story—a rotating ball and optical sensors translate movements of the mouse to Macintosh's screen pointer with pin-point accuracy.



13.5"

10.9"

9.7"



Mouse connector:

External disk drive connector:

Polyphonic sound port.

RS232, RS422 AppleBus serial communications ports for printers, modems and other peripherals.

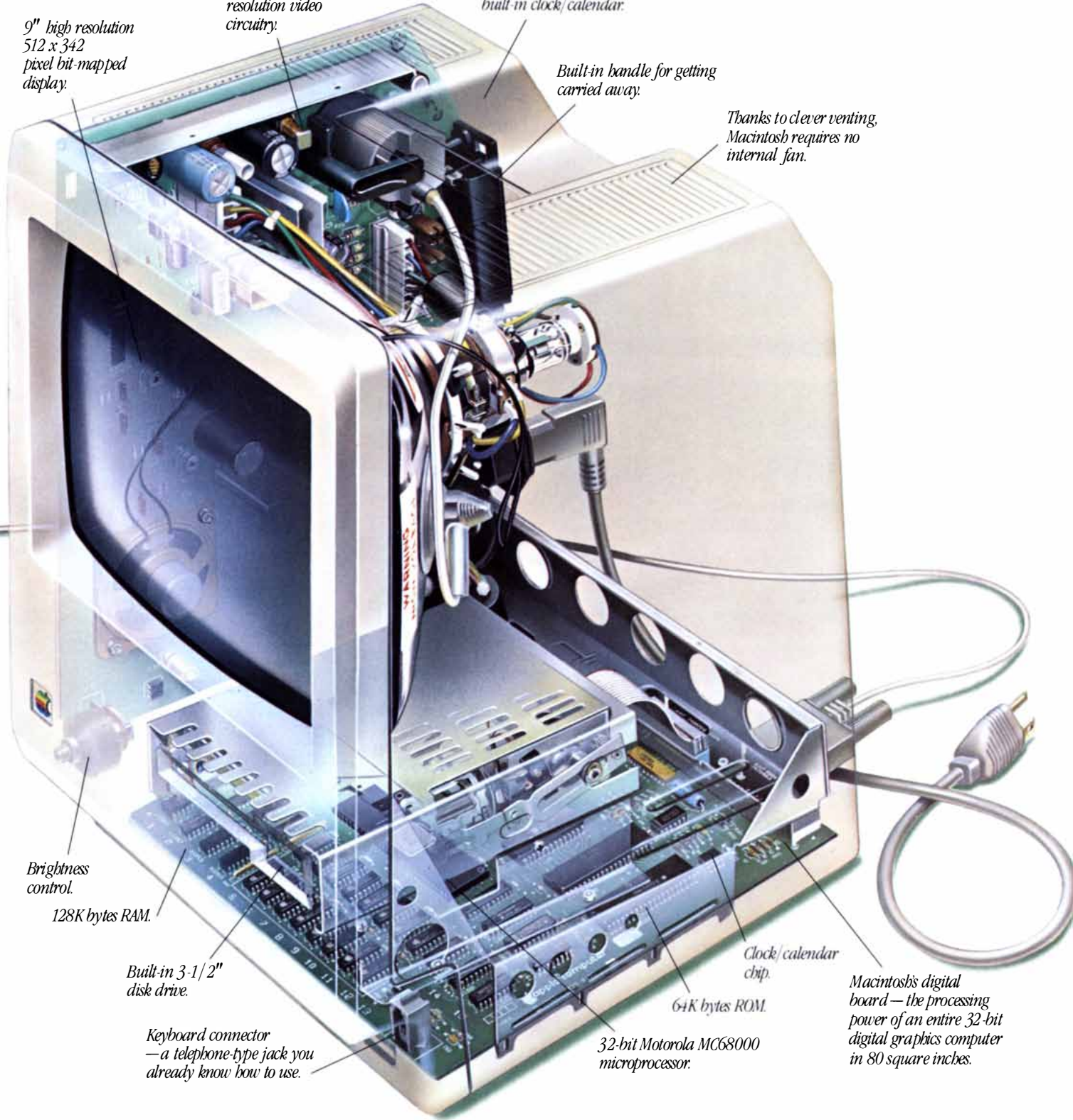
Ultra compact, switching-type power supply and high resolution video circuitry.

Battery for Macintosh's built-in clock/calendar.

Built-in handle for getting carried away.

Thanks to clever venting, Macintosh requires no internal fan.

9" high resolution 512 x 342 pixel bit-mapped display.



Brightness control.

128K bytes RAM.

Built-in 3-1/2" disk drive.

Keyboard connector — a telephone-type jack you already know how to use.

Clock/calendar chip.

64K bytes ROM.

32-bit Motorola MC68000 microprocessor.

Macintosh's digital board — the processing power of an entire 32-bit digital graphics computer in 80 square inches.

What to give the computer that has everything.

Macintosh comes well outfitted. The system includes the main unit (computer, display, built-in disk drive and firmware), a detached keyboard you can put wherever it feels most comfortable, the mouse, a System Disk (Finder and Desk Accessories), a Guided

Tour of Macintosh tutorial disk and audio cassette, and one (count it), one manual.

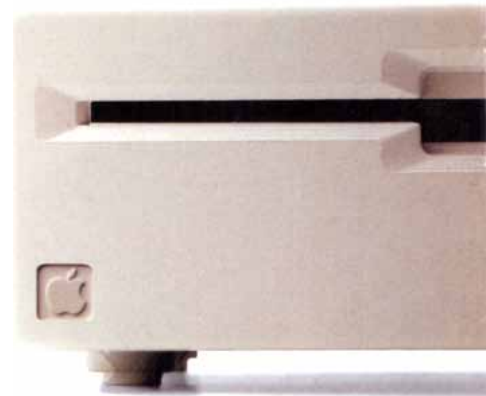
Everything you'll need to start doing everything you'll need to do.

But, should your needs suddenly expand, so can Macintosh. As easily as putting a plug in a socket.

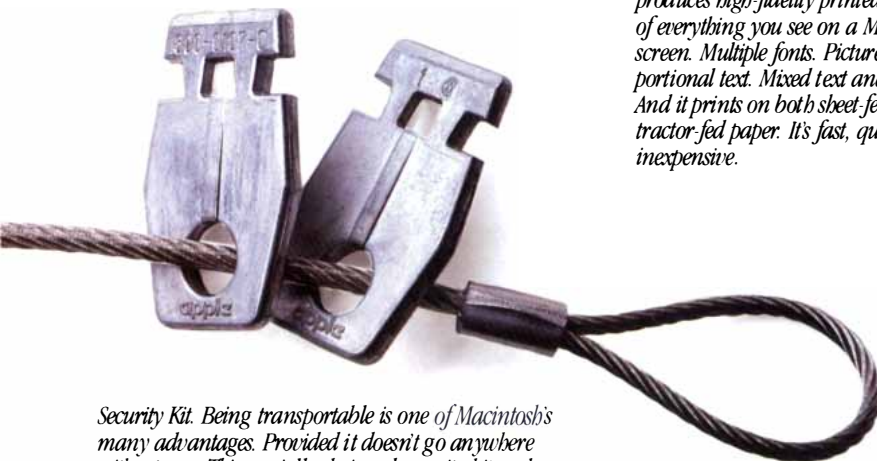
Apple Numeric Keypad. Patterned after the standard accountant's calculator 10-key pad, the Numeric Keypad speeds up doing spreadsheets, accounting, any number of number-related tasks. It plugs directly into the keyboard, and works with Macintosh applications.



Apple Imagewriter Printer. Imagewriter produces high-fidelity printed copy of everything you see on a Macintosh screen. Multiple fonts. Pictures. Proportional text. Mixed text and graphics. And it prints on both sheet-fed and tractor-fed paper. It's fast, quiet and inexpensive.



Apple Modem. Using MacTerminal, a standard telephone and the Apple Modem, you can plug yourself into electronic information services like Dow Jones News/Retrieval™, The Source™ and CompuServe®. Or communicate with other computers. It operates completely automatically, with both auto-dial and auto-answer, and comes in 300 and 300/1200 baud models.



Security Kit. Being transportable is one of Macintosh's many advantages. Provided it doesn't go anywhere without you. This specially designed security kit makes sure it doesn't. Metal plates snap into the main unit and keyboard. Then, a strong, steel cable loops through and locks to your desk.



Soft Carrying Case. At less than 20 pounds in weight, Macintosh is easily carried from here to there. But handles always help. This durable, water-resistant carrying case is thickly padded so the Macintosh main unit, keyboard, mouse, manual and disks fit snugly inside.

Macintosh External Disk Drive. By adding a second high-capacity (400K bytes) 3-1/2" disk drive like the one already built into your Macintosh, you can access more documents and programs without swapping disks. It also speeds making back-up copies of your information.

MacProject, MacDraw and Lisa are trademarks of Apple Computer, Inc.

Macintosh is a trademark licensed to Apple Computer, Inc.

DEC is a registered trademark of Digital Equipment Corporation.

Machines Corporation.

1-2-3 and Lotus are trademarks of Lotus Development Corporation.

Microsoft is a registered trademark of Microsoft Corporation.

Multiplan is a trademark of Microsoft Corporation.

Corporation.

VisiCalc is a registered trademark of VisiCorp.

The Source is a servicemark of Source TeleComputing Corporation, a subsidiary of The Readers Digest Association, Inc.

Company, Inc.

CompuServe is a registered trademark of CompuServe Corp.

WordStar is a trademark of MicroPro International Corporation.

Printed in USA.

We could, as they say in computerese, dump another Gigabyte (write another 50,000 or so pages) on Macintosh.

But you really can't appreciate how insanely great Macintosh is until you bring your index finger to an authorized Apple dealer.

Over 1,500 of them are eagerly waiting to put a mouse in your hand. To prove that, if you can point, you can use a Macintosh.



And if you can fill out a credit application, in most cases you can take one home the very same day. With the help of an Apple credit card.

Which makes owning the world's newest computer just as easy as using it.

Soon there'll be just two kinds of people. Those who use computers. And those who use Apples.



For an authorized Apple dealer nearest you call (800) 538-9696. In Canada, call (800) 268-7796 or (800) 268-7637. © 1984 Apple Computer Inc.

THE ESPN AUDIENCE HAS 40% MORE COMPUTER BUYERS. YOU'LL BE IN THE CHIPS.

ESPN's audience buys more computer hardware and software compared to the U.S. average. 56% more ESPN viewing households own video games. 33% more own home computers. And 40% more of ESPN's adult male viewers are responsible for the purchase of business computer systems.

Our audience spends more on computers because they can afford to. They have a higher household income than the U.S. average.

But you're not reaching this high consuming, upscale audience with your broadcast network buys anymore. More affluent viewers are turning to cable TV and away from the networks. The

ESPN REPLACES NETWORK UNDERDELIVERY IN BASIC AND PAY TV HH							
	TOTAL	TOTAL U.S.		CABLE		PAYCABLE	
	COST \$(M)	GRP	INDEX	GRP	INDEX	GRP	INDEX
NETWORK SPORTS SCHEDULE (# SPORTS ANNOUNCEMENTS)	2500	323	100	268	83	273	85
NETWORK ESPN SPORTS ANNOUNCEMENTS	2500	301	93	348	108	423	131

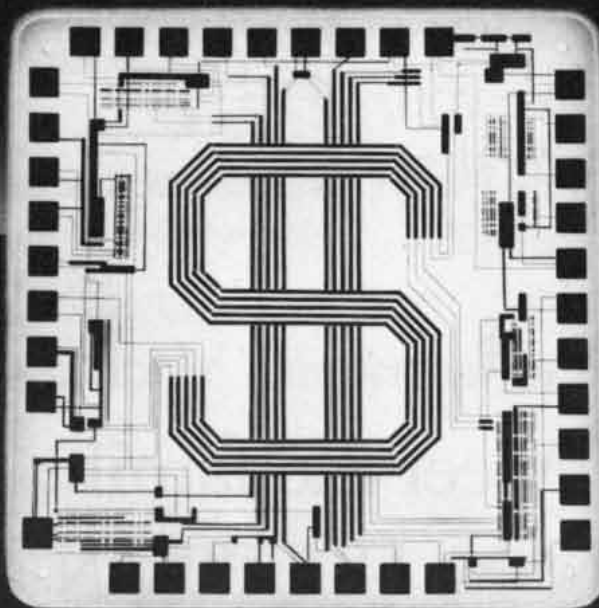
SOURCE: July 82 NTA On-Line Cable Facility Costs based on 1982 industry estimates.

networks just can't deliver these homes like they used to. But ESPN can.

Adding ESPN to a network sports schedule provides greater weight in TV households with cable and particularly pay cable. And because of our heavy concentration of male viewers, we deliver more upscale

men per household than the broadcast networks. All for a low out of pocket cost. How do we do it? It's simple. ESPN is the Total Sports Network. And as everyone knows, men love to watch sports. And that's exactly what they've been doing.

So, plug into ESPN's audience and you'll be programmed for success.



THE TOTAL SPORTS NETWORK™

© 1984 ESPN

ADVERTISING SALES OFFICES: NEW YORK (212) 661-6040 • DETROIT (313) 874-1818 • LOS ANGELES (213) 738-6003 • CHICAGO (312) 938-4223

Apple's® new baby has



Microsoft BASIC
on Apple's new Macintosh

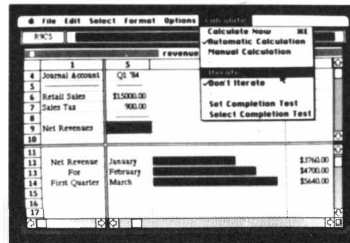
It's called Macintosh™. And it has a lot of our personality.

We're called Microsoft®. And our part of Macintosh is five new programs that are bright, intuitive, outgoing, understanding and born to perform.

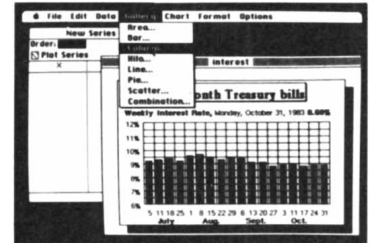
Our pride, your joy.

Taking advantage of Macintosh's mouse and rich graphics, we've designed software that works like you, even thinks like you.

All our programs share the same plain English commands. So what once took days to learn, now takes hours or minutes to learn with Macintosh.



Microsoft Multiplan



Microsoft Chart

Meet the family.

Our financial whiz is MULTIPLAN®, an electronic spreadsheet that actually remembers how you work. Even offers suggestions on spreadsheet set-up.

When it comes to writing, nothing will travel faster than

our best features.

our WORD, available this Fall. Using the mouse, it will let you select commands faster than you can say "cheese."

Our most artistic child will be CHART. Available late this Summer.

It gives you 42 presentation-quality chart and graphic styles to choose from.

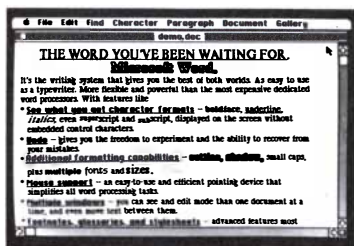
Later this year, we'll offer our most manageable child, FILE, an advanced personal record management program.

MICROSOFT
The High Performance Software

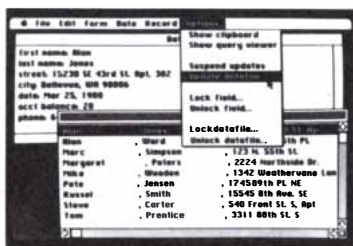
And BASIC, the language spoken by nine out of ten

microcomputers world-wide, is the granddaddy of them all.

Now enhanced to take full advantage of the



Microsoft Word

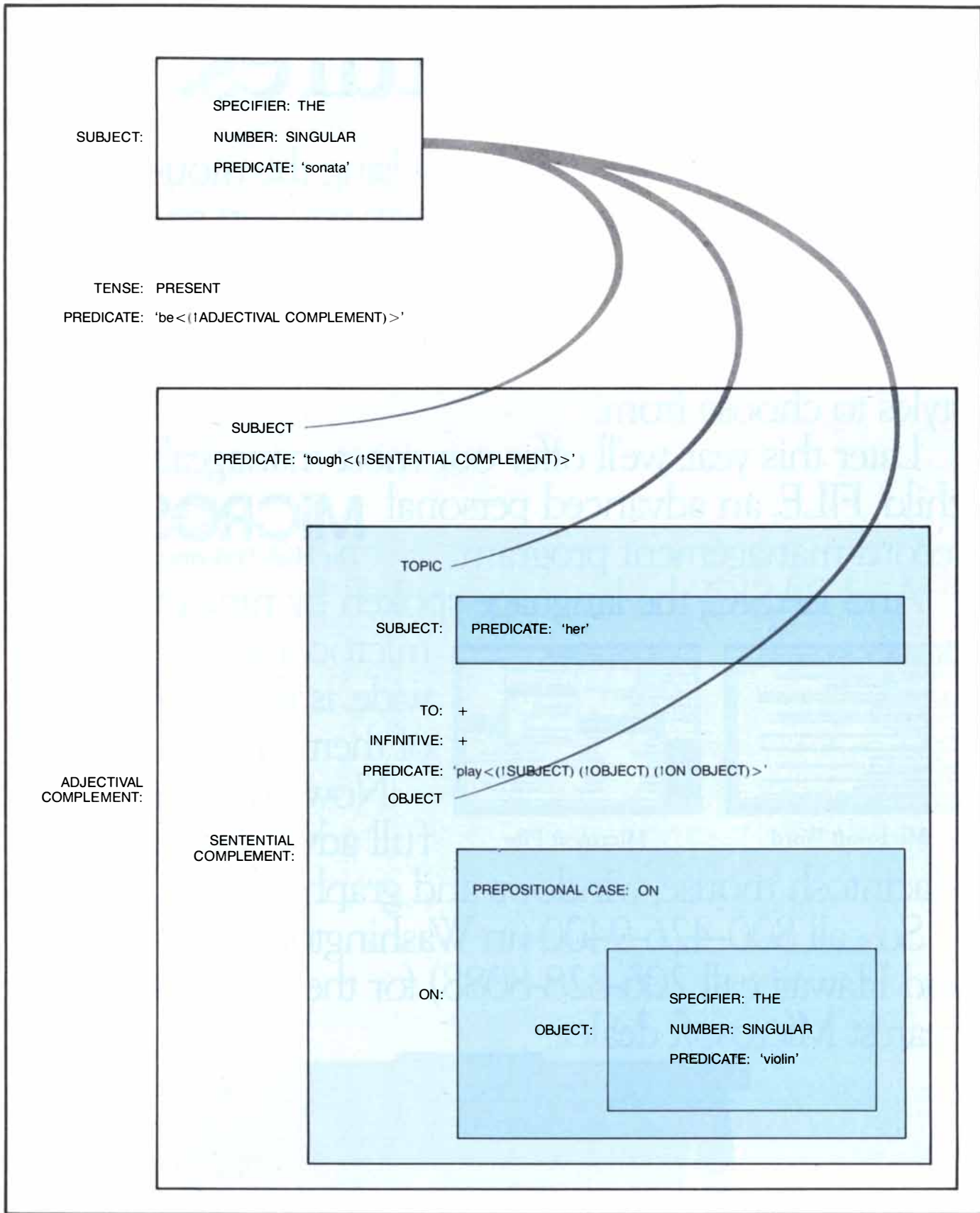


Microsoft File

Macintosh mouse, windows and graphics.

So call 800-426-9400 (in Washington State, Alaska and Hawaii call 206-828-8088) for the name of your nearest Microsoft dealer.

MICROSOFT



REPRESENTATION OF A SENTENCE in a way that makes explicit the linguistic relations among its parts has been a goal of the science of linguistics; it is also a necessary aspect of the effort to design computer software that "understands" language, or at any rate can draw inferences from linguistic input. In this illustration a sentence is given in "functional structure" form, which has the property that

when part of a sentence plays a role in another part, the former is "nested" in the latter. The nesting is shown by placing one box in another, or (in three places) by a "string." The sentence was analyzed by Ronald M. Kaplan and Joan Bresnan of Stanford University and the Xerox Corporation's Palo Alto Research Center. Another functional-structure diagram appears in the illustration on pages 142-143.

Computer Software for Working with Language

Programs can manipulate linguistic symbols with great facility, as in word-processing software, but attempts to have computers deal with meaning are vexed by ambiguity in human languages

by Terry Winograd

In the popular mythology the computer is a mathematics machine: it is designed to do numerical calculations. Yet it is really a language machine: its fundamental power lies in its ability to manipulate linguistic tokens—symbols to which meaning has been assigned. Indeed, “natural language” (the language people speak and write, as distinguished from the “artificial” languages in which computer programs are written) is central to computer science. Much of the earliest work in the field was aimed at breaking military codes, and in the 1950’s efforts to have computers translate text from one natural language into another led to crucial advances, even though the goal itself was not achieved. Work continues on the still more ambitious project of making natural language a medium in which to communicate with computers.

Today investigators are developing unified theories of computation that embrace both natural and artificial languages. Here I shall concentrate on the former, that is, on the language of everyday human communication. Within that realm there is a vast range of software to be considered. Some of it is mundane and successful. A multitude of microcomputers have invaded homes, offices and schools, and most of them are used at least in part for “word processing.” Other applications are speculative and far from realization. Science fiction is populated by robots that converse as if they were human, with barely a mechanical tinge to their voice. Real attempts to get computers to converse have run up against great difficulties, and the best of the laboratory prototypes are still a pale reflection of the linguistic competence of the average child.

The range of computer software for processing language precludes a comprehensive survey; instead I shall look at four types of program. The programs deal with machine translation, with word processing, with question an-

swering and with the adjuncts to electronic mail known as coordination systems. In each case the key to what is possible lies in analyzing the nature of linguistic competence and how that competence is related to the formal rule structures that are the theoretical basis of all computer software.

The prospect that text might be translated by a computer arose well before commercial computers were first manufactured. In 1949, when the few working computers were all in military laboratories, the mathematician Warren Weaver, one of the pioneers of communication theory, pointed out that the techniques developed for code breaking might be applicable to machine translation.

At first the task appears to be straightforward. Given a sentence in a source language, two basic operations yield the corresponding sentence in a target language. First the individual words are replaced by their translations; then the translated words are reordered and adjusted in detail. Take the translation of “Did you see a white cow?” into the Spanish “¿Viste una vaca blanca?” First one needs to know the word correspondences: “vaca” for “cow” and so on. Then one needs to know the structural details of Spanish. The words “did” and “you” are not translated directly but are expressed through the form of the verb “viste.” The adjective “blanca” follows the noun instead of preceding it as it does in English. Finally, “una” and “blanca” are in the feminine form corresponding to “vaca.” Much of the early study of machine translation dwelt on the technical problem of putting a large dictionary into computer storage and empowering the computer to search efficiently in it. Meanwhile the software for dealing with grammar was based on the then current theories of the structure of language, augmented by rough-and-ready rules.

The programs yielded translations so bad that they were incomprehensible. The problem is that natural language does not embody meaning in the same way that a cryptographic code embodies a message. The meaning of a sentence in a natural language is dependent not only on the form of the sentence but also on the context. One can see this most clearly through examples of ambiguity.

In the simplest form of ambiguity, known as lexical ambiguity, a single word has more than one possible meaning. Thus “Stay away from the bank” might be advice to an investor or to a child too close to a river. In translating it into Spanish one would need to choose between “*orilla*” and “*banco*,” and nothing in the sentence itself reveals which is intended. Attempts to deal with lexical ambiguity in translation software have included the insertion of all the possibilities into the translated text and the statistical analysis of the source text in an effort to decide which translation is appropriate. For example, “*orilla*” is likely to be the correct choice if words related to rivers and water are nearby in the source text. The first strategy leads to complex, unreadable text; the second yields the correct choice in many cases but the wrong one in many others.

In structural ambiguity the problem goes beyond a single word. Consider the sentence “He saw that gasoline can explode.” It has two interpretations based on quite different uses of “that” and “can.” Hence the sentence has two possible grammatical structures, and the translator must choose between them [see bottom illustration on page 133].

An ambiguity of “deep structure” is subtler still: two readings of a sentence can have the same apparent grammatical structure but nonetheless differ in meaning. “The chickens are ready to eat” implies that something is about to eat something, but which are the chickens? One of the advances in linguistic

theory since the 1950's has been the development of a formalism in which the deep structure of language can be represented, but the formalism is of little help in deducing the intended deep structure of a particular sentence.

A fourth kind of ambiguity—semantic ambiguity—results when a phrase can play different roles in the overall meaning of a sentence. The sentence “David wants to marry a Norwegian” is an example. In one meaning of the sentence the phrase “a Norwegian” is referential. David intends to marry a particular person, and the speaker of the sentence has chosen an attribute of the person—her being from Norway—in order to describe her. In another meaning of the sentence the phrase is attributive. Neither David nor the speaker has a particular person in mind; the sentence simply means that David hopes to marry someone of Norwegian nationality.

A fifth kind of ambiguity might be called pragmatic ambiguity. It arises from the use of pronouns and special nouns such as “one” and “another.” Take the sentence “When a bright moon ends a dark day, a brighter one will follow.” A brighter day or a brighter moon? At times it is possible for translation software to simply translate the ambiguous pronoun or noun, thereby preserving the ambiguity in the translation. In many cases, however, this strategy is not available. In a Spanish translation of “She dropped the plate on the table and broke it,” one must choose either the masculine “*lo*” or the feminine “*la*” to render “it.” The choice forces the translator to decide whether the masculine “*plato*” (plate) or the feminine “*mesa*” (table) was broken.

In many ambiguous sentences the meaning is obvious to a human reader,

but only because the reader brings to the task an understanding of context. Thus “The porridge is ready to eat” is unambiguous because one knows porridge is inanimate. “There’s a man in the room with a green hat on” is unambiguous because one knows rooms do not wear hats. Without such knowledge virtually any sentence is ambiguous.

Although fully automatic, high-quality machine translation is not feasible, software is available to facilitate translation. One example is the computerization of translation aids such as dictionaries and phrase books. These vary from elaborate systems meant for technical translators, in which the function of “looking a word up” is made a part of a multilingual word-processing program, to hand-held computerized libraries of phrases for use by tourists. Another strategy is to process text by hand to make it suitable for machine translation. A person working as a “pre-editor” takes a text in the source language and creates a second text, still in the source language, that is simplified in ways facilitating machine translation. Words with multiple meanings can be eliminated, along with grammatical constructions that complicate syntactic analysis. Conjunctions that cause ambiguity can be suppressed, or the ambiguity can be resolved by inserting special punctuation, as in “the [old men] and [women].” After the machine translation a “post-editor” can check for blunders and smooth the translated text.

The effort is sometimes cost-effective. In the first place, the pre-editor and post-editor need not be bilingual, as a translator would have to be. Then too, if a single text (say an instruction manual) is to be translated into several languages, a

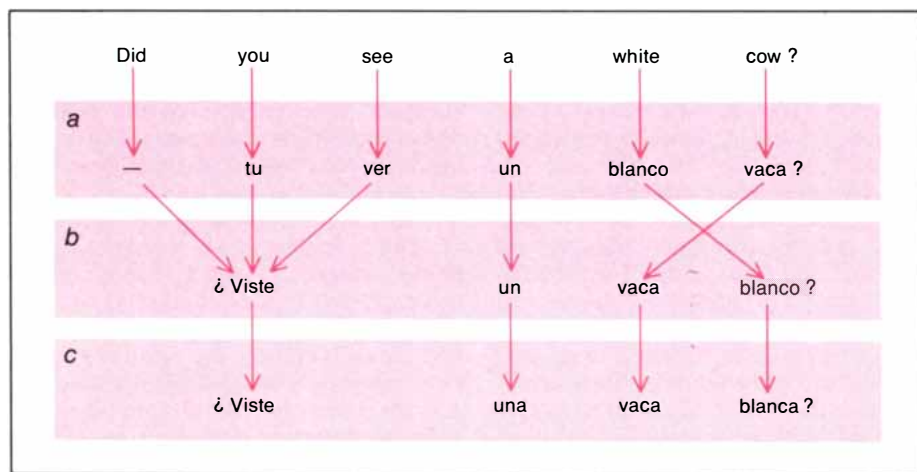
large investment in pre-editing may be justified because it will serve for all the translations. If the author of the text can be taught the less ambiguous form of the source language, no pre-editor is needed. Finally, software can help in checking the pre-edited text to make certain it meets the specifications for input to the translation system (although this is no guarantee that the translation will be acceptable).

A machine-translation system employing pre- and post-editing has been in use since 1980 at the Pan-American Health Organization, where it has translated more than a million words of text from Spanish into English. A new system is being developed for the European Economic Community, with the goal of translating documents among the official languages of the community: Danish, Dutch, English, French, German, Greek and Italian. Meanwhile the theoretical work on syntax and meaning has continued, but there have been no breakthroughs in machine translation. The ambiguity pervading natural language continues to limit the possibilities, for reasons I shall examine more fully below.

I turn next to word processing, that is, to software that aids in the preparation, formatting and printing of text. Word processors deal only with the manipulation and display of strings of characters and hence only with superficial aspects of the structure of language. Even so, they pose technical problems quite central to the design of computer software. In some cases the end product of a word-processing program is no more than a sequence of lines of text. In others it is a complex layout of typographic elements, sometimes with drawings intercalated. In still others it is a structured document, with chapter headings, section numbers and so on, and with a table of contents and an index compiled by the program.

The key problems in designing word-processing software center on issues of representation and interaction. Representation is the task of devising data structures that can be manipulated conveniently by the software but still make provision for the things that concern the user of the system, say the layout of the printed page. Interaction takes up the issue of how the user expresses instructions and how the system responds.

Consider the fundamental problem of employing the data-storage devices of a computer to hold an encoded sequence of natural-language characters. The first devices that encoded text were card-punch and teletype machines, and so the earliest text-encoding schemes were tailored to those devices. The teletype machine is essentially a typewriter that converts key presses into numerical codes that can be transmitted electronically;



MACHINE TRANSLATION of text from one language into another was thought to be quite feasible in the 1950's, when the effort was undertaken. In the first step of the process (a) the computer would search a bilingual dictionary to find translations of the individual words in a source sentence (in this case Spanish equivalents of the words in the sentence “Did you see a white cow?”). Next the translated words would be rearranged according to the grammar of the target language (b). The changes at this stage could include excision or addition of words. Finally, the morphology of the translation (for example the endings of words) would be adjusted (c).

"dBASE II[®] gave us something that money can't buy."

*Richard Sommers
Lead Programmer/Analyst
at a major health
maintenance organization.*



"dBASE II gave us time.
"And in the research
battle against breast cancer,
time is an invaluable
weapon.

"Our research people
are not computer people.
They're doctors and nurses.
So I had to write a customized
'layman's' application for them *very fast*."

**"My program development speed
even impressed me."**

"Using dBASE II, the relational
database management system (DBMS)
from Ashton-Tate, I was able to quickly
develop a very large and sophisticated
program for research data storage and
analysis. The real beauty of the new
program is its speed and ease of use.
A simple two-word command starts the
program, so data can be entered much
faster. And when our researchers need
to query the database, they ask their
questions in English using medical
terminology familiar to them, without
having to deal with computerese.

"In the past few months, I've
recommended dBASE II to at least four
of my programming colleagues in other
hospitals."

Put time on your side with dBASE II.

When you're customizing an
applications program and fighting the
clock at the same time, you won't find
a faster, more flexible solution than
dBASE II* from Ashton-Tate. We'll be
glad to rush you all the details. Ashton-
Tate, 10150 West Jefferson Boulevard,
Culver City, CA 90230. (800) 437-4329,
Ext. 217. In Colorado, (303) 799-4900.
In the U.K., call (0908) 568866.

ASHTON · TATE 

*Suggested retail price, \$700.
dBASE II is a registered trademark of Ashton-Tate.
©Ashton-Tate 1983

**America's largest household products company
Do they know something your company doesn't?**



Ask them. They're Procter & Gamble.

*Motorola is a world leader in advanced electronics
for memory, logic and voice and data communications.*

relies on our business information systems.



MOTOROLA / Four-Phase Systems



INTRODUCING RENAULT SPORTWAGON

What's the difference between the wagons from Renault, Volvo, and Mercedes? On the one hand, all three are engineered to carry a load in a very un-wagonlike manner. They come equipped with rack-and-pinion steering, front and rear anti-sway bars and sophisticated engines with 2 litre-plus displacement. And they carry an impressive list of luxury credentials at no extra cost. Features like AM/FM stereo, rear wiper/washer with defogger, ergonomically-designed seats and even air conditioning. But, while all three cars have much in common, there is one thing you'll carry in the Volvo and Mercedes that you won't in a Renault: larger payments. The front wheel drive Renault Sportwagon: the afford-

able European at **\$9,595.** Manufacturer's suggested retail price. Tax, license, destination charges, metallic clearcoat paint (\$150), optional equipment extra. Sold by American Motors.  Safety belts save lives.

RENAULT

THE ONE TO WATCH

thus there are teletype codes for most of the keys on a typewriter. The codes include the alphabetic characters *A* through *Z*, the digits 0 through 9 and common punctuation marks such as the period and the comma. Standards are harder to establish, however, for symbols such as #, @, ¢ and }. And what about keys that print nothing, such as the tab key, the carriage-return key and the backspace key?

The difficulties that arise in choosing standards can be illustrated by one peculiarity of text encoding. The teletype code distinguishes between a carriage return (which returns the type carriage to the beginning of the line without advancing the paper) and a line feed (which advances the paper without repositioning the carriage). Hence the end of a line is marked by a sequence of two characters: a carriage return and a line feed. One code would suffice, and so some programs eliminate either the carriage return or the line feed, or they replace both characters with another code entirely. The problem is that various programs employ different conventions, so that lines encoded by one program may not be readable by another.

The problems become worse when a full range of characters—punctuation marks, mathematical symbols, diacritical marks such as the umlaut—is considered. Moreover, word processing is now being extended to languages such as Chinese and Japanese, which require thousands of ideographic characters, and to languages such as Arabic and Hebrew, which are written from right to left. Coding schemes adequate for English are useless for alphabets with thousands of characters. It should be said that the schemes continue to vary because political and economic forces play a role in the design of computer systems. A given manufacturer wants to promulgate a standard that suits its own equipment; thus some present-day standards exist because they were offered by a vendor that dominates a market. On the other hand, technical matters such as the efficiency of certain software running on certain hardware perpetuate differences in detail. It will be quite a while before universal standards emerge and users gain the ability to transport text from one word-processing system to any other.

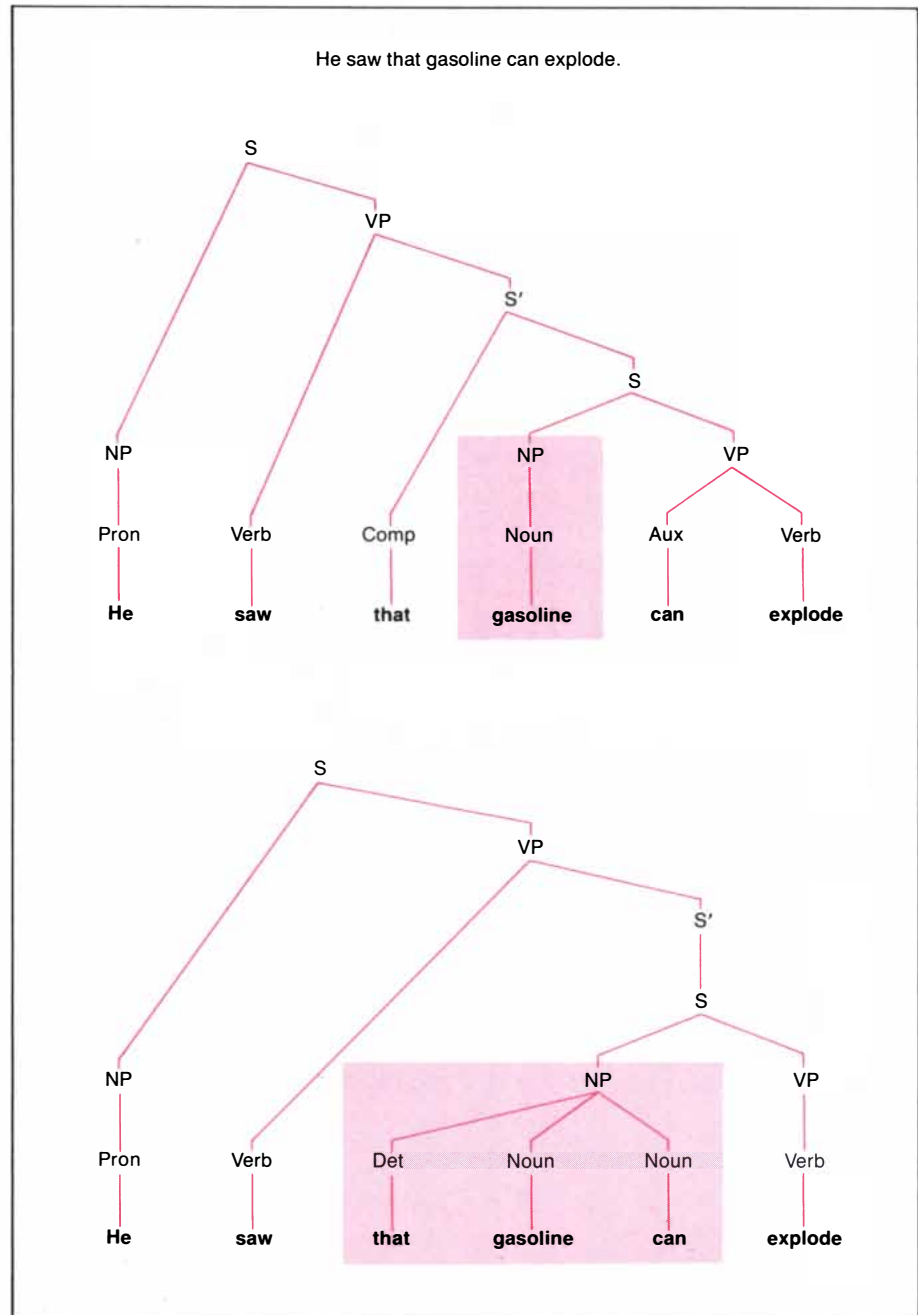
Encoding schemes aside, there is the form of the letters themselves. On a typewriter keyboard an *A* is simply an *A*. Typographically, however, an *A* is an *A* or an *A* or an **A**. In the new field of digital typography the computer is a tool for the design and presentation of forms of type. Some of the efforts in the field are applied to the forms themselves: in particular the representation of characters as composites of dots and spaces. Additional efforts go into the devising of code for the computer stor-

Stay away from the bank.

bank *n* 1. the rising terrain that borders a river or lake.

bank *n* 2. an establishment for the deposit, loan, issuance and transmission of money.

AMBIGUOUS MEANINGS permeate natural languages (that is, languages that people speak and write) and thus subvert the attempt to have computers translate text from one language into another. Here lexical ambiguity, the simplest type of ambiguity, is diagrammed. In lexical ambiguity a word in a sentence has more than one possible meaning. In this case the word is “bank” (*color*), which might equally well refer to either a river or a financial institution. A translator must choose. The following four illustrations show more complex types of ambiguity.



STRUCTURAL AMBIGUITY arises when a sentence can be described by more than one grammatical structure. Here the conflicting possibilities for the sentence “He saw that gasoline can explode” are displayed in the form of grammatical “trees.” In one of the trees the sentence has a subordinate clause whose subject is “gasoline” (*color*); the sentence refers to the recognition of a property of that substance. In the other tree “gasoline can” is part of a noun phrase (*NP*) meaning a container of gasoline; the sentence refers to the sight of a specific explosion.



WHAT PEOPLE CARRIED BEFORE THEY CARRIED THE CITIBANK PREFERRED VISA CARD.

The bulging wallet.
Universal symbol of success and power.

The Citibank Preferred Visa Card deflates that myth.

We offer you a credit line from \$5,000 to \$50,000. That's more than enough to start taking some credit cards out of your wallet.

You'll also get four times the acceptance of the American Express Gold Card. That's another card you can leave at home.

But most importantly, only the Citibank Preferred Visa lets you tap into the worldwide financial resources of Citicorp. That means opportunities to invest in CDs, high

interest savings plans,*—even the chance to buy gold bullion.

And everything you buy with your card earns you bonuses. You can use them to get guaranteed savings on anything from reproductions of antiques to original works of contemporary art.

To get all this power behind you, just fill out the attached application and send it in. But you'll need an income of at least \$25,000 to qualify. If you'd like more information, call us toll-free at 800-952-2152.

Then start emptying your wallet to make

room for the Citibank Preferred Visa. Even though it's just one card, you'll be carrying a lot more weight.

CITIBANK 
A CITICORP COMPANY



THE CARD TO END ALL CARDS

*Federal regulations require substantial penalties for early withdrawals from time accounts.
Copyright, Citicorp: 1984. Citibank (South Dakota), N.A. Member FDIC.

age of text that combines different fonts (such as Times Roman and Helvetica) and different faces (such as *italic* and **boldface**).

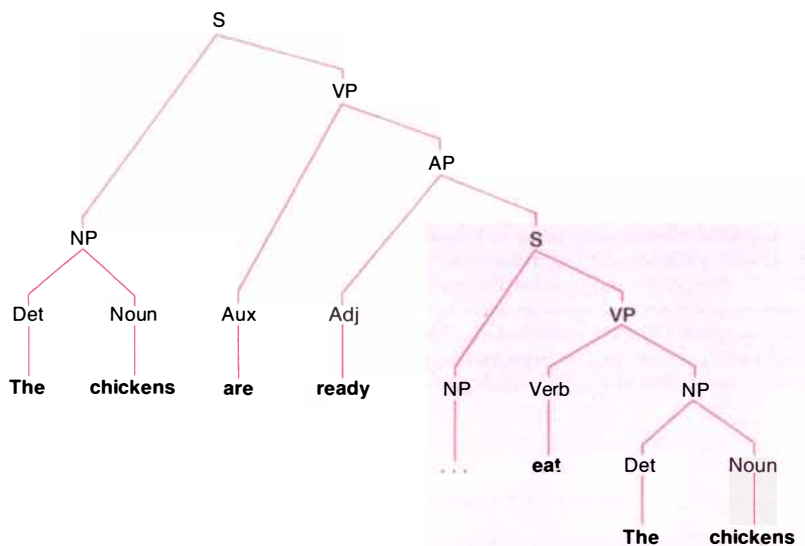
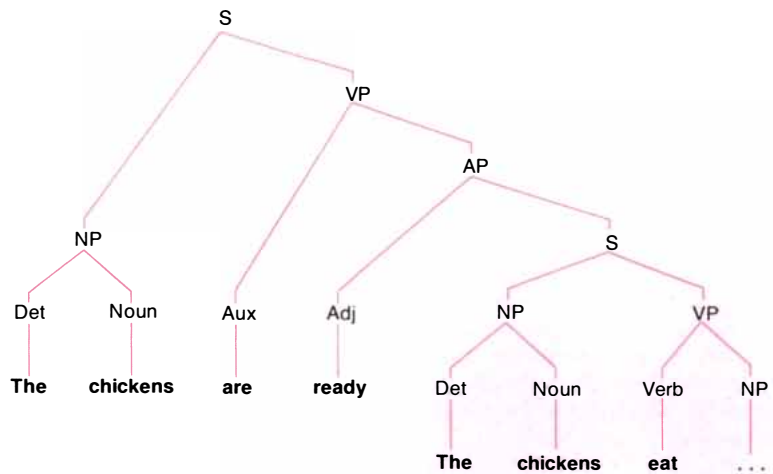
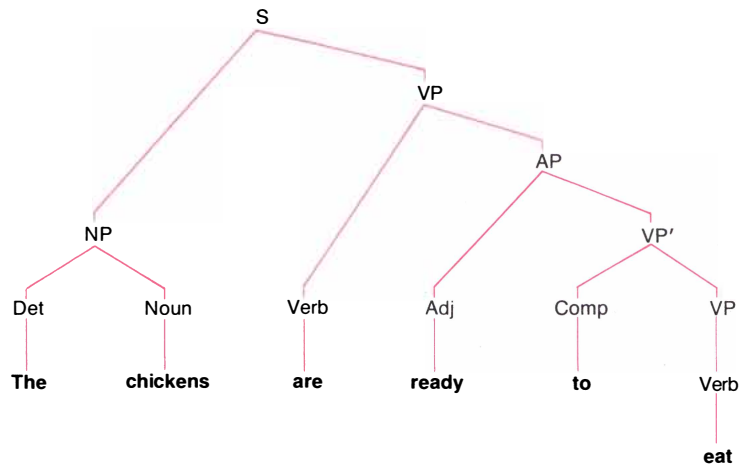
So far I have dealt only with stored sequences of characters. Yet one of the major tasks of a word-processing program is to deal with margins and spacing—with the “geography” of the printed page. In the typesetting language called TEX commands that specify non-standard characters, change the style of type, set the margins and so on are embedded in the text [see top illustration on page 138]. A command to TEX is distinguished from ordinary text by the backslash character (\). The stored text is “compiled” by the TEX program, which interprets the embedded commands in order to create a printed document in the specified format.

The compiling is quite complex, and a good deal of computation is often needed to get from code created by means of a word-processing program to code that readily drives a printer or a typesetting machine. An algorithm that justifies text (fills the full width of each line of type) must determine how many words will fit in a line, how much space should be inserted between the words and whether a line would be improved by dividing and hyphenating a word. The algorithm may also take actions to avoid visual defects such as a line with wide interword spacing followed by a line that is very compact. Positioning each line on the page is further complicated by the placement of headings, footnotes, illustrations, tables and so on. Mathematical formulas have their own typographic rules.

TEX and similar programs are primitive with respect to another aspect of word processing: the user interface. The high-resolution display screens becoming available are now making it possible for the computer to display to the user a fair approximation of the pages it will print, including the placement of each item and the typeface to be employed. This suggests that the user should not have to type special command sequences but might instead manipulate page geography directly on the screen by means of the computer keyboard and a pointing device such as a “mouse.” The resulting interface between the computer and the user would then fall into the class of interfaces known as WYSIWYG, which stands for “What you see is what you get.”

It is worth noting that programs for manipulating text are called different things by different professions. Programmers call them text editors, but in business and publishing they are referred to as word processors; in the latter fields an editor is a person who works to improve the quality of text. Computer software is emerging to aid in this

The chickens are ready to eat.



DEEP-STRUCTURAL AMBIGUITY arises when a sentence has a single apparent structure but nonetheless has more than one possible meaning. In this example the sentence is “The chickens are ready to eat.” Its grammatical structure (top) leaves the role of the chickens ambiguous: in one interpretation they will eat; in the other they will be eaten. Deep-structure trees make the chickens’ role explicit: they are the subject of the sentence (middle), in which case their food is undetermined, or they are the object (bottom), and their eaters are undetermined.

more substantive aspect of editing. It deals with neither the visual format of language nor the conceptual content but with spelling, grammar and style. It includes two kinds of programs: mechanized reference works and mechanized correctness aids.

An example of a mechanized reference work is a thesaurus program designed so that when the writer designates a word, a list of synonyms appears on the display screen. In advanced systems the thesaurus is fully integrated into the word-processing program. The writer positions a marker to indicate the word to be replaced. The thesaurus is then invoked; it displays the alternatives in a "window" on the screen. The writer positions the marker on one of the alternatives, which automatically replaces the rejected word.

The design of such a program involves both linguistic and computational issues. A linguistic issue is that the mechanism for looking up a word should be flexible enough to accept variant forms. For example, the store of information pertaining to "endow" should be accessible to queries about "endowed," "endowing," "endows" and even "unendowed" or "endowment." Recognizing the common root in such words calls for a morphological analysis, which can be done by techniques developed in the course of work on machine translation. Computational issues include devising methods for storing and searching through a thesaurus or a dictionary, which must be fairly large to be useful.

A correctness aid deals with spelling, grammar and even elements of style. The simplest such programs attempt to match each word in a text with an entry in a stored dictionary. Words that have no match are flagged as possible misspellings. Other programs look for common grammatical errors or stylistic infelicities. For example, the Writer's Workbench software developed at AT&T Bell Laboratories includes programs that search for repeated words, such as "the the" (a common typing mistake), for incorrect punctuation such as "?." and for wordy phrases such as "at this point in time." A different correctness aid calls attention to "pompous phrases" such as "exhibit a tendency" and "arrive at a decision" and suggests simpler replacements such as "tend" and "decide." Still another correctness aid searches for gender-specific terms such as "mailman" and "chairman" and suggests replacements such as "mail carrier" and "chairperson."

In addition to searching a text for particular strings of characters, some correctness-aid programs do statistical analyses. By calculating the average length of sentences, the length of words and similar quantities, they compute a "readability index." Passages that score poorly can be brought to the writer's attention. No program is yet able to make a comprehensive grammatical analysis of a text, but an experimental system called Epistle, developed at the International Business Machines Corporation, makes some grammatical judgments. It employs a grammar of

400 rules and a dictionary of 130,000 words. As with all software that tries to parse text without dealing with what the text means, there are many sentences that cannot be analyzed correctly.

Is there software that really deals with meaning—software that exhibits the kind of reasoning a person would use in carrying out linguistic tasks such as translating, summarizing or answering a question? Such software has been the goal of research projects in artificial intelligence since the mid-1960's, when the necessary computer hardware and programming techniques began to appear even as the impracticability of machine translation was becoming apparent. There are many applications in which the software would be useful. They include programs that accept natural-language commands, programs for information retrieval, programs that summarize text and programs that acquire language-based knowledge for expert systems.

No existing software deals with meaning over a significant subset of English; each experimental program is based on finding a simplified version of language and meaning and testing what can be done within its confines. Some investigators see no fundamental barrier to writing programs with a full understanding of natural language. Others argue that computerized understanding of language is impossible. In order to follow the arguments it is important to examine the basics of how a language-understanding program has to work.

A language-understanding program needs several components, corresponding to the various levels at which language is analyzed [see illustrations on pages 138–144]. Most programs deal with written language; hence the analysis of sound waves is bypassed and the first level of analysis is morphological. The program applies rules that decompose a word into its root, or basic form, and inflections such as the endings *-s* and *-ing*. The rules correspond in large part to the spelling rules children are taught in elementary school. Children learn, for example, that the root of "baking" is "bake," whereas the root of "barking" is "bark." An exception list handles words to which the rules do not apply, such as forms of the verb "be." Other rules associate inflections with "features" of words. For example, "am going" is a progressive verb: it signals an act in progress.

For each root that emerges from the morphological analysis a dictionary yields the set of lexical categories to which the root belongs. This is the second level of analysis carried out by the computer. Some roots (such as "the") have only one lexical category; others have several. "Dark" can be a noun or

David wants to marry a Norwegian.

$\exists x \text{ Norwegian}(x) \wedge \text{Want}(\text{David}, [\text{Marry}(\text{David}, x)])$

$\text{Want}(\text{David}, [\exists x \text{ Norwegian}(x) \wedge \text{Marry}(\text{David}, x)])$

SEMANTIC AMBIGUITY arises when a phrase can play different roles in the meaning of a sentence. Here the roles of the phrase "a Norwegian" become explicit when the sentence "David wants to marry a Norwegian" is "translated" into a logical form based on the notation called predicate calculus. According to one interpretation, the speaker of the sentence has a particular person in mind and has chosen nationality as a way to specify who. Hence the sentence means: There exists (\exists) an x such that x is Norwegian and (\wedge) x is the person David wants to marry. According to another interpretation, neither David nor the speaker has any particular person in mind. David might be going to Norway hoping to meet someone marriageable.

She dropped the plate on the table and broke it.

She dropped the plate on the table and broke [the plate].

She dropped the plate on the table and broke [the table].

PRAGMATIC AMBIGUITY arises when a sentence is given more than one possible meaning by a word such as the pronoun "it." Suppose a computer is given the sentence shown in the illustration. If the computer has access to stored knowledge of the grammar of English sentences but lacks access to commonsense knowledge of the properties of tables and plates, the computer could infer with equal validity that the table was broken or that the plate was broken.

What?

What is the latest R&D activity in Japan in the field of industrial robots?

What are the market trends for frozen orange juice?

What are the mechanical properties of shape memory materials?

Introducing In-Search— immediate answers to millions of questions like these— through DIALOG®.

In-Search is an easy-to-use personal computer software package that lets you instantly retrieve answers to your questions from DIALOG, the world's largest collection of online databases.

With In-Search, your personal computer and modem, you can access this information over telephone lines.

All the Answers.

Imagine scanning thousands of articles in seconds. In-Search can do just that.

In-Search gives you over 80 million articles from thousands of sources: Newspapers, magazines, technical journals, investment reports, wire services, annual reports and the Yellow Pages.

Easy Answers.

1. Type the words or subject you wish to research into your personal computer.
2. In-Search takes it from there, bringing up all the articles and references you need.
3. Simple on-screen graphics guide you through every step.

Fast Answers.

In-Search can find in minutes what could take days or weeks to find. You can now spend more time using information because you spend less time tracking it down.

Once you have the information, you can store it on disk or print it out. In-Search is compatible with most word processing programs.

How to get the Answers.

To see a demonstration or to purchase In-Search, call (408) 986-1200 for your nearest dealer. Or send this coupon and \$5.00 for a demonstration disk designed for IBM-compatible or T.I. personal computers.



In-Search™



Menlo Corporation

4633 Old Ironsides Dr.
Suite 400
Santa Clara, CA 95050

- Enclosed is \$5.00 for my In-Search demonstration diskette.
 Please send me a free booklet with more details.

NAME _____

COMPANY _____

PHONE _____

ADDRESS _____

CITY _____

STATE _____

ZIP _____

MY OCCUPATION IS _____

104

a \inset
 This is a sample of a {\iitalic justified} piece of text, which contains {\eightpoint small letters {\bold and }} {\bigFont big ones}. It includes foreign words such as \quote pe~na' quote—which is Spanish—and foreign letters like \alpha\ and \aleph, which can be baffling, and includes one \hskip 1.3in wide space.

b ...

01110100	01100101	01110010	01110011	00000000	00100111	00101101	11010011	00001000	01100001	01101110	01100100	00000000
t	e	r	s	NEW ENTITY	FONT CODE	X-POSITION	Y-POSITION	X-INCRE-MENT	a	n	d	NEW ENTITY
00110100	00110001	10110110	00101101	01100010	01101001	01100111	00100000	01101111	01101110	01100101	01101011	00101110
FONT CODE	X-POSITION	Y-POSITION	X-INCRE-MENT	b	i	g	SPACE	o	n	e	s	*
00000000	00000001	10101111	10110110	00101100	01001001	01110100	00100000	01101001	...			
NEW ENTITY	FONT CODE	X-POSITION	Y-POSITION	X-INCRE-MENT	l	t	SPACE	i				

c This is a sample of a *justified* piece of text, which contains small letters and **big ones**. It includes foreign words such as "peña"—which is Spanish—and foreign letters like α and ℵ, which can be baffling, and includes one wide space.

WORD PROCESSING, that is, the computer-aided preparation and editing of text, requires several representations of the text, because the format best for interactions between the software and its user is not efficient for sending instructions to a printing machine, nor can it efficiently give a preview of the result of the printing. In the typesetting language TEX the user's typed input (a) includes commands that specify nonstandard characters, change the style of type, set margins

and so on. Such commands are distinguished by a backslash (\). The TEX software "compiles" the input, producing computer code that will drive a printing machine (b). To that end the code is divided into "entities," each of which specifies the typeface and the starting position for a sequence of words. Coded "X increments" space out the words to fill the distance between margins on the printed page; thus they "justify" lines of type. The printed page (c) shows the result.

an adjective; "bloom" can be a noun or a verb. In some instances the morphological analysis limits the possibilities. (In its common usages "bloom" can be a noun or a verb, but "blooming" is only a verb.) The output of the morphological and lexical analysis is thus a sequence of the words in a sentence, with each word carrying a quantity of dictionary and feature information. This output serves in turn as the input to the third component of the program, the parser, or syntactic-analysis component, which applies rules of grammar to determine the structure of the sentence.

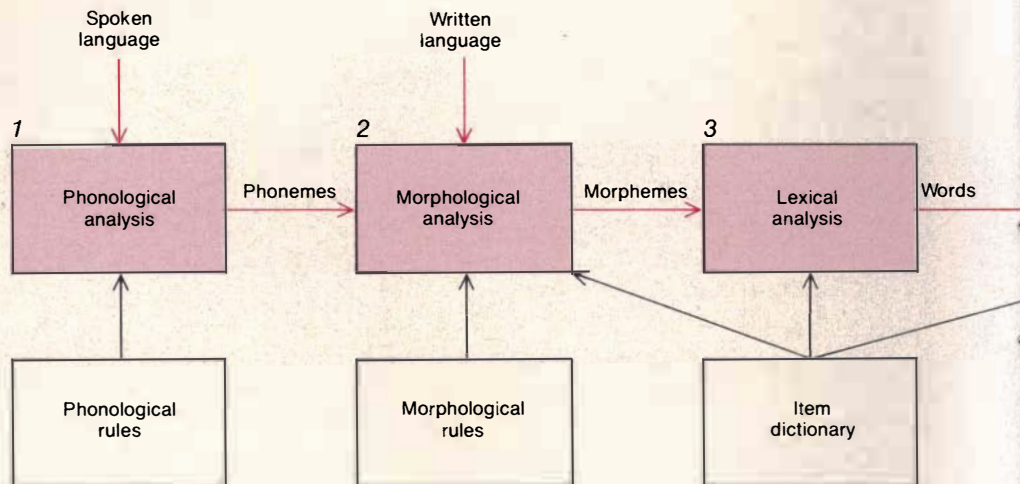
Two distinct problems arise in designing an adequate parser. The first problem is the specification of a precise set of rules—a grammar—that determines the set of possible sentence structures in a language. Over the past 30 years much work in theoretical linguistics has been directed toward devising formal linguistic systems: constructions in which the syntactic rules of a language are stated so precisely that a computer could employ them to analyze the language. The generative transformational grammars invented by Noam Chomsky of the Massachusetts Institute of Technology were the first comprehensive attempt; they specify the syntax of a language by means of a set of rules whose mechanical application generates all allowable structures.

The second problem is that of the parsing itself. It is not always possible to tell, when a part of a sentence is encoun-

tered, just what role it plays in the sentence or whether the words in it go together. Take the sentence "Roses will be blooming in the dark gardens we abandoned long ago." The words "in the dark" might be interpreted as a complete phrase; after all, they are grammatically well formed and they make sense. But the phrase cannot form a coherent unit in a complete analysis of the sentence because it forces "Roses will be blooming in the dark" to be interpreted

as a sentence and therefore leaves "gardens we abandoned long ago" without a role to play.

Parsers adopt various strategies for exploring the multiple ways phrases can be put together. Some work from the top down, trying from the outset to find possible sentences; others work from the bottom up, trying local word combinations. Some backtrack to explore alternatives in depth if a given possibility fails; others use parallel processing



COMPUTERIZED UNDERSTANDING OF LANGUAGE requires the computer to draw on several types of stored data (white boxes) and perform several levels of analysis (colored boxes). If the language is spoken, the first analysis is phonological (1); the computer analyzes sound waves. If the language is written, the first analysis is morphological (2); the computer decomposes each word into its root, or basic form, and inflections (for example -ing). Next is lexi-

to keep track of a number of alternatives simultaneously. Some make use of formalisms (such as transformational grammar) that were developed by linguists. Others make use of newer formalisms designed with computers in mind. The latter formalisms are better suited to the implementation of parsing procedures. For example, "augmented-transition networks" express the structure of sentences and phrases as an explicit sequence of "transitions" to be followed by a machine. "Lexical-function grammars" create a "functional structure" in which grammatical functions such as head, subject and object are explicitly tied to the words and phrases that serve those functions.

Although no formal grammar successfully deals with all the grammatical problems of any natural language, existing grammars and parsers can handle well over 90 percent of all sentences. This is not entirely to the good. A given sentence may have hundreds or even thousands of possible syntactic analyses. Most of them have no plausible meaning. People are not aware of considering and rejecting such possibilities, but parsing programs are swamped by meaningless alternatives.

The output of a parsing program becomes the input to the fourth component of a language-understanding program: a semantic analyzer, which translates the syntactic form of a sentence into a "logical" form. The point is to put the linguistic expressions into a form that makes it possible for the computer to apply reasoning procedures and draw inferences. Here again there are competing theories about what representation is most appropriate. As with parsing, the key issues are effectiveness and efficiency.

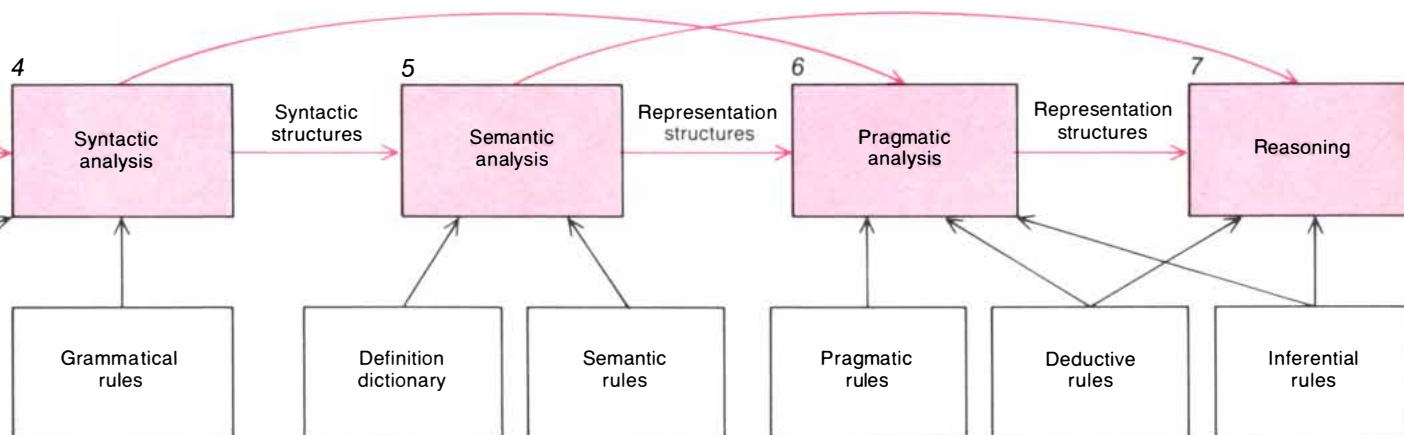
Effectiveness depends on finding the appropriate formal structures to encode the meaning of linguistic expressions. One possibility is predicate calculus, which employs the quantifiers \forall to mean "all" and \exists to mean "there exists." In predicate calculus "Roses will be blooming..." is equivalent to the assertion "There exists something that is a rose and that is blooming..." This entails a difficulty. Is one rose adequate to represent the meaning of "roses will be blooming," or would it be better to specify two or more? How can the computer decide? The dilemma is worsened if a sentence includes a mass noun such as "water" in "Water will be flowing..." One cannot itemize water at all. In designing a formal structure for the meaning of linguistic expressions many similar problems arise from the inherent vagueness of language.

Efficiency must also be considered, because the computer will employ the logical form of a sentence to draw inferences that in turn serve both the analysis of the meaning of the sentence and the formulation of a response to it. Some formalisms, such as predicate calculus, are not directly amenable to efficient computation, but other, more "procedural" representations have also been devised. Consider the effort to answer the question "Are there flowers in the gardens we abandoned long ago?" The computer needs to know that roses are flowers. This knowledge could be represented by a formula in predicate calculus amounting to the assertion "Everything that is a rose is a flower." The computer could then apply techniques developed for mechanical theorem-proving to make the needed deduction. A different approach would be to give certain inferences a privileged computational status. For example, basic clas-

sifications could be represented directly in data structures [see bottom illustration on page 144]. Such deductions are required constantly for reasoning about the ordinary properties of objects. Other types of fact (for example that flowers need water in order to grow) could then be represented in a form closer to predicate calculus. The computer could draw on both to make inferences (for example that if roses do not get water, they will not grow).

A good deal of research has gone into the design of "representation languages" that provide for the effective and efficient encoding of meaning. The greatest difficulty lies in the nature of human commonsense reasoning. Most of what a person knows cannot be formulated in all-or-nothing logical rules; it lies instead in "normal expectations." If one asks, "Is there dirt in the garden?" the answer is almost certainly yes. The yes, however, cannot be a logical inference; some gardens are hydroponic, and the plants there grow in water. A person tends to rely on normal expectations without thinking of exceptions unless they are relevant. But little progress has been made toward formalizing the concept of "relevance" and the way it shapes the background of expectations brought to bear in the understanding of linguistic expressions.

The final stage of analysis in a language-understanding program is pragmatic analysis: the analysis of context. Every sentence is embedded in a setting: it comes from a particular speaker at a particular time and it refers, at least implicitly, to a particular body of understanding. Some of the embedding is straightforward: the pronoun "I" refers to the speaker; the adverb "now" refers to the moment at which the sen-



cal analysis (3), in which the computer assigns words to their lexical category (noun, for instance) and identifies "features" such as plurals. Then comes syntactic analysis, or parsing (4): the application of rules of grammar to yield the structure of the sentence. After that comes semantic analysis (5). Here the sentence is converted into a

form that makes it amenable to inference-drawing. The final stage is pragmatic (6): it makes explicit the context of the sentence, such as the relation between the time at which it is spoken and the time to which it refers. The computer is now in a position to draw inferences (7), perhaps in preparation for responding to the sentence.

IBM PC Software: the value of choosing



Shoes.

If they don't fit, they're not worth wearing.

Software programs.

If they don't fit, they're not worth using.

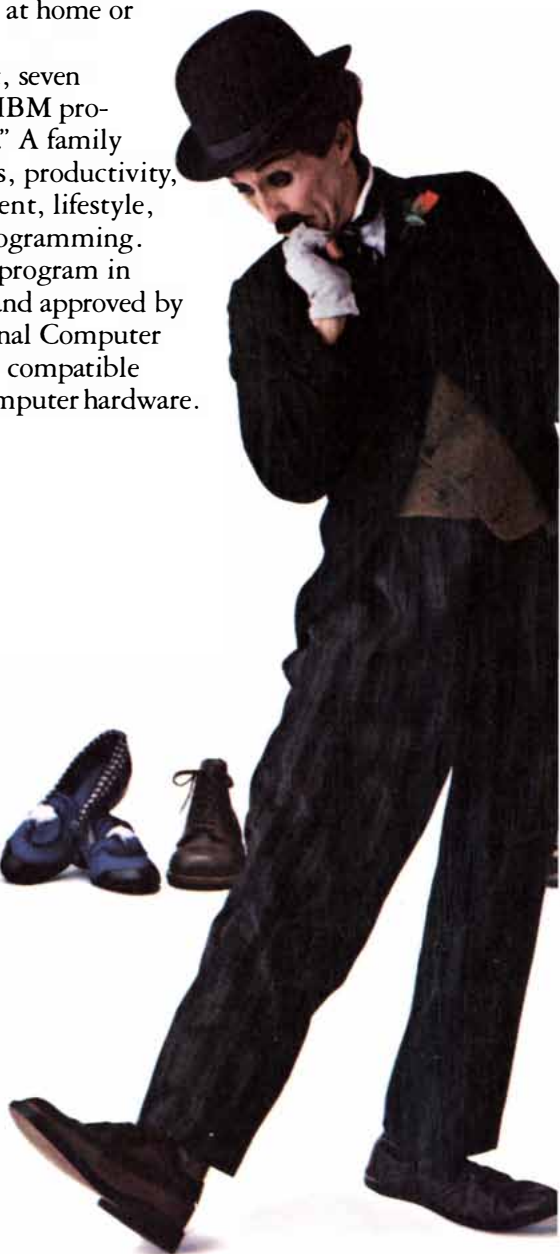
That's why it's altogether fitting that IBM Personal Computer Software offers you a choice.

Size up the selection.

You'll find many types of programs in the IBM software library. They'll help keep you on your toes in the office, at home or in school.

There are, in fact, seven different categories of IBM programs called "families." A family of software for business, productivity, education, entertainment, lifestyle, communications or programming.

Of course, every program in every family is tested and approved by IBM. And IBM Personal Computer Software is made to be compatible with IBM Personal Computer hardware.



programs that fit.

Putting your best foot forward.

Although every person isn't on equal footing when it comes to using personal computer software, there's something for almost everyone in the IBM software library.

For example, you may be on a shoestring budget and want a big selection of programs with small price tags.

You may be introducing students to computing and want programs that are simple to use and simple to learn.

You may run a business requiring sophisticated inventory and payroll programs. Or you may run a business requiring a single accounting program.

You may write interoffice memos and want a streamlined word processing program. Or you may be a novelist looking for a program with features worth writing home about.

Now you can find IBM Personal Computer Software that fits — to help you accomplish specific tasks and reach individual goals.

Stroll into a store today.

What's the next step?

Visit an authorized IBM Personal Computer dealer or IBM Product Center near you. To find out exactly where, call 800-447-4700. In Alaska or Hawaii, 800-447-0890.

Ask your dealer to demonstrate your choice of programs. Then get comfortable. Sit down at the keyboard and try IBM software on for size.



IBM[®]

Personal Computer Software

tence is uttered. Yet even these can be problematic: consider the use of "now" in a letter I write today expecting you to read it three or four days hence. Still, fairly uncomplicated programs can draw the right conclusion most of the time. Other embedding is more complex. The pronoun "we" is an example. "We" might refer to the speaker and the hearer or to the speaker and some third party. Which of these it is (and who the third party might be) is not explicit and in fact is a common source of misunderstanding when people converse.

Still other types of embedding are not signaled by a troublesome word such as "we." The sentence "Roses will be blooming..." presupposes the identification of some future moment when the roses will indeed be in bloom. Thus the sentence might have followed the sentence "What will it be like when we get home?" or "Summer is fast upon us." Similarly, the noun phrase "the dark gardens we abandoned long ago" has a context-dependent meaning. There may be only one instance of gardens in which we have been together; there may be more than one. The sentence presupposes a body of knowledge from which the gardens are identifiable. The point is that a phrase beginning with "the" rarely specifies fully the object to which it refers.

One approach to such phrases has been to encode knowledge of the world in a form the program can use to make inferences. For example, in the sentence "I went to a restaurant and the waiter was rude" one can infer that "the waiter" refers to the person who served the speaker's meal if one's knowledge includes a script, so to speak, of the typical

events attending a meal in a restaurant. (A particular waiter or waitress serves any given customer.) In more complex cases an analysis of the speaker's goals and strategies can help. If one hears "My math exam is tomorrow, where's the book?" one can assume that the speaker intends to study and that "the book" means the mathematics text employed in a course the speaker is taking. The approach is hampered by the same difficulty that besets the representation of meaning: the difficulty of formalizing the commonsense background that determines which scripts, goals and strategies are relevant and how they interact. The programs written so far work only in highly artificial and limited realms, and it is not clear how far such programs can be extended.

Even more problematic are the effects of context on the meaning of words. Suppose that in coming to grips with "the dark gardens we abandoned long ago" one tries to apply a particular meaning to "dark." Which should it be? The "dark" of "those dark days of tribulation" or that of "How dark it is with the lights off!" or that of "dark colors"? Although a kernel of similarity unites the uses of a word, its full meaning is determined by how it is used and by the prior understanding the speaker expects of the hearer. "The dark gardens" may have a quite specific meaning for the person addressed; for the rest of us it is slightly mysterious.

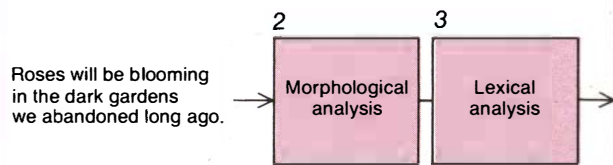
At first it might seem possible to distinguish "literal" uses of language from those that are more metaphorical or poetical. Computer programs faced with exclusively literal language could

then be freed from contextual dilemmas. The problem is that metaphor and "poetic meaning" are not limited to the pages of literature. Everyday language is pervaded by unconscious metaphor, as when one says, "I lost two hours trying to get my idea across." Virtually every word has an open-ended field of meanings that shade gradually from those that seem utterly literal to those that are clearly metaphorical.

The limitations on the formalization of contextual meaning make it impossible at present—and conceivably forever—to design computer programs that come close to full mimicry of human language understanding. The only programs in practical use today that attempt even limited understanding are natural-language "front ends" that enable the user of a program to request information by asking questions in English. The program responds with English sentences or with a display of data.

A program called SHRDLU is an early example. Developed in the late 1960's, it enables a person to communicate with a computer in English about a simulated world of blocks on a tabletop. The program analyzes requests, commands and statements made by the user and responds with appropriate words or with actions performed in the simulated scene. SHRDLU succeeded in part because its world of conversation is limited to a simple and specialized domain: the blocks and a few actions that can be taken with them.

Some more recent front-end interfaces have been designed with practical applications in mind. A person wanting access to information stored in the computer types natural-language sentences



Word	Root	Lexical categories	Features
Roses	rose	Noun	[plural]
will		Verb (auxiliary)	[modal]
be		Verb (auxiliary) Verb (copular)	[infinitive] [infinitive]
blooming	bloom	Verb (intransitive)	[progressive]
in		Preposition	
the		Determiner	[definite]
dark		Adjective Noun	[mass]
gardens	garden	Noun Verb	[plural] [third-person, singular, present]
we		Pronoun	[first-person, plural, nominative]
abandoned	abandon	Verb (transitive) Verb (transitive)	[past] [participle]
long		Adjective	
ago		Adverb	

SUCCESION OF ANALYSES done by a hypothetical computer program suggests how software that understands language works. In this illustration the program has been given the sentence "Roses will be blooming in the dark gardens we abandoned long ago." The first analyses (morphological and lexical) yield a list of the words in the

sentence, with their roots, their lexical categories and their features. "Blooming," for instance, is a progressive verb: it signifies an act in progress. The data serve as input for the syntactic level of analysis: the parsing of the sentence. Here the surface, or grammatical, structure of "Roses will be blooming..." is put in the form of a tree. Pre-

that the computer interprets as queries. The range of the questioning is circumscribed by the range of the data from which answers are formulated; in this way words can be given precise meaning. In a data base on automobiles, for example, "dark" can be defined as the colors "black" and "navy" and nothing more than that. The contextual meaning is there, but it is predetermined by the builder of the system, and the user is expected to learn it.

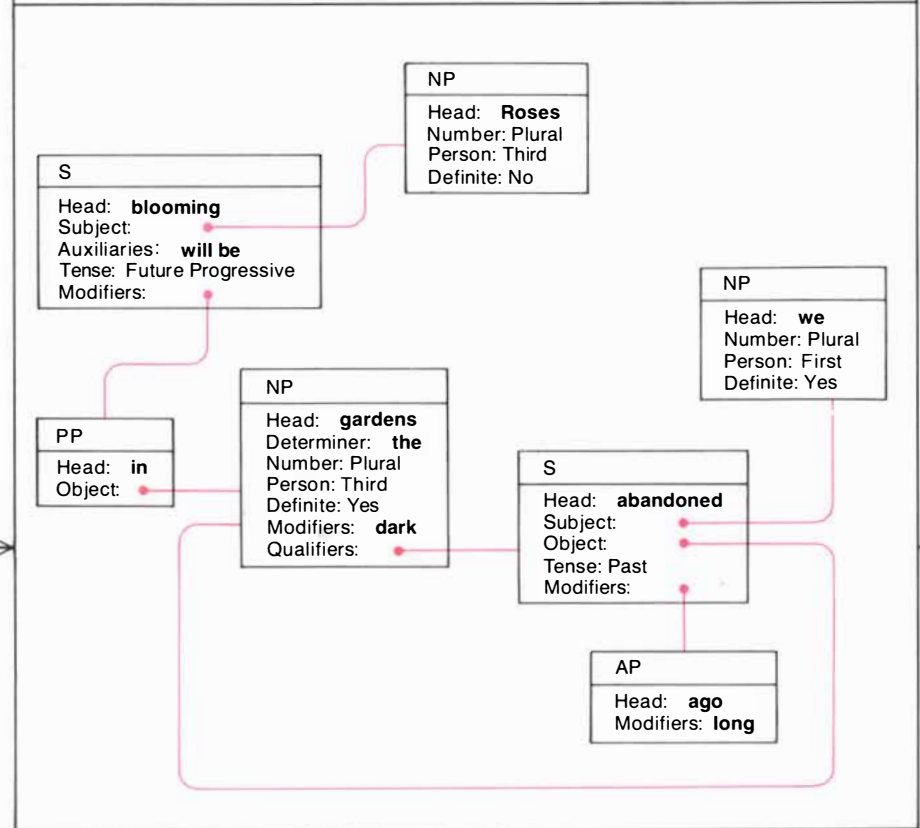
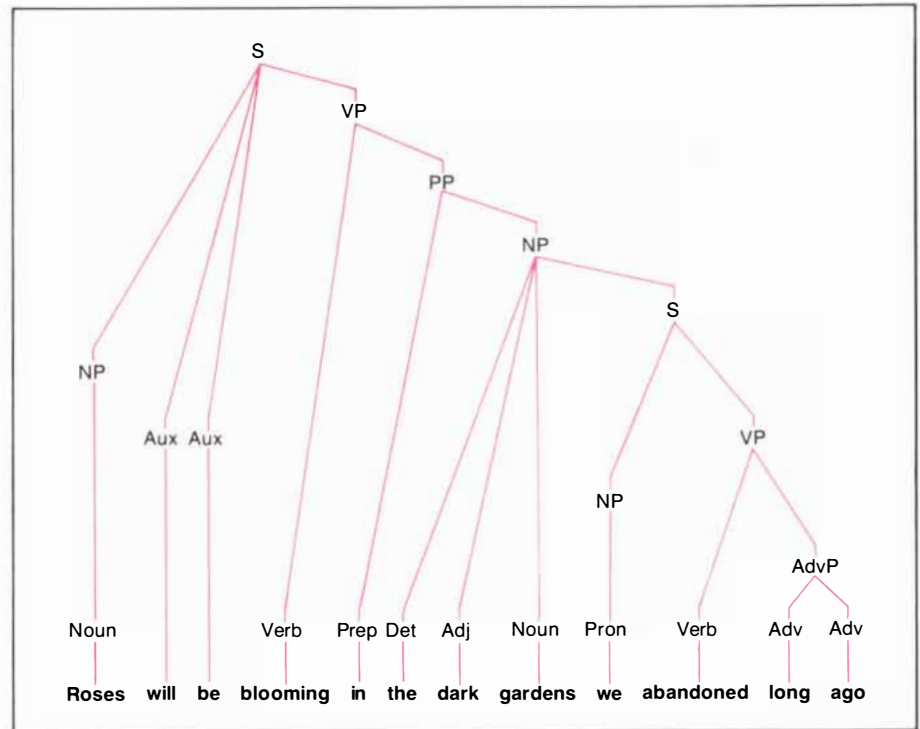
The main advantage of a natural-language front end is that it presents a low initial barrier to potential users. Someone invited to pose a question in English is usually willing to try, and if the computer proves unable to handle the specific form of the question, the user is probably willing to modify the wording until it works. Over time the user will learn the constraints imposed by the system. In contrast, a person who must learn a specialized language in order to formulate a question may well feel that an inordinate amount of work is being demanded.

I want finally to look at a rather new type of system called a coordinator. In brief it replaces standard electronic mail with a process that aids the generation of messages and monitors the progress of the resulting conversations. Coordinators are based on speech-act theory, which asserts that every utterance falls into one of a small number of categories. Some speech acts are statements: "It's raining." Some are expressive: "I'm sorry I stepped on your toe." Some are requests: "Please take her the package" or "What is your name?" Some are commitments: "I'll do it tomorrow." Some

are declarative: "You're fired." (Declaratives differ from statements in that they take effect by virtue of having been said.)

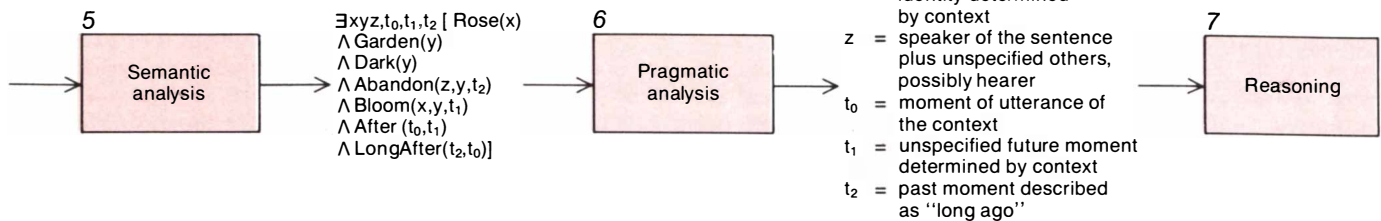
The classification of speech acts is useful because acts in the various categories do not occur at random. Each

speech act has "felicity conditions" under which it is an appropriate thing to say and "conditions of satisfaction" under which it is fulfilled. For example, a request or a commitment carries with it, either implicitly or explicitly, a time by which it should be satisfied. Moreover,



sumably the computer discards numerous incorrect trees. For example, it discards a tree in which "Roses will be blooming in the dark" is construed as a sentence. The deep structure of "Roses will be blooming..." is put in the form of a functional-structure diagram. There the relations between the parts of a sentence become explicit; they are

shown by strings between boxes. Some relations were explicit in the surface structure (for example that "roses" is the subject of "blooming"). Others were not (for example that "gardens" is the object of "abandoned"). The syntactic analysis is supplied to the final stages of the program, which appear in the top illustration on the next page.



ANALYSES CONCLUDE with the conversion of the syntactic structure of "Roses will be blooming..." into a form from which the computer can draw inferences. In this example the conversion is based on predicate calculus; thus the semantic-analysis module of the hypothetical software represents the logical content of "Roses will be blooming..." by symbols that can be translated as "x is a rose and y is a garden and z is dark..." Finally, the pragmatic-analysis module

specifies what is known about the variables x, y, z, t_0, t_1 and t_2 . The variable x , for example, is "quantified": it declares the existence of something instead of identifying a particular object. In other words, the computer takes "roses" as referring to roses in general, not to particular roses. Hence roses is not a "definite" noun. (That decision was made in the course of semantic analysis.) On the other hand, z remains ambiguous because it stands for the ambiguous pronoun "we."

each speech act is part of a conversation that follows a regular pattern. The regularity is crucial for successful communication.

As with every aspect of language, the full understanding of any given speech act is always enmeshed in the unarticulated background expectations of the speaker and the hearer. The speech act "I'll be here tomorrow" might be a prediction or a promise, and "Do you play tennis?" might be a question or an invitation. In spoken conversation intonation and stress play a prominent part in establishing such meaning.

Coordinator systems deal with the speech acts embodied in messages by specifying what needs to be done and when. The system does not itself attempt to analyze the linguistic content of messages. Instead the word-processing software at the sender's end asks the sender to make explicit the speech-act content of each message. A person may write "I'll be happy to get you that report" in the message itself but must add (with a few special keystrokes) that the

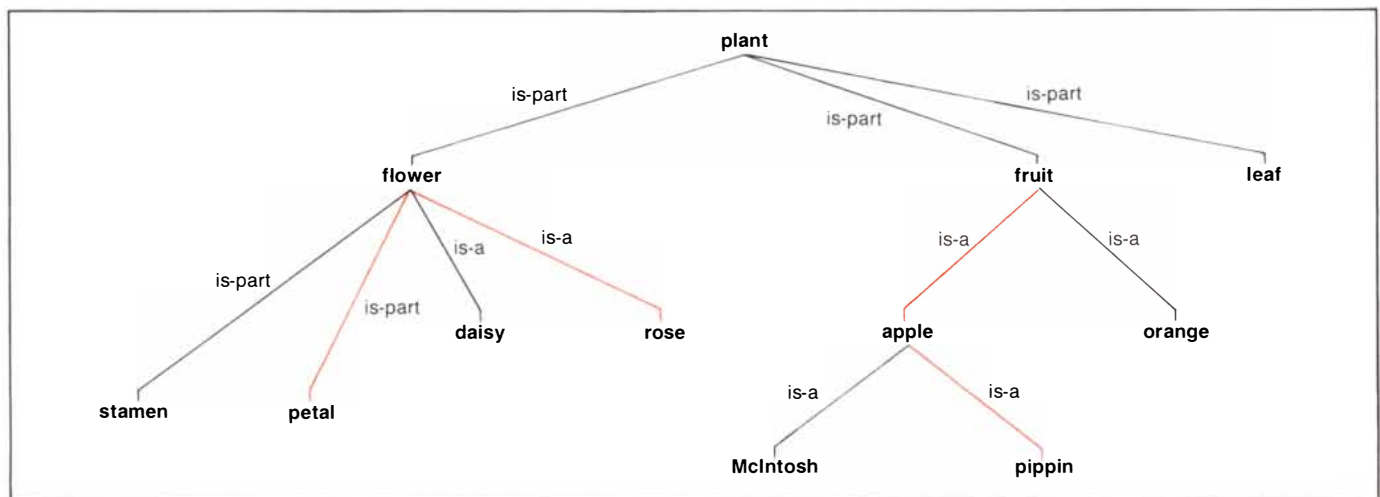
message is an ACCEPT of a particular REQUEST. The computer system can then keep track of messages and their interconnections. In particular the system can monitor the completion of conversations, calling the users' attention to cases in which something immediate is pending or in which an agreed-on time for satisfaction has not been met.

From a broad perspective, coordinators are just one member of a large family of software that gives users a structured medium in which language is augmented by explicit indications of how things fit together. Another type of software in this family provides tools for outlining and cross-indexing documents. Still another type is a computerized bulletin board that enables users to store and receive messages not addressed to a specific receiver. The messages are "posted" with additional structure that indicates their content and helps interested readers to find them.

The most obvious prediction about the future of computer software dealing with language is that the decrease-

ing cost of hardware will make applications that are possible but impractical today available quite widely in the future. Yet software that mimics the full human understanding of language is simply not in prospect. Some specific trends can be noted.

The first is that spoken language will get more emphasis. To be sure, the computerized understanding of spoken language presents all the difficulties of written language and more. Merely separating an utterance into its component words can vex a computer; thus hopes for a "voice typewriter" that types text from dictation are just as dim as hopes for high-quality machine translation and language-understanding. On the other hand, many useful devices do not require the analysis of connected speech. Existing systems that can identify a spoken word or phrase from a fixed vocabulary of a few hundred items will improve the interface between users and machines; the recent emergence of inexpensive integrated-circuit chips that



SEMANTIC NETWORK is a specialized form of stored data that represents logical relations so that certain types of inference can be drawn efficiently by a computer. Here a simple tracing of links in

the network (*color*) has yielded the inference that a pippin is a fruit and that a rose has petals. Facts not readily represented by a network can be represented in other ways, for example by predicate calculus.

process acoustic signals will facilitate the trend. Speech synthesizers that generate understandable utterances (although not in a natural-sounding voice) will also play an increasing role. Improved speech "compression" and encoding techniques will make acoustic messages and acoustic annotation of computer files commonplace.

A second trend in software dealing with language is that constraints on linguistic domain will be handled with increasing care and theoretical analysis. At several points in this article I have noted instances in which computers deal with meaning in an acceptable way because they operate in a limited domain of possible meanings. People using such software quickly recognize that the computer does not understand the full range of language, but the subset available is nonetheless a good basis for communication. Much of the commercial success of future software that deals with language will depend on the discovery of domains in which constraints on what sentences can mean still leave the user a broad range of language.

A third trend lies in the development of systems that combine the natural and the formal. Often it is taken for granted that natural language is the best way for people to communicate with computers. Plans for a "fifth generation" of intelligent computers are based on this proposition. It is not at all evident, however, that the proposition is valid. In some cases even the fullest understanding of natural language is not as expressive as a picture. And in many cases a partial understanding of natural language proves to be less usable than a well-designed formal interface. Consider the work with natural-language front ends. Here natural language promotes the initial acceptance of the system, but after that the users often move toward stylized forms of language they find they can employ with confidence, that is, without worrying about whether or not the machine will interpret their statements correctly.

The most successful current systems facilitate this transition. Some systems (including coordinators) mix the natural and the formal: the user is taught to recognize formal properties of utterances and include them explicitly in messages. Thus the computer handles formal structures, while people handle tasks in which context is important and precise rules cannot be applied. Other systems incorporate a highly structured query system, so that as the user gains experience the artificial forms are seen to save time and trouble. In each case the computer is not assigned the difficult and open-ended tasks of linguistic analysis; it serves instead as a structured linguistic medium. That is perhaps the most useful way the computer will deal with natural language.



Modell

Chivas Regal • 12 Years Old Worldwide • Blended Scotch Whisky • 86 Proof
© 1984 General Wine & Spirits Co., N.Y.

Computer Software for Graphics

No longer the exclusive domain of specialists, interactive computer graphics is fast becoming the standard medium of communication between computers and all kinds of users

by Andries van Dam

Ivan E. Sutherland, a pioneer in the programming of computers to create and manipulate images, once remarked about his favorite subject: "I think of a computer display as a window on Alice's Wonderland in which a programmer can depict either objects that obey well-known natural laws or purely imaginary objects that follow laws he has written into his program. Through computer displays I have landed an airplane on the deck of a moving carrier, observed a nuclear particle hit a potential well, flown in a rocket at nearly the speed of light and watched a computer reveal its innermost workings."

Until recently Sutherland's experience of the seemingly magical powers of interactive computer graphics could be shared by only a handful of workers: mainly scientists and engineers engaged in computer-aided design, data analysis and mathematical modeling. Now the privilege of exploring real and imaginary worlds through the looking glass of the computer is becoming increasingly common. Indeed, graphics is well on its way to being the standard form of communication with computers.

There are a number of reasons for this change. First, dramatic improvements in the price-performance ratio of certain components of computer hardware have made sophisticated graphics terminals and graphics-based personal computers widely affordable. In particular, advances in the design and fabrication of microelectronic circuits have led to a new generation of memory "chips" that offer enormous information-storage capacity at extremely low unit cost. This development has in turn made the technique of raster graphics economically competitive. A raster is the pattern of horizontal scanning lines in a television-type display. In raster-graphics systems each pixel, or picture element, in the raster is represented individually in the computer's memory and hence can be controlled independently by software, giving the programmer maximum flexibility in the creation and manipulation of images.

Meanwhile corresponding improvements in software have greatly extended the range of applications that can be handled pictorially. New software packages for business applications, for example, make it possible to display data in the form of charts and graphs even on inexpensive home computers. In addition standard high-level software packages for graphics are becoming widely available, making it easier for new applications programs to be written and transported from one make of computer to another.

Another factor in the growing popularity of computer graphics is the way computer displays contribute to what has come to be called a "user-friendly" operator-machine interface. This overworked term refers to a philosophy of software design perhaps best exemplified by a set of techniques developed in the 1970's at the Xerox Corporation's Palo Alto Research Center. Computer displays based on this approach (which was influenced by earlier work by Douglas C. Engelbart's group at the Stanford Research Institute) are now available in commercial products ranging from the Xerox Star work station to the Apple Computer, Inc., Macintosh personal computer. A notable feature of this kind of user interface is the "desktop" meta-

phor: the display is divided into separate, possibly overlapping regions called windows, which can be thought of as pieces of paper spread out on a desk. Each window can serve as the display for a different application program; thus one can work simultaneously with both textual and pictorial material, and with the aid of simulated "cut and paste" operations one can compose these different elements into a single document.

The new user-friendly systems are generally based on the WYSIWYG ("What you see is what you get") approach, in which the display resembles as closely as possible what will eventually be printed out (or otherwise recorded in hard-copy form). To design a page of text, for example, no specialized formatting codes (such as *.pp* for "paragraph" or *.s2* for "skip two lines") have to be entered, to be interpreted by a separate "batch formatting" program only after the user has finished editing. Instead margins and indentations are adjusted by manipulating a facsimile of a ruler with markings on it for the various stops, and the text is continuously reformatted to the current settings as it is edited. Because the characters are generated entirely by graphics software, they can be displayed in almost any size or font, spaced either equally or proportionally to their width. Mathemati-

IMAGINARY WORLD of vines and flowers is viewed from the inside in this computer image created by Ned Greene of the New York Institute of Technology. The vines trace the edges of a three-dimensional lattice analogous to the crystal structure of diamond. The image is a frame from an animated sequence in which the viewpoint moves down one of the "corridors" between the vines. The objects in the scene were first defined mathematically as meshes of polygons. The vines were rendered with a "bump mapping" technique that gives the impression of relief by adjusting the shading according to depth information obtained from X-ray images of a plaster cast of real tree bark. The leaves and flower sepals were colored by mapping previously recorded "paintings" onto their mesh representations with a technique known as texture mapping. To produce a smooth gradation of color across the flower petals, colors were assigned to the vertices in the mesh representations of the petals and the vertex colors were then interpolated across the polygonal faces by the rendering program. The appearance of fog was achieved by reducing the contrast as an exponential function of distance from the viewer. "Without fog or some other form of depth cueing," Greene notes, "the scene is practically incomprehensible." There are approximately 1.9 million polygons in the scene, and rendering them took 18 hours on a Digital Equipment Corporation VAX 11/780 minicomputer. Each pixel, or picture element, in the 1,536-by-1,536 raster pattern carries 24 bits of color information. Besides Greene, Jules Bloomenthal, Paul Heckbert and Lance Williams wrote programs used in the project.

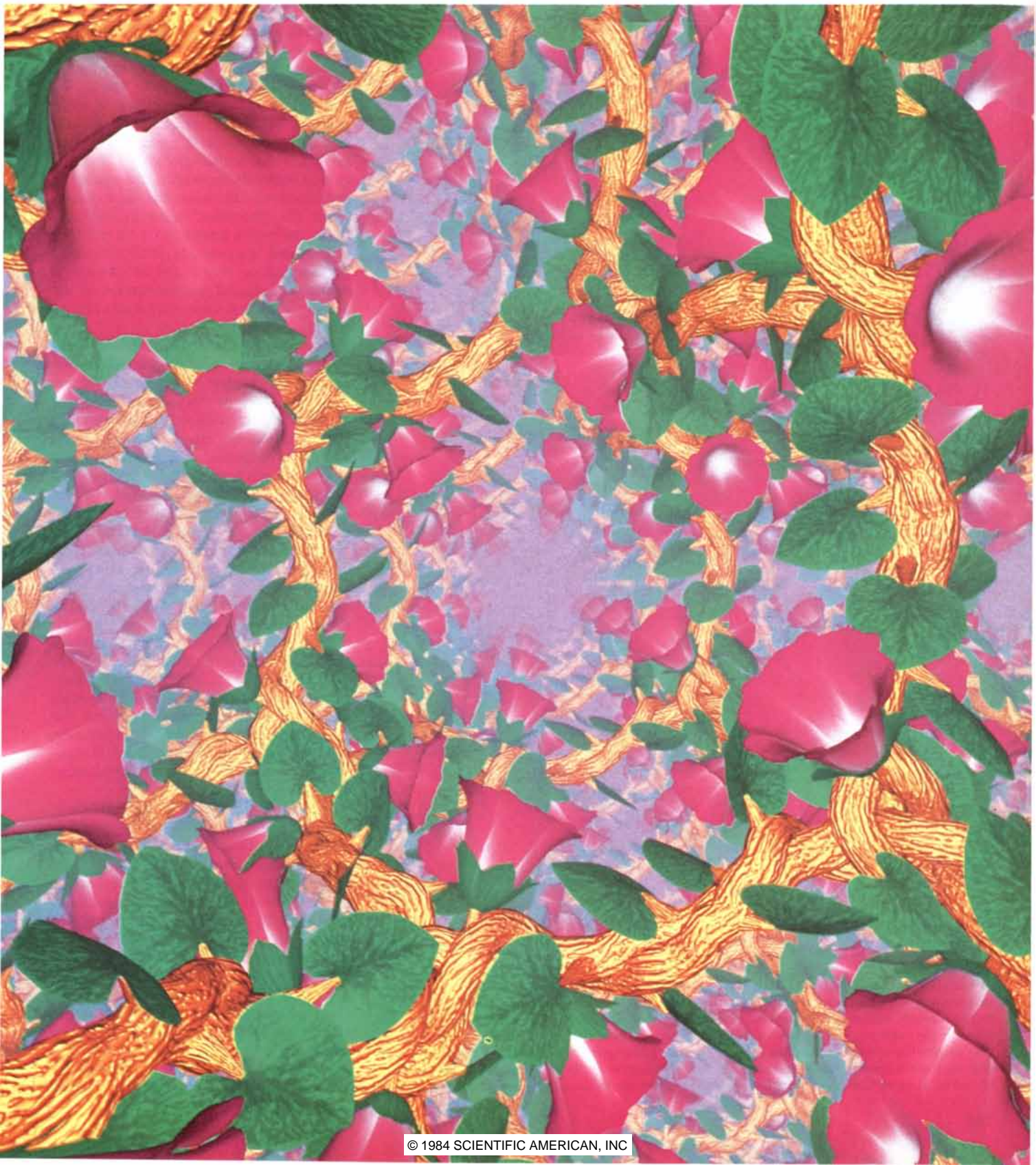
cal symbols, non-Roman alphabets and even Chinese or Japanese ideograms can be handled in much the same way.

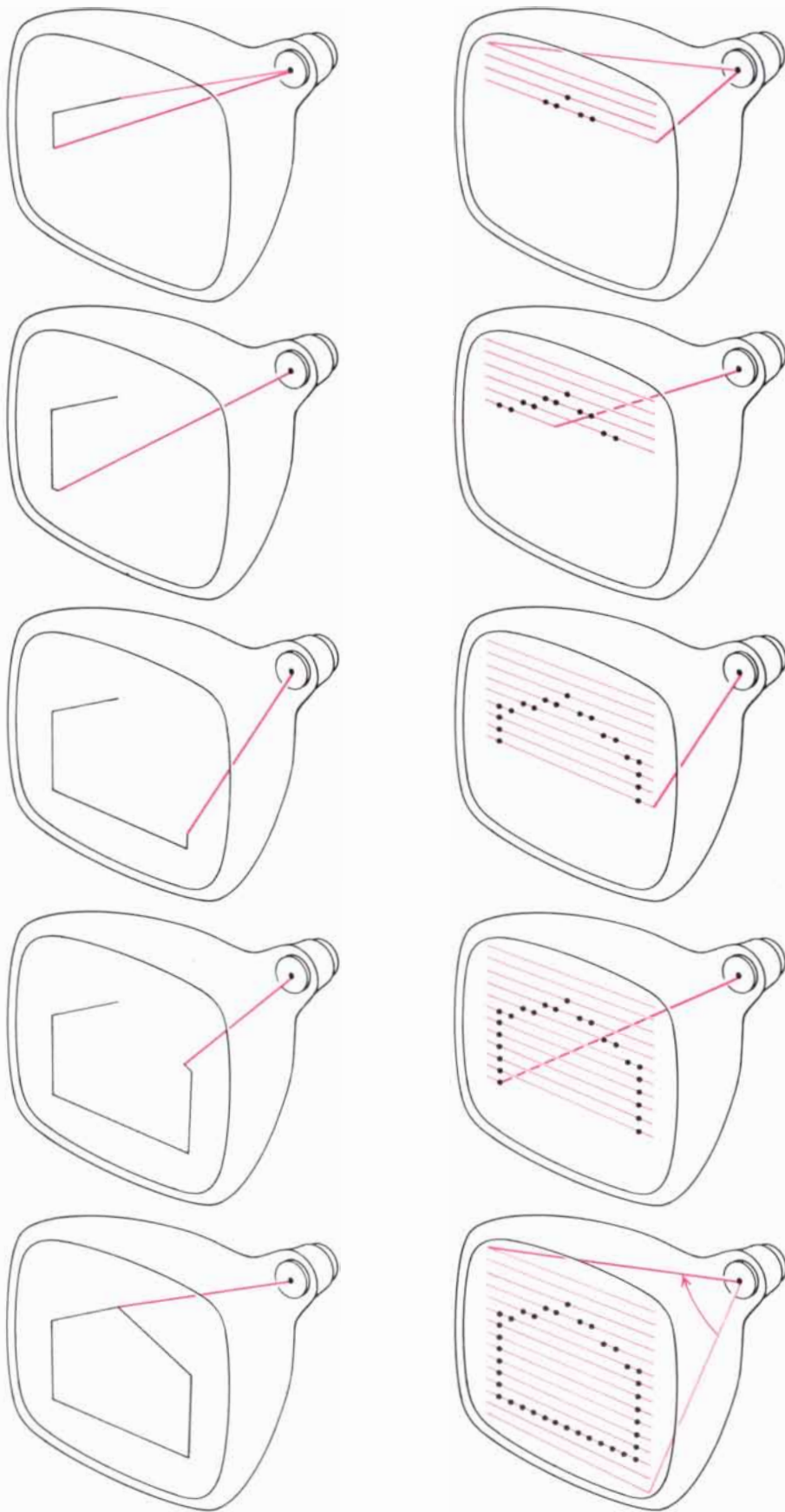
The application programs for such systems rely on a uniform set of conventions to specify commands. For example, instead of typing a series of commands on an alphanumeric keyboard, one can pick and choose from various "menus," or lists of commands, that appear on the display. A command is exe-

cuted by simply pointing to it with the aid of a device such as a light pen or a "mouse" (a mechanism one slides on the desk to move a pointer on the screen). Simple graphic symbols, called icons, represent familiar office items such as file drawers, folders, wastebaskets, calculators and clocks; the functions they symbolize can be selected by pointing to them. It has been found that an interface based on menus and icons is pre-

ferred by most people over a strictly alphanumeric interface because, when these graphic features are properly designed, they seem more natural, are easier to learn and use, require little memorizing and result in fewer mistakes.

Computer graphics has also become commonplace in a variety of other, everyday contexts. Children (and even many adults) are gaining a kind of literacy in graphics by playing arcade





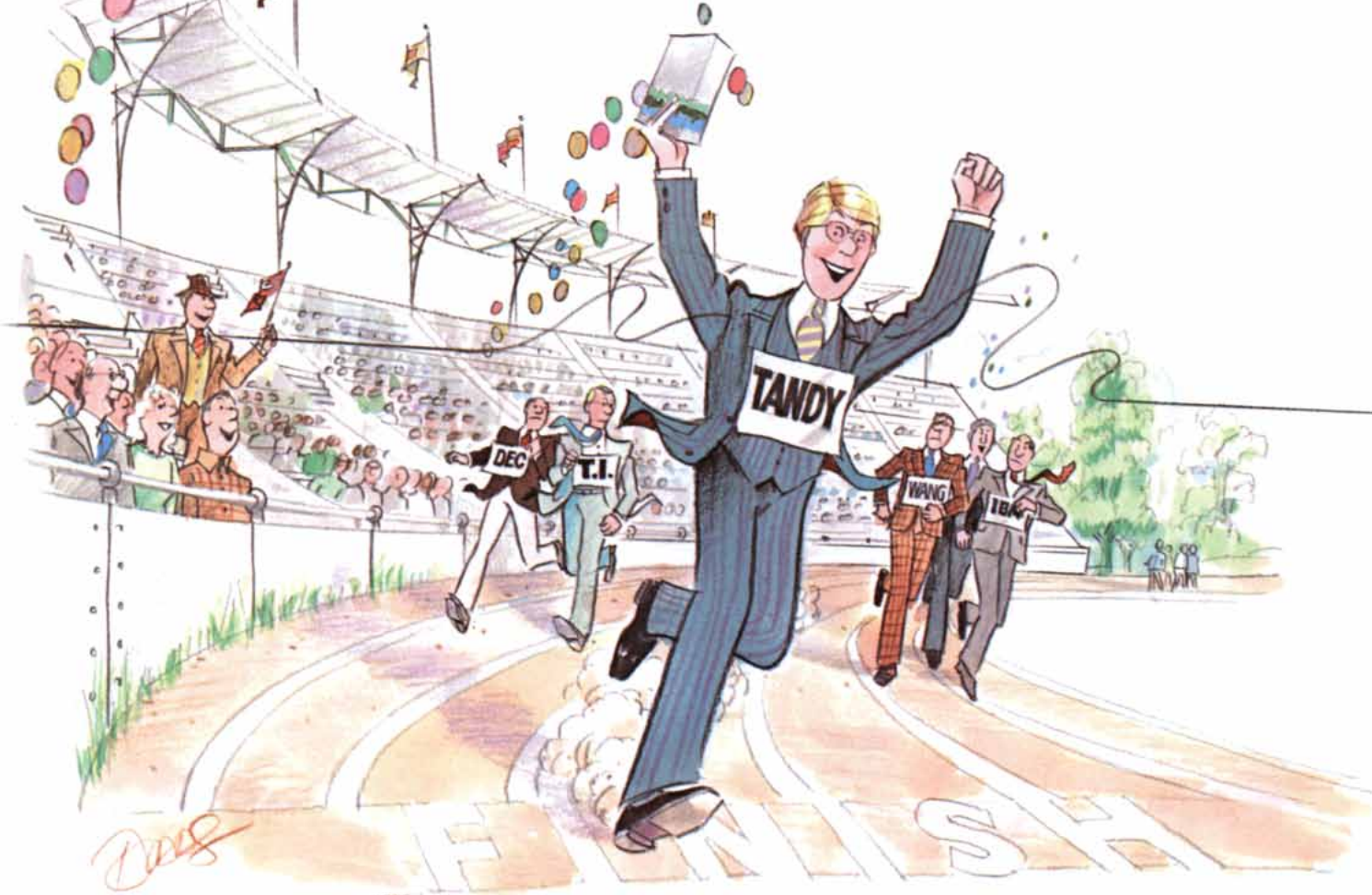
TWO MODES OF OPERATION are available for producing an image on the screen of a cathode-ray tube. In a vector display (*left*) the electron beam is steered continuously between any two points on the screen to create a straight line called a vector. The simple line drawing of a house, for example, is the result of several such operations. In a raster, or television-type, display (*right*) the electron beam traces out a regular raster pattern of horizontal scanning lines, and the beam's intensity is increased at the pixels closest to the straight lines to form the corresponding picture. Raster graphics has recently become the dominant form of computer display.

games and doing educational exercises based on visual effects that often entail a good deal of animation and interaction. In addition artists are now producing eye-catching and sometimes spectacular computer-animated displays for television advertising and for special effects in science-fiction films, taking copious amounts of computer time to produce each highly detailed frame. Image synthesis, currently one of the most rapidly expanding fields of computer graphics, is discussed in greater detail below.

Most interactive graphics displays are based on the technology of the cathode-ray tube (although solid-state flat panels are coming into vogue for some purposes, such as portable computers). The electron beam in a cathode-ray tube strikes a phosphor-coated screen, which emits light with an intensity that depends on the kinetic energy of the electrons. Because the light output from the phosphor fades in milliseconds, the entire image must be redrawn at frequent intervals, typically 30 times per second or more; the redrawing is based on a digital representation of the picture stored in a memory unit called a refresh buffer.

The electron beam is steered to the desired place on the screen in one of two modes: the vector mode or the raster mode. In a vector display the beam can be deflected continuously between any two points in the display's two-dimensional x,y coordinate system to create a crisp, straight line, called a vector. The result of several such operations is a line drawing. Characters are also composed of short vectors. A set of basic display "primitives"—the lines, arcs, characters and other elements of an image—is stored in the refresh buffer in the form of a list of coded commands specifying the endpoint coordinates and other attributes of the primitives, such as their thickness, intensity and color. For display systems with a "real time" three-dimensional viewing capability special hardware is provided to perform the "viewing transformation," an operation that consists in projecting three-dimensional primitives onto the two-dimensional screen.

While the image is being refreshed, either the computer itself or special-purpose hardware can be commanded to assign translation, rotation or scaling values to the endpoints of the vectors or to the viewing-transformation parameters in order to change the picture for the next refresh cycle. These parameters can be specified by an animation program or by the operator, using a mouse, a joystick or dials. The ability to have either the objects on the screen or the viewpoint of the user appear to move smoothly has been found to be very helpful in giving people kinesthetic feedback as they explore the structure of



Lotus 1-2-3™ on the Tandy® 2000 Outruns IBM's PC!

If your job seems like a constant race against the clock, you need Lotus 1-2-3. This versatile "integrated" software package combines spreadsheet analysis with graphics and information management in a single program.

Lotus 1-2-3 can help you get the job done right. But if you want the job done fast, make sure your computer is a Tandy TRS-80 Model 2000.

With a Tandy 2000, you can sort through your Lotus database in less than half the time it takes on an

IBM PC. And if you need to recalculate your spreadsheet, you can do it in less than a third of the time!

A complete two-disk Tandy 2000 system with monitor and 1-2-3 sells for \$3793, and can be leased for only \$130 per month.* For those of you who already own a 256K Tandy 2000, Lotus 1-2-3 can be yours for \$495.

So tonight, jog by your nearby Radio Shack Computer Center and see how a Tandy 2000 lets you race right by IBM. And IBM compatibles, too.



Available at over 1200
Radio Shack Computer Centers and at
participating Radio Shack stores and dealers.

Radio Shack® COMPUTER CENTERS

A DIVISION OF TANDY CORPORATION

**New! 1985 TRS-80 catalog.
Send me a copy.**

Mail to: Radio Shack, Dept. 85-A-112
300 One Tandy Center, Fort Worth, Texas 76102

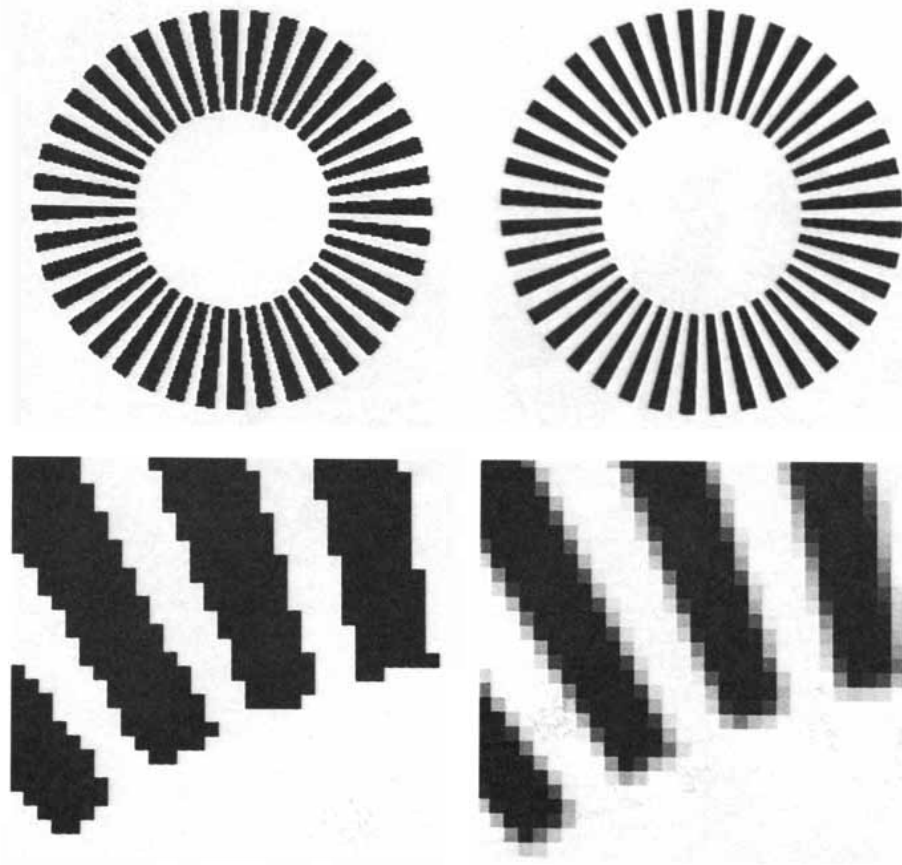
NAME _____
COMPANY _____
ADDRESS _____
CITY _____
STATE _____ ZIP _____
TELEPHONE _____



We Invite Comparison!

In speed, graphics, disk storage and support, the Tandy 2000 offers more than IBM's PC. Come in and see!

*Plus applicable user/sales tax. Prices apply at Radio Shack Computer Centers and participating stores and dealers. Comparisons based on database sorting of 172 records with four fields using a primary and secondary key, and a spreadsheet recalculation of 3 columns by 200 rows. Lotus 1-2-3/Registered TM of Lotus Development Corp. IBM/Registered TM International Business Machines Corp. TI/Registered TM Texas Instruments, Inc. Wang/Registered TM Wang Laboratories, Inc. DEC/Registered TM Digital Equipment Corp.



“JAGGIES,” or jagged edges, appear in a raster display along any lines or edges that are neither horizontal nor vertical, owing to the way such display “primitives” are approximated by discrete sets of closest pixels. This artifact, also known as staircasing, is barely noticeable in the radial pattern at the upper left; it can be seen more clearly in the enlargement at the lower left. One way to minimize the problem in systems in which there are multiple bits per pixel is to vary the intensity of the pixels lying on the boundary in order to blur the edge, as in the pattern at the upper right and in the corresponding enlargement at the lower right. The jagged-edge problem is a form of aliasing; the solution shown here is called anti-aliasing. The diagrams were produced on a computer screen by Paul S. Strauss and James K. Rinzler of Brown University.

an unfamiliar three-dimensional scene.

Vector graphics, which was initially the most common mode of computer display, offers several advantages: it represents display primitives in a way that requires little memory; the primitives are crisply drawn, and the operator can change the image continuously in real time. Its main disadvantage is that it cannot show solid areas; both two- and three-dimensional objects must be represented by “wire frame” diagrams. Furthermore, if there are too many display primitives on the screen for all of them to be redrawn in the time allotted for a single refresh cycle, there are too few cycles and the image flickers.

In a raster display the beam is not deflected in a pattern determined by the image being drawn. Instead, as in the case of a television set, the beam traces out a regular raster pattern. The only control is on the beam’s intensity. In a color display the intensities of three beams—one each for red, green and blue—are controlled individually; each beam strikes its corresponding phosphor dot in a triad of red, green and blue dots for each pixel. The primitives in

a raster display are formed by intensifying the pixels that are closest to the straight line, curve or edge that is defined by the endpoints of the primitive. Solid areas are filled in by intensifying all the interior pixels. Raster displays, because of their fixed deflection pattern, are generally simpler and less expensive than vector displays. On the other hand, raster displays usually call for a much greater memory capacity in the refresh buffer, which must now store an intensity value or a color value amounting to at least one bit for every pixel on the screen. (In this context the refresh buffer is also known as a frame buffer or a bit map.) One advantage of storing the image in the form of individual pixels rather than higher-level primitives is that the former representation is completely independent of the number of primitives specified for display. As a result raster displays avoid the problem of flicker.

A vector display draws lines and edges in a way that is analogous to the way a draftsman uses a ruler or a T square. A raster display, in contrast, relies on an electronic version of the poin-

tillist technique developed by the 19th-century French Impressionist painter Georges Seurat. A discrete sampling technique of this kind may cause individual pixels to be noticeable, and primitives that are neither horizontal nor vertical have jagged edges. This artifact, sometimes referred to as staircasing or the “jaggies,” is a form of aliasing, a common problem in signal processing. It can be minimized by increasing the resolution of the display or by varying the intensity of the pixels lying on the boundary in order to blur the line or the edge. (The latter process is sometimes referred to as anti-aliasing.) A new technique that simulates higher resolution and avoids blurring is called pixel phasing by its developer, the Megatek Corporation. In this approach each pixel can be slightly repositioned by shifting it by a quarter, a half or three-quarters of a pixel diameter horizontally or vertically; the size of the pixel can also be adjusted to help fill the gaps.

The ability to specify each pixel’s intensity or color value independently in raster systems is particularly important for the creation of detailed character fonts or icons. Typically a font is defined as a set of small pixel arrays, one for each character or icon. As a character or icon is needed, its array is copied from the computer’s main memory or from a special part of the frame buffer to the part holding the representation of the characters on the screen. On the other hand, because of the large number of pixels that must be updated whenever a significant part of the image is moved or deleted, making changes is usually much slower on raster displays than it is on vector displays, where only encoded primitives have to be changed. Modern raster systems enable the programmer to quickly copy and move rectangular blocks in the frame buffer by means of special operations that facilitate the “scrolling” of text, the rearrangement of windows and the creation of simple animated sequences. Such systems may also provide a display-list representation, with rapid “rasterization” from encoded primitive form to pixel form for the frame buffer.

The function of high-level support software is to insulate the programmer from these kinds of low-level hardware details so that he can concentrate on matters pertaining directly to the application. In the early days of computer graphics this was not possible, because graphics applications were programmed at the assembly-language level. Efficiency took precedence over ease of programming, and the transportability of programs from one make of computer to another was hardly even a consideration. It was not until the late 1960’s and early 1970’s that the drive began to write graphics programs at a higher level and



The effort was as important as the end.

Again, this summer, the best swimmers and divers gathered to compete for the most coveted awards in amateur sports. And though many did win medals, all are worthy of praise. For in their effort to achieve that goal can be found their most noble triumph — the desire to be the very best they can be.



Phillips Petroleum has sponsored United States Swimming since 1973, and United States Diving since 1979, helping with operating costs of national and international competitions leading to world championships. We continue to support them. Because their ideals touch the lives of everyone.



**No one knows more
about putting fiber optics to
work than New York Telephone.**

**Our advanced lightwave technology
doesn't just promise virtually
error-free transmission.
It guarantees it.**



The New York Telephone network. Taking the lead in the information age.

Fiber optics may be something new to most people, but not to us. We've been working with fiber optics technology for years.

As a matter of fact, fiber optics, the transmission medium of the future, is in place and working at New York Telephone today.

Right now, New York Telephone has 32,000 fiber miles of lightwave cables serving the telecommunications needs of many of America's largest financial and industrial companies. And we have fiber optics projects already built or under construction in New York, Long Island, Westchester, Albany, Syracuse, Buffalo and other locations around the state.

The special things fiber optics can do for you.

Fiber optics can transmit voice, data and video messages from one point to another faster, more reliably and less expensively than any other transmission system available.

Possibly the most important feature of our fiber optics network is its ability to transmit information practically error free. A major reason that reliability is made possible is that fiber optics do not have to electronically repeat transmissions as many times as conventional facilities do. Other reasons are that they're immune to interference from moisture and electrical currents and they transmit data in a digital format—the language that computers "speak."

And, since fiber optics utilize laser light sources, they can transmit voice, data and video at speeds of up to 405 million bits of information per second.

A fast, efficient network.

But fiber optics are only one part of our remarkable network.

New York Telephone's network consists of a vast combination of electrical equipment, microchips, processors and memory units—all woven together to work with fantastic speed, efficiency and precision.

And we're constantly working with new technologies to modernize and upgrade that network.

As the communications needs of New Yorkers and businesses such as yours grow in scope and sophistication, New York Telephone will have the solutions. We are anticipating the future. We have made the necessary capital commitments. We have the people, the ingenuity, the imagination and the technology. As we have for nearly a century, we will continue to provide you with high-quality communications service in the future.

**New York lives on
New York Telephone.**



New York Telephone

A NYNEX Company



In the Air Force no idea is too far out.

Air Force engineers are designing tomorrow's technology today. It takes imagination to dream new dreams and skills to bring those dreams to life.

If you're an electrical or aerospace engineer, or plan to be, the Air Force gives you a chance to push your skills to the limit and learn new ones. And while you're growing,

you'll be helping your country grow stronger, too.

For more detailed information, call us toll-free at 1-800-423-USAF (in Calif. 1-800-232-USAF). Better yet, send your resume to HRS/RSAANE, Randolph AFB, TX 78150. We're waiting for your ideas.

AIM HIGH AIR FORCE

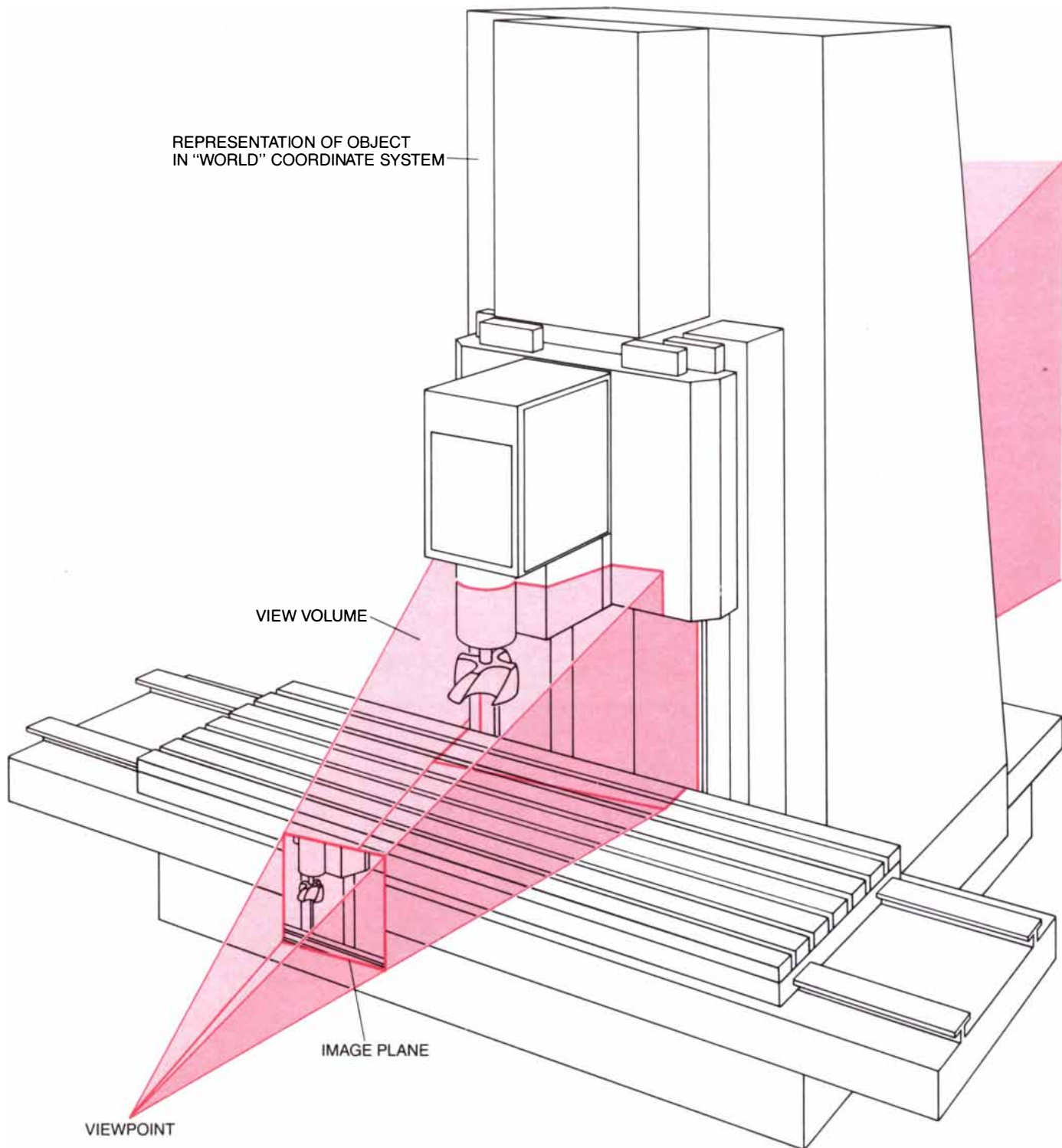
to make them independent of any particular computer system.

The first graphics software of the new era was designed in imitation of the input-output strategy of high-level programming languages. Idealized "virtual" devices corresponding to real interactive devices were generated by means of low-level "device driver" programs that handled both the tricky graphics

hardware and the equally tricky input-output communications with the central processing unit. Each virtual display had a square virtual screen designed to coincide with the largest square that could fit on the actual display surface or the plotter. The same unit coordinate system was used to address the virtual screen regardless of the real screen's dimensions. Each virtual display could

also have virtual input devices. Input devices not available on a particular console could be simulated by means of the devices present; in this way one could create, for example, a virtual keyboard, virtual dials or even a virtual mouse.

Most graphics programs at the time were developed for applications in computer-aided design and data visualization. The programs ran on vector dis-



VIEWING TRANSFORMATION is the operation whereby a stored representation of an object (in this case an idealized milling machine) is projected from a three-dimensional "world" coordinate system onto the two-dimensional coordinate system of a screen, represented by the image plane in this schematic diagram. Mimicking the opera-

tion of a camera, the software "clips" the parts of the object that are outside the view volume and then projects only the parts that are inside the view volume onto the screen. For a three-dimensional perspective projection the clipping boundary is typically a pyramid. In this case hidden edges were eliminated before the projection stage.

play systems and plotted pictures derived from an application data base, known as the application model. The graphics software provided the application programmer with a two- or three-dimensional "world" coordinate system equally suited to handling angstrom units, centimeters, miles or light-years.

The world coordinate system enabled the programmer to abstract the definition of primitives to a level even further removed from the hardware than that of the virtual screen's standard coordinate system. The software also handled the entire viewing-transformation operation, specifying the area currently in

view in the world coordinate system and the region of the virtual screen on which it was to appear. The viewing-transformation software "clipped" primitives that were outside the viewing area and projected only those primitives that were inside the viewing area onto the actual screen. In two dimensions the clipping boundary was a rectangle, whereas in three dimensions it could be either a rectangular solid (for a parallel projection) or a pyramid (for a perspective projection).

In effect, the type of graphics software that was first developed a decade or more ago can be described by the "synthetic camera" metaphor: the application program constructs a world consisting of objects such as flow-chart symbols, circuit elements or atoms in the application-dependent model, including all the appropriate attributes and parameters, and then extracts geometric information from it to pass on to the graphics software. The graphics software, which is typically under the viewer's control, then takes a snapshot of the specified primitives in the viewer's world from the specified viewpoint and posts the snapshot on the screen. Thus modeling is the responsibility of the application program, and viewing some part of the model is the responsibility of the synthetic-camera software. The application itself typically consists of two subsystems: a graphics editor that enables the viewer to create and manipulate the application model and its visual representation, and an independent set of postprocessing packages that analyze the completed application model. In computer-aided design these packages include provisions for simulating and testing the design and subsequently specifying manufacturing data for fabrication and construction, often by numerically controlled machine tools.

Two standard graphics packages are now becoming available for all categories of commercial displays: the three-dimensional Core Graphics System sponsored by the Association for Computing Machinery and the two-dimensional Graphical Kernel System adopted by the International Standards Organization. Derived from a common ancestor, they are both essentially synthetic-camera packages. As it happens, they were largely designed before raster graphics became the dominant form of computer display. Although they can handle raster primitives such as pixel arrays and filled polygons, they still operate in the world coordinate system, with user-defined objects. For many simple applications in raster graphics the application program cannot take enough advantage of the facilities of the package to justify the considerable computational "overhead" needed to handle more complex applications. Moreover, a pro-

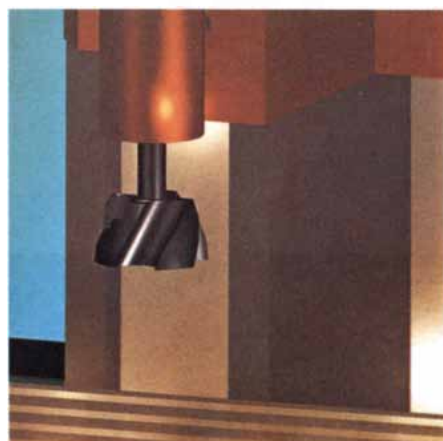
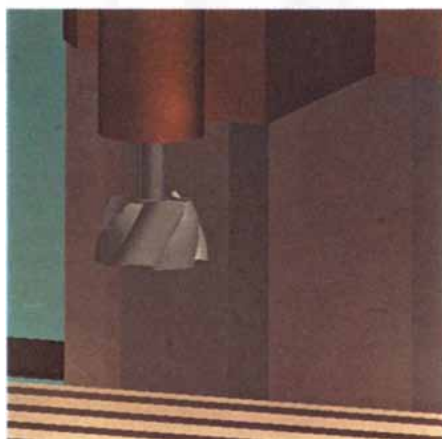
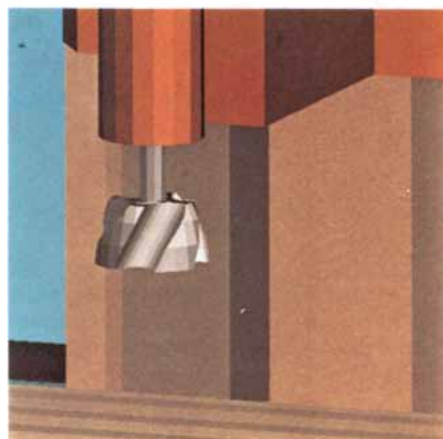
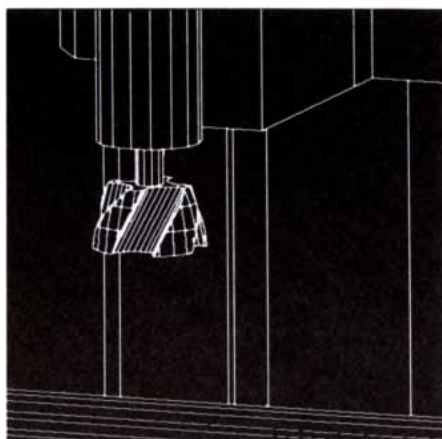
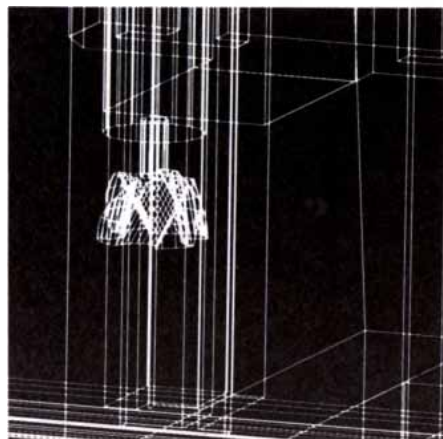
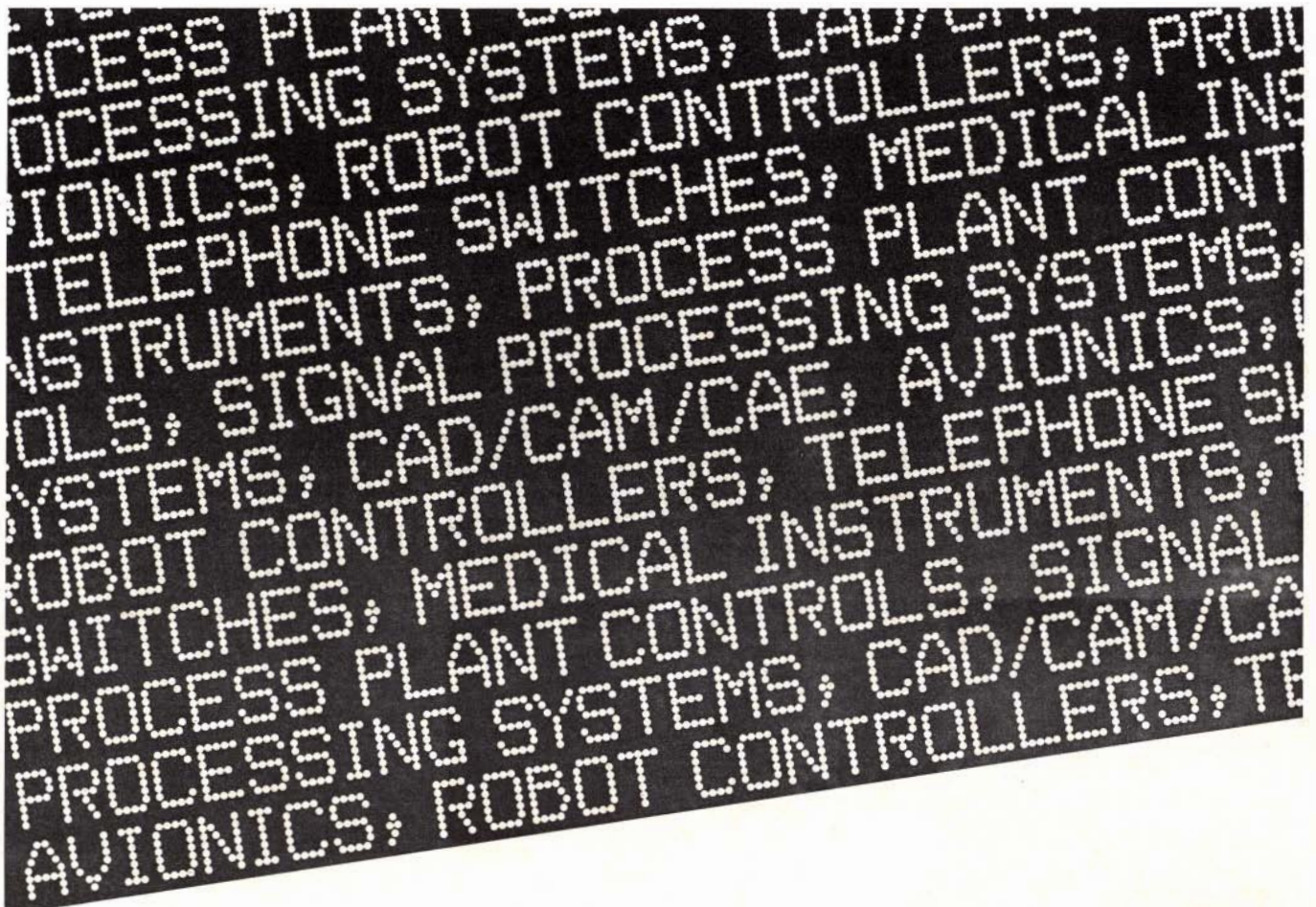


IMAGE SYNTHESIS BY COMPUTER can be thought of as proceeding through a sequence of steps, although in fact they are often interlaced in a single program. In this demonstration of the process the object (a close-up of the milling machine) is defined as a mesh of polygons and is displayed first in the form of a "wire frame" diagram (1). Hidden edges are then removed (2). In the next step shading (in this case color shading) is applied individually to the polygons as a function of the angle of the polygon with respect to the light sources and of its surface properties; the result is a picture with an unnatural, faceted appearance (3). The discontinuities at the shared edges between adjacent polygons can be smoothed by Gouraud shading (4), and specular (mirrorlike) highlights can be added by Phong shading (5). In the final step anti-aliasing smooths out the jaggies (6). The images were made by Rinzler and Strauss in collaboration with Roger L. Gould, Richard L. Hagy, David H. Laidlaw and Gerald I. Weil, all students at Brown.



OUR MTOS REAL-TIME OPERATING SYSTEMS ARE THE NUMBER ONE CHOICE FOR CONTROL APPLICATIONS WORLDWIDE.

Since 1973, IPI's MTOS family has been used in more control applications than any other real-time operating system.

From radar to robotics. Avionics to process control. The list of potential MTOS applications is limited only by your imagination.

Available for 8-bit and 16-bit Intel and Motorola microprocessors.

MTOS is fast. It's simple to use. And all our systems are conceptually compatible. So once you've learned to use one system, you can use them all.

MTOS is also rich in coordination and other critical services. And unlike some operating systems, it's written in assembly language; the result is speed and compactness.

MTOS offers multiprocessor support.

MTOS is the only system that will control several microprocessors on a common bus. For advanced applications such as vision systems and signal processing, this unique capability is indispensable.

Our customer list is a who's who in high technology.

MTOS is sold in the United States and in twenty countries overseas. Our customers include many Fortune 500 companies and some of the most prestigious names in academia.

Specially configured versions of MTOS are available for various hardware, such as the IBM® PC. Other versions are

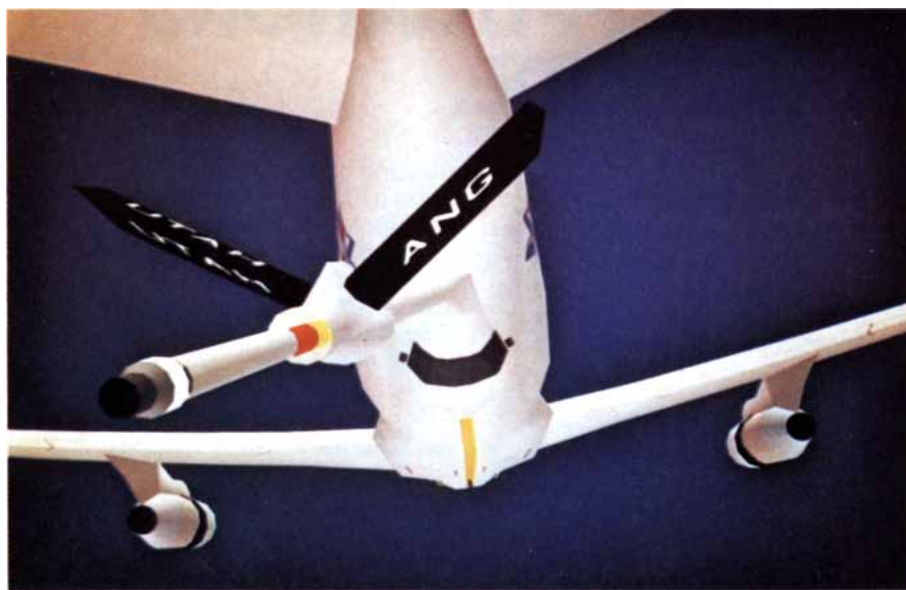
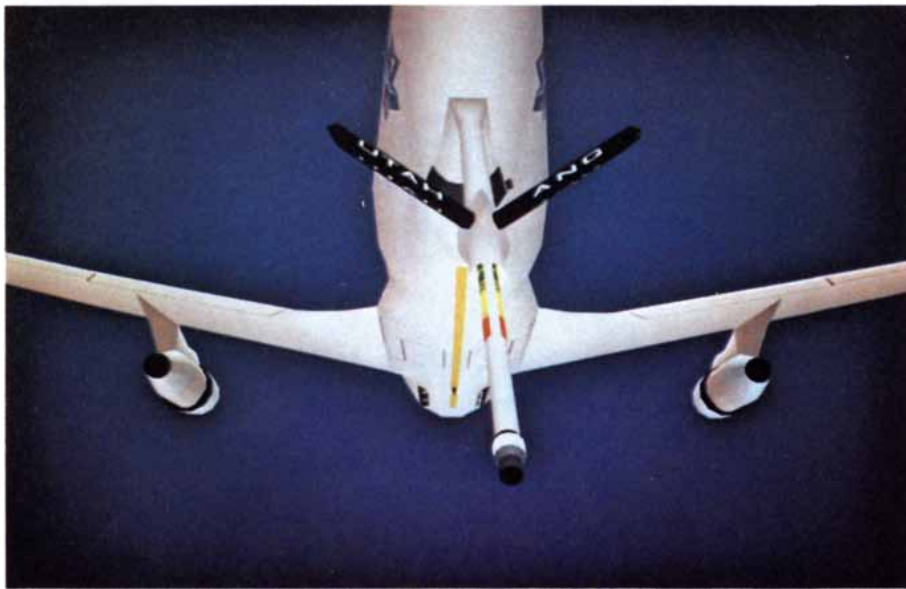
available in source form, and are sold under a liberal licensing policy.

To learn more about real-time operating systems in general, and MTOS in particular, call or write for our free booklet "On Operating Systems." Industrial Programming, Inc., 100 Jericho Quadrangle, Jericho, NY 11753. Telephone: 800-228-MTOS (in New York State call 516-938-6600). Telex 429808 (ITT).

IBM is a registered trademark of International Business Machines Corporation.

iipi Industrial Programming Inc.

The standard-setter in operating system software.



HIGH-QUALITY, REAL-TIME ANIMATION is achieved in certain special-purpose applications, as seen in this sequence of frames selected from an interactive flight-simulation system for training pilots in aerial-refueling procedures. The ultrahigh-performance system, which is capable of generating 50 frames per second, is a product of Evans & Sutherland.

gram relying on a graphics package may not be able to take full advantage of several powerful new hardware capabilities associated with raster-graphics work stations and personal computers.

Programs that do not adhere to the modeling-followed-by-viewing scheme of the synthetic-camera metaphor include the "painting" programs now becoming popular on raster-graphics systems. The objects manipulated in such programs are not world coordinate objects but rather the individual pixels, and the package must enable the viewer to recolor, move or even logically combine arbitrary regions in the frame buffer. Painting by manipulating pixels in the frame buffer is analogous to making a photograph by altering areas on the emulsion directly rather than by exposing the film through a camera aimed at a real scene. Synthetic-camera packages are unsuitable for such low-level, device-dependent operations.

The situation is further complicated by the need to manage multiple windows on the bit-mapped screen. The existing graphics packages have no way to handle multiple application programs. Most raster-graphics work stations are therefore provided with a "window manager," a low-level part of the system software that keeps track of which program runs in which window and where the window is defined on the screen. One window might run a painting program, another a word-processing program and a third an application program based on a standard graphics package. The window manager must handle such problems as moving windows, covering and uncovering overlapping windows, scaling or clipping primitives to fit in the currently visible part of a window and finally rasterizing the visible primitives for display on the screen. As yet there is no commonly accepted design for such window managers.

For the foreseeable future a number of graphics standards designed by different communities of users will coexist. Examples include the Initial Graphics Exchange Specification, an engineering-drawing standard for computer-aided design, and the North American Presentation Level Protocol Syntax, for displays of text and graphics on television. All the standards share the common purpose of defining primitives, their attributes and their groupings in named collections for selective manipulation as a group. Eventually these diverse standards should be brought together.

Most traditional applications of computer graphics have been two-dimensional. Lately, however, there has been increasing commercial interest in three-dimensional applications, arising from the significant progress made in the past decade on the twin problems of modeling three-dimensional scenes and

displaying them as realistically as possible. In flight simulators for training pilots, for example, the emphasis is on responding to input from both the pilot and the instructor. To give the impression of smooth motion the simulator must present a fairly realistic picture of a dynamically changing world at a rate of at least 30 frames per second. In contrast to this real-time animation, pictures for the advertising and entertainment industries are computed off-line, often for hours, in order to get maximum realism or visual impact. In computer-aided design there is now increased emphasis on creating wire-frame diagrams interactively and then promptly displaying them in a fully rendered version. The newest hardware even makes possible the interactive creation of polygonal "solid" objects.

Two-dimensional objects are modeled by such primitives as lines defined by two endpoints, polygons defined by a list of vertexes and possibly a fill pattern, circles defined by a center, a radius and possibly a fill pattern, and polynomial curves defined by their coefficients. In three dimensions the corresponding primitives are defined by adding the z coordinate. Primitives that exist only in three dimensions can also be defined; they include polyhedrons, pyramids, spheres, cylinders and surfaces described by certain polynomial functions.

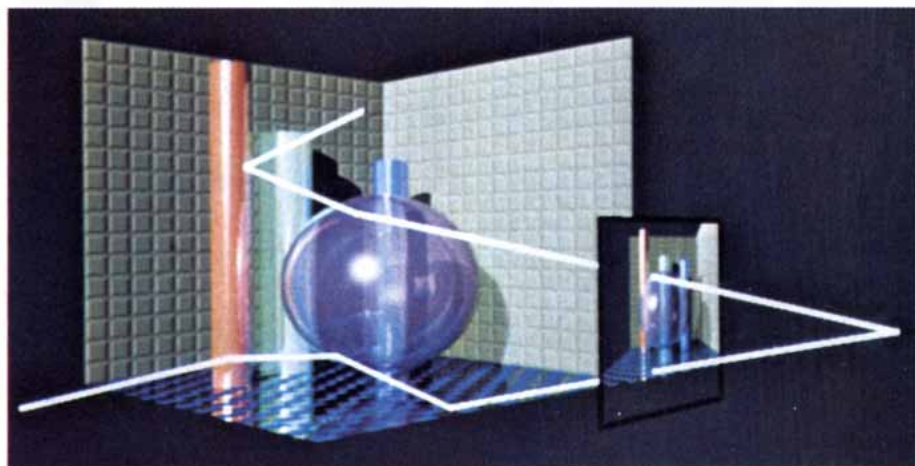
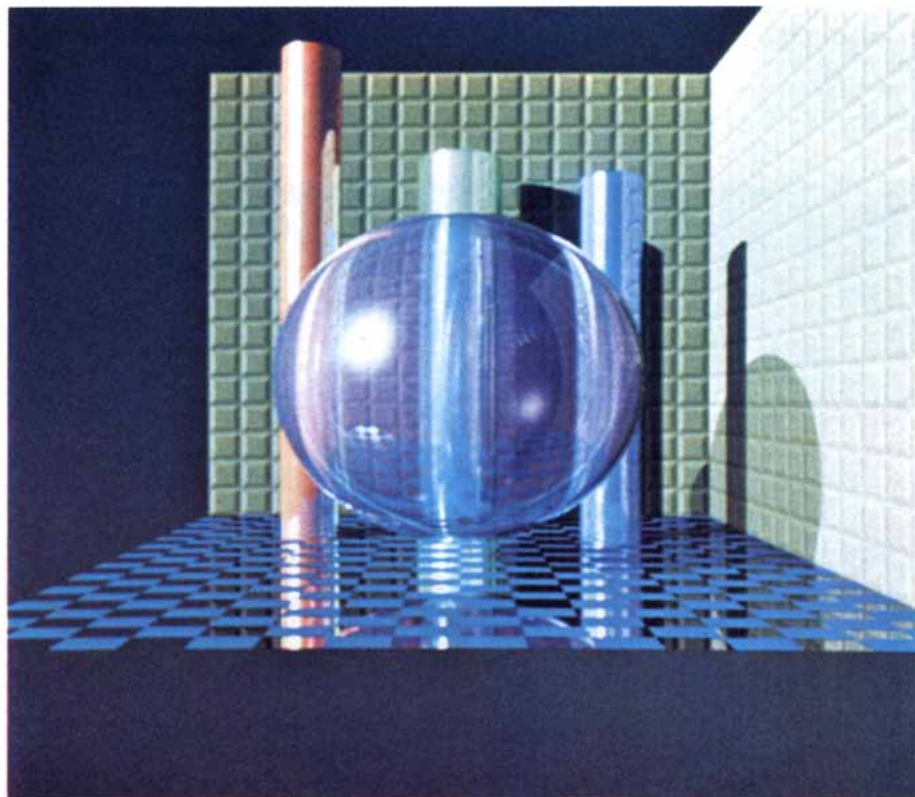
Solid-modeling systems for creating three-dimensional objects rely on either interactive or off-line specification of parameters. Off-line specification can be done with data files created by another program or with a text editor. Alternatively a procedural description, such as the one used to generate fractal curves and landscapes, can do the job. Furthermore, an object can be modeled directly as a solid or indirectly as a volume bounded by its surface.

In systems based on constructive solid geometry objects are modeled directly by solid primitives such as blocks, cylinders and spheres. The primitives can be combined by means of three-dimensional set operations such as union (joining two objects), intersection (taking a common subset) and difference (taking all of the first object except for those parts it has in common with the second object). Indirect representation is done in boundary-representation systems that may also provide set operators but that define an object as bounded by polygonal facets, cylindrical facets or even surface patches defined by polynomial functions. Such "free form" surface definition with curved patches has become important to aerospace and automobile companies for sculpting the bodies of their vehicles.

An object with rotational symmetry can also be described by a surface of revolution; a vase or a bottle is defined

by its generator (the silhouette curve) and an axis of revolution. Analogous to such a rotational sweep is the translational sweep: here a face of arbitrary complexity, including holes, is translated along a space curve to create a volume. An idealized gear can be made by first defining a quarter section of a face, using symmetry operations to complete the face and then sweeping the face along a short straight path to define the cylindrical form of the solid gear.

Many other mathematical techniques are useful in defining classes of objects, and hybrid systems incorporate a variety of techniques. The special case of making objects interactively presents an additional problem with these methods in that the user is forced to look at a two-dimensional projection of a three-dimensional scene in which depth is difficult to assess. Among the techniques that can give the user some feedback as the specification process proceeds are



RAY-TRACING TECHNIQUE relies on an extremely time-consuming algorithm to compute the reflection and refraction of light from the surfaces of imaginary objects. Basically the computer program traces individual light rays, starting at the viewpoint and passing backward through each pixel in the image plane until the ray hits a surface. The reflected ray is then continued to see if it could have come from a light source, either directly or after reflection from another object. For a transparent surface a second, refracted ray must also be traced. In this demonstration of the technique, produced by Lee Westover and Turner Whitted of the University of North Carolina and Numerical Design Ltd., a ray-traced image is shown at the top and the technique by which the image was made is shown in the computer-generated diagram at the bottom. White lines trace two reflected rays; refracted components of the rays are omitted.

multiple views (such as the standard orthographic projections of front, side and top, as well as a three-dimensional perspective view), drawing in a plane of constant x , y or z coordinates and aids such as dynamically updated dimension lines and two- or three-dimensional grids with tick marks.

When the objects in the scene have been defined, the next phase is to pass the object description to the image-synthesis programs for rendering. Current image-synthesis algorithms work either with polygonal descriptions or with polynomial or other higher-order definitions of mathematical surfaces. It is common to reduce higher-level definitions to a simpler "piecewise" approximation with a mesh of small polygons

prior to rendering. The rendering process can be idealized as a sequence of steps, but the steps are often interlaced in an actual program. All of them are basically adaptations of the fundamental laws of optics.

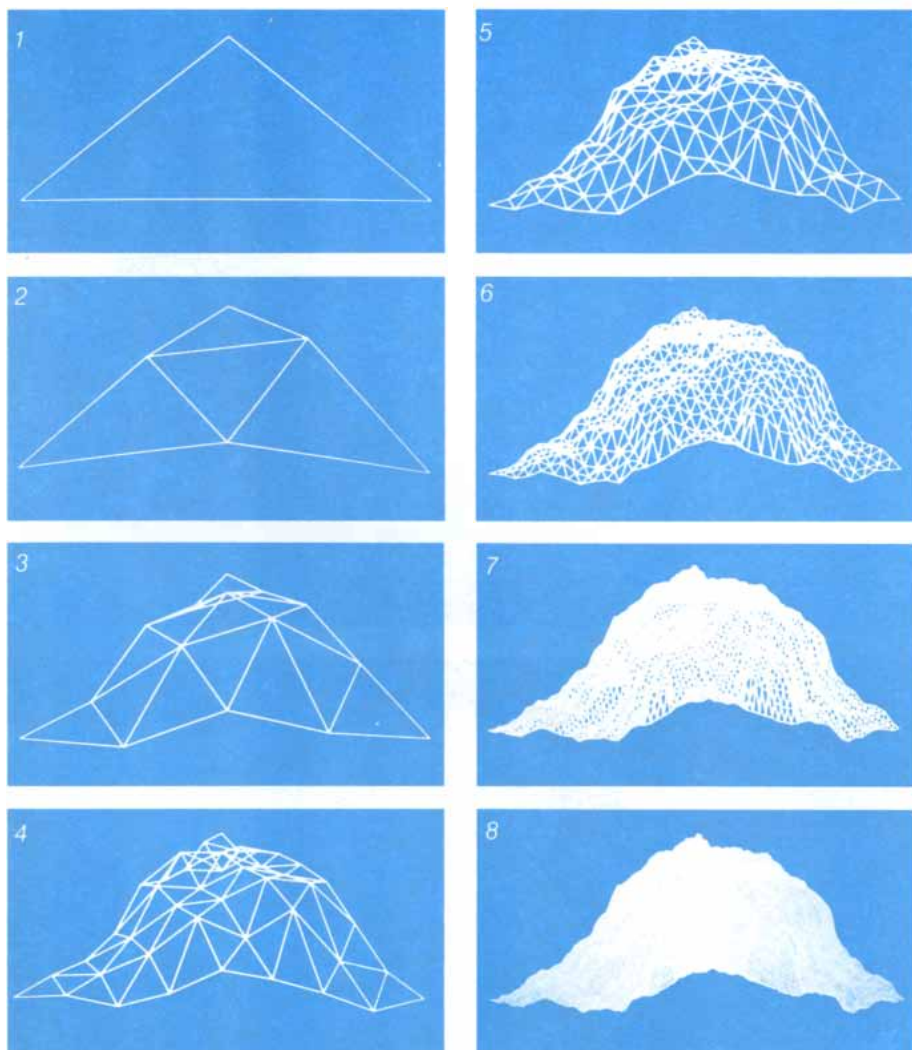
The first step is the elimination of hidden surfaces, that is, surfaces or parts of surfaces that are not visible from the point of view of the synthetic camera. This category includes both surfaces on the far side of objects and those obscured by other objects closer to the viewpoint. Various techniques for the elimination of hidden surfaces can be implemented in hardware. The algorithms typically assume that the screen lies in the $z = 0$ projection plane for the scene that lies behind it. For example,

the z -buffer algorithm maintains a separate buffer of z values, one value for each pixel. The z value of a pixel records the depth of the corresponding point on the nearest polygon encountered so far that projects on the pixel. When a new polygon is transformed, clipped and projected onto the $z = 0$ plane, the z values of its component pixels are compared one at a time with those stored in the z buffer. Only if the z value of a pixel in the current polygon is smaller (signifying that the polygon is closer to the screen at this pixel than any polygon previously encountered) is the current pixel "in view" and stored in both the refresh buffer and the z buffer. It actually becomes visible only if it has not been replaced by the time the last polygon is processed.

Unlike the z -buffer algorithm, which takes polygons in any order, the "painter's algorithm" first orders them from back to front. In the event that pairs of polygons cannot be simply ordered, they are subdivided until their pieces can be. They are then projected and "painted" in the frame buffer in back-to-front order so that the polygons closer to the viewpoint properly eclipse more distant ones, without the need for further computation.

After the visible surfaces have been computed the next step is to calculate the shading for each one. The shading rule must take into account both the properties of the surface (its color, texture and reflectance) and the relative location, orientation and properties of the light sources and other surfaces. Illumination models for light sources can take into account ambient light, point sources (such as the sun or an incandescent bulb) or distributed sources (such as a window or a fluorescent tubes).

Ambient light is most easily modeled by adding a constant amount of light intensity to all surfaces, but naturally that strategy affords no way of differentiating among surfaces. The reflection of point sources of light by matte, or diffuse, surfaces (those that scatter light equally in all directions) is modeled by Lambert's cosine law, which states that intensity varies as the cosine of the angle between the direction of the light source and a vector perpendicular to the surface, called the surface normal. The brightest illumination appears when the surface is perpendicular to the light source. For shinier surfaces that yield specular, or mirrorlike, highlights, such as brightly polished wood or metal, the amount of light reflected depends on both the angle of the light source and the angle of the viewpoint with respect to the surface normal. The surface acts like a mirror in that it reflects most of the light only when the angles are almost equal (that is, when the viewpoint and the light source are arranged symmetri-



SIMPLE TECHNIQUE for generating a realistic image of a mountain is loosely based on the concepts of fractal geometry originally formulated by Benoit B. Mandelbrot of the IBM Thomas J. Watson Research Center. This demonstration of the technique is reproduced through the courtesy of Lucasfilm Ltd. Starting with the single triangle shown in step 1, the computer program generates step 2 from it by the following procedure. First, break each side of the triangle at its midpoint. Second, displace each midpoint by a distance proportional to the length of the corresponding side. (The factor of proportionality can be generated at random or taken from a table of, say, 100 well-dispersed random numbers.) Third, connect the three new points to one another to form four new triangles. Step 3 is generated from step 2 by applying the same procedure in turn to each of the four new triangles, generating 16 triangles, to each of which the procedure is applied again in step 4, and so on. Although the subdivision algorithm is simple, it can yield a very complex polygonal surface. The mountainlike surface in step 8 can later be rendered by standard computer-graphics techniques to produce a finished landscape.

cally with respect to the surface normal). As the angles become more unequal, the light intensity falls off rapidly. Adding the components of ambient, diffuse and specular reflection gives the intensity of a single surface. If color is involved, there is an equation for each of the three primary colors.

The effect of this combination of operations is an unnatural, faceted appearance. Since the polygon is described by a single surface normal, adjacent polygons with different surface normals have different intensity values, and there is a noticeable discontinuity at the shared edge. Gouraud shading (named after its inventor, Henri Gouraud) averages intensity values at the vertices of the polygons and then across the scan lines to achieve smoothness. Phong shading (named after its inventor, the late Bui-Tuong Phong) improves on Gouraud shading by using a more detailed calculation that is more sensitive to the directional effects of specular highlights. The newest medium-priced, high-performance raster-graphics systems are now capable of doing the entire rendering job for approximately 3,000 polygons per second. They proceed by first processing an object hierarchy, including the application of geometric transformations to its components to simulate motion, next computing the viewing transformation and then carrying out the hidden-surface and smooth-shading algorithms. A few years ago this level of performance was available only on special-purpose flight simulators costing millions of dollars.

Other effects to be dealt with include shadows, light transmission and surface properties such as texture and grain. Shadow algorithms for point sources resemble algorithms for eliminating hidden surfaces in that they determine what surfaces can be "seen" from the light sources. Surfaces that are simultaneously visible from the viewpoint and from the light sources are not in shadow, whereas those that are visible from the viewpoint but not from the light source are. For distributed light sources the complex calculations must include both the umbra and the penumbra.

Light transmission is an even more difficult subject. "Specular" transmission, characteristic of transparent surfaces such as glass, is determined by the substance's index of refraction. Diffuse transmission through translucent materials such as frosted glass causes scattering in all directions. The most computationally complex and most realistic algorithms for dealing with both reflection and refraction are called ray-tracing algorithms. In essence they trace individual light rays to determine which of them end up at the viewpoint and how they got there. In order to avoid having to deal with an infinity of lines emanat-



COMPOSITE IMAGE of a seaside landscape, titled "Point Reyes," was produced by a team of workers at Lucasfilm. The landscape was defined by a variety of techniques; the different elements of the scene were rendered separately and later combined. The simple procedural-modeling technique shown in the illustration on the opposite page was used by Loren Carpenter to define the rocks, the mountains and the lakes; he also wrote the hidden-surface program and an "atmosphere" program for the sky and the haze. Rob Cook directed the project, designed the road, hills, fence and rainbow, and wrote the texture-mapping software. Tom Porter provided the procedurally drawn texture for the hills and also wrote the software for combining the elements to form a composite image. Bill Reeves defined the grass by means of a "moving particle" system he developed; he also wrote the modeling software. David Salesin put the ripples on the puddles, and Alvy Ray Smith designed and rendered the flowering plants.

ing from a point source, the process works backward, starting at each pixel. Each ray starting at the viewpoint and passing through a pixel is projected backward until it hits a surface. The backward tracing of the reflected ray then continues to determine whether it could have come from a light source or from reflection from another object. For a transparent surface a second, refracted ray must also be traced. Each ray is in effect a probe that must be tested for intersection with each object; only a small percentage of the rays ultimately have antecedents in a light source. New techniques exist to handle the reflection and refraction of diffuse light, but they are still very expensive in terms of the amount of computation they require.

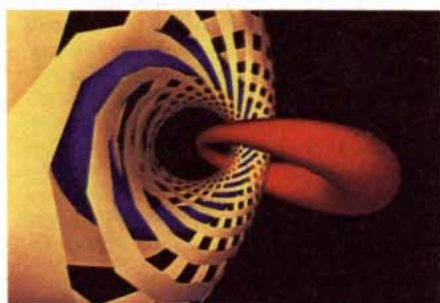
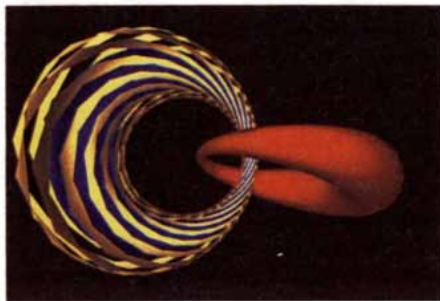
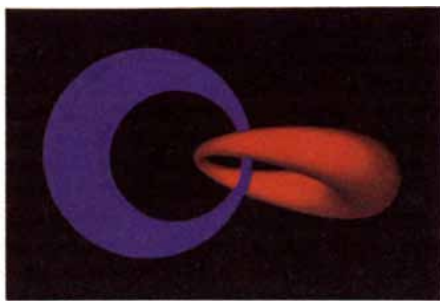
Surface texture can be handled by various models that build in local irregularities. For mapping a two-dimensional pattern onto a surface a pattern of intensity values can be used to modulate the intensities computed by the shading and shadowing algorithms. Some of the most recent work in image synthesis is concerned with effects such as depth of field, motion blur and the realistic rendering of objects in nature that exhibit both statistical regularity and irregularity, such as mountains, water, sky, trees and bushes.

Whether one is dealing with simple block diagrams or highly realistic pictures, the most important function of computer graphics is to increase one's

understanding, to enable one to experiment without danger, discomfort or undue cost and to help answer "what if" questions. For most modeling and simulation studies, however, mere static representations will not suffice: the phenomena one generally wants to understand are dynamic. A static picture may be worth a thousand words, but a moving picture is often worth many static ones. A key capability of the new generation of powerful work stations is to reveal the behavior of objects as they vary over time, by means of real-time user-controlled animation.

Among the objects exhibiting dynamic behavior that are of particular interest to programmers are programs and their data structures. Since the beginning of computer graphics in the early 1960's there has been considerable interest in using a diagrammatic approach to the design of computer hardware and software. Block diagrams, flow charts, module-interconnection diagrams, data-flow diagrams and many other symbolic representations have been used to portray systems whose designs were specified by typing in statements in a textual language. Although much hardware design today is done with graphical symbols, there are as yet no programming languages in common use in which the basic elements are pictorial rather than textual.

Thus graphics programs are specified in an ordinary programming language



such as FORTRAN or Pascal with “calls” to special-purpose graphics packages, not by specifying graphically what is wanted. The reasons for this curious contrast between the specification techniques for hardware and those for software include the compactness and precision a conventional programming language affords and the ease with which changes can be made with a text editor that includes good facilities for seeking out specified text patterns. Experience with true pictorial languages is lacking.

Although computer scientists know rather well what facilities a programmer needs to specify *how* to do something in a conventional programming language, they are still not very good at the much more difficult problem of letting a user specify *what* is to be done and then having the system automatically compile an appropriate procedure from this specification. Meanwhile considerable progress has been made in the development of work-station-based programming environments. In general these facilities enable the programmer to edit interactively and to “debug” programs with the aid of multiple views of programs and data, represented in the form of text or icons. The views are dynamically updated as the program is executed.

Two promising areas for the application of dynamic computer graphics are the classroom and the laboratory. Many schools and universities are turning to microcomputer-based instruction. One of the techniques adopted is the traditional “programmed learning” style of computer-assisted instruction, which emphasizes the acquisition of facts and skills. Computer-assisted in-

MATHEMATICAL APPLICATION of computer graphics is represented in this sequence of frames, adapted from the computer-animated film “Topology and Mechanics,” by Huseyin Kocak, Frederick Bisshopp, Thomas F. Banchoff and David Laidlaw of Brown. A hypersphere, an analogue in four-dimensional space of an ordinary sphere, can be visualized by “filling” it with two interlinked circles and a succession of surrounding toroidal surfaces. (The operation is roughly analogous to slicing a sphere into two points at opposite poles and a succession of parallel circles in between.) The frames show a series of perspective projections onto three-dimensional space, made from a viewpoint on the hypersphere, of two toroidal surfaces (one blue and one red) closely enveloping the two circles of the hypersphere. A third toroidal surface (yellow) is shown as it moves from the blue surface to the red one in six steps. The yellow surface has been cut into bands to reveal its linkage with the other two surfaces. This method of “decomposing” a hypersphere was inspired by work done by the German mathematician Heinz Hopf in 1931. The computer is invaluable in implementing such mathematical procedures, since it can easily be made to manipulate abstract objects in a higher-dimensional space.

struction is now being augmented by simulated “laboratory” experimentation and by “browsing environments” in which information is available much as it is in an encyclopedia or a library. Five years ago, at the instigation of my colleague Robert Sedgewick, the computer-science department at Brown University began to study the applications of workstation technology to education and research. Last year we inaugurated a novel electronic classroom, equipped with 55 high-performance work stations connected in a high-speed network. Most introductory courses in the computer-science curriculum are now taught in this specially constructed auditorium, as are sections of courses in differential equations, differential geometry and neuroscience.

Our aim is to offer students an opportunity to “see” an abstract phenomenon, and thereby to develop some geometric intuition about it, before delving into the details of programming or mathematics. We also want to involve them with the material more quickly by combining the classroom lecture with the laboratory experiment. Most students in a formal lecture are passive, even when questions are encouraged. As a result they do not really confront the material until they have to do homework or a laboratory exercise. Now an instructor can introduce a new topic by talking his way through an animated sequence of images viewed by all students and then letting each of them work independently on the same “interactive movie.”

Our primary support environment, the Brown Algorithm Simulator and Animator (BALSA), enables users to control the speed of an animated sequence, to decide which views of the subject to look at and to specify the input data to be processed. There is even a facility for running programs backward and undoing their graphical effects.

Multiple dynamic views of complex objects have turned out to be valuable not only to students but also to research workers. Much work has begun at Brown to learn how the technique can be generalized, both to other subjects in science and engineering and to fields that have no tradition of graphically representing their objects and processes of study. Are there intrinsic reasons for the absence of pictorial representation in some fields, or is it merely a cultural artifact? If many other ways of using graphics in various disciplines are found, how might that change classroom pedagogy? This question is particularly relevant as work stations start proliferating on the campus and many students have 24-hour-per-day access to them in their dormitory rooms. How do instructors not accustomed to programming specify and implement “courseware” without having to invest the usual preparation time for computer-aided



ELECTRONIC CLASSROOM developed by Robert Sedgewick and the author together with their colleagues in the computer-science department at Brown is equipped with 55 high-performance work stations connected in a high-speed network. The instructor can first pre-

view a topic through an animated sequence of images viewed by all students and then let each of them work independently on the same "interactive movie." The specially constructed auditorium has been used for courses in computer science, mathematics and neuroscience.

instruction of 100 hours or more per classroom hour? Many years of development and experimentation will be needed before this promising new capability can be widely disseminated.

Our interactive classroom demonstrations at Brown are an example of the more general category of what might be called "electronic books." A rather different example is the Spatial Data Management System developed at the Massachusetts Institute of Technology. This system makes it possible to browse in a two-dimensional "data land," an arbitrarily large desktop populated by icons. The cursor can be moved to point at any icon to display its contents. The data base stores text, diagrams, photographs, sound and television frames (retrieved in real time from a videodisk).

In a related experimental system at Brown the desktop metaphor is replaced by that of an associative network of text pages and footnotes, marginalia and cross-references. The reader can follow a trail of cross-references between associated topics, much as one would with an encyclopedia. Such a nonlinear manuscript has been called a hypertext by Theodor H. Nelson, a leader in the

movement to exploit computer-display technology to create a new literary form. Other projects at the frontier of computer graphics, initiated by Nicholas P. Negroponte and his co-workers at M.I.T., include an interactive automobile-maintenance-and-repair manual that can explicate text by creating customized "movies" based on sequences of stored frames from videodisks, and an electronic newspaper that continuously scans the wire-service reports and its data base and formats stories and pictures for the reader on the basis of a stored reader-interest profile.

Computer professionals, and particularly specialists in graphics, can take pleasure in the fact that finally our promises of the utility and convenience of computing for the general public are being kept. Computer graphics, once the domain of experts, is commonplace as even children in elementary school work with windows, mice and computer displays as instruments for drawing and indeed for imagining. Thinking and programming in terms of graphics are becoming an integral part of learning to construct algorithms.

And what of the future? There must still be an improvement of orders of magnitude in the price-performance ra-

tio of hardware before the average user can have the equivalent of a real-time flight simulator on his desk (let alone on his lap), and much more progress will have to be made in understanding the interaction of the laws of physics and aesthetics before computer artists will know how to render convincingly realistic scenes that also please the eye. Eventually true three-dimensional displays will open up yet another realm: research on digital holograms may result in real-time computation of lifelike scenes. On the input side, additional work is needed on the user interface. For example, pictures must be better integrated with sound, as in the case of voice input and output. Much progress will also have to be made in understanding the structure of natural language and in related areas of artificial intelligence before users can carry on conversations with their computers. New methods of providing tactile control and feedback are also needed for exploring the "feel" of the objects seen on the screen. For me the ultimate ideal is expressed in the old comic strip "Mandrake the Magician": "Mandrake gestures hypnotically..." and in the twinkling of an eye a new scene, a new sensory environment, is conjured up.

ODESTA HELI

An Interactive Information System
For Individuals With A Need To Know

With Odesta Helix, you can now manage any information with easily created forms. Begin using Odesta Helix immediately, without having to set "field parameters" or learn a programming language. You can type in a free-form report, or a table of calculated numbers. You can even use pictures from MacPaint to illustrate your inventory or teach language skills. Odesta Helix is a new program, not an old program made to work on a new machine. Odesta Helix lets you experience the real power and flexibility of windows, pull-down menus, icons, and mouse interaction because it takes a new approach to information and knowledge formation.

File Edit Icons Display Search

Packing List

Order Form Invoice

Cu Invoice
Cu AirBorn Nocturnal Bill to: Ms. F. Godmother
239 Getaway Drive 23 Park Lane
Suite 2743 Room 502
Dallas, Texas 73402 Northbrook IL 60062

Part #	Description	Unit	Qty	Price
2A02-3	Pumpkin Coach	49.95	1	49.95
2A03-7	Mice for Coach	2.95	8	23.60
2A13-9	Glass Slipper	19.95	1	19.95
Total				93.50
Tax				4.68
Shipping				22.50
Pay this amount				120.68

Invoice #: 12703
Customer Order #: 430

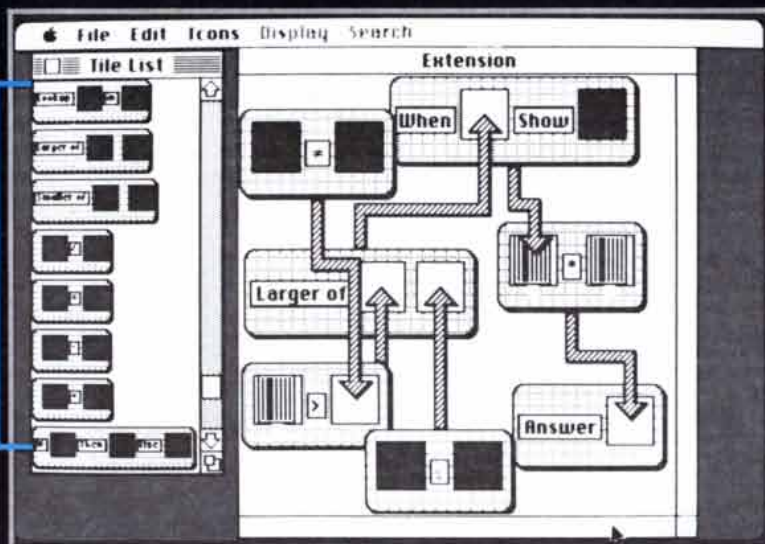
Automatically generated. Available for use in any other reports or forms.

Set up your own forms (or customize provided forms) and then input information only once. Here is just one example: Enter an order, and generate an invoice, adjust inventory, update customer records, and keep track of receivables. Everything you enter can be easily modified, redefined, and interrelated. There are no barriers placed between you and the way you want to do things.

Odesta Helix & Macintosh™

A Personal Decision Support System An Individualized Knowledge Base

Tiles act as arithmetic, text, Boolean and date operators, functions and valves.



Visual building blocks let you set up even the most complicated statement or calculation and use it whenever you want. You "flow" the information from one block into another – just by pointing. Each "diagram" shrinks down to become an abacus icon (see "Number Format" screen shot). Move its icon within and between forms, or embed it in another "tile". The magic of the tiles and arrows has to be seen to be believed – ask your dealer for a demonstration.

Odesta Helix automatically creates icons to represent every piece of your information. Make them appear if you wish, and rearrange them within or between forms. Define and restrict them if you wish, for data checking and security. But most of the time, just let them stay hidden "behind" the form.

Write a letter, enter a memo, or log in your research – all within Odesta Helix. You don't need another program to work your text. Cut, copy, paste, search and replace within or between any of your forms and reports. Or combine information with a letter and do a "mail merge".

No special commands to learn in order to search through your "ocean" of information. Simply ask a question by pointing to a spot on a form, or by filling in spaces with your own questions – Odesta Helix will search and sort at the click of a mouse. Example: "Find all books published before 1940, that are first editions costing more than \$13.00, and that are about World War I and are illustrated." Imagine the possibilities.



Computer Software for Information Management

Enormous volumes of stored data are of use only if information can be retrieved quickly in an understandable form. Software for the purpose must reflect the structure of the data base and of the storage medium

by Michael Lesk

As anyone with a cluttered office knows, having a large quantity of information on hand does not guarantee ready access to any particular piece of information. In the past two decades there has been a rapid increase in the capacity of electronic machines to store information and an equally rapid decrease in the cost of storage. In general the development of software for organizing and retrieving the electronically stored data has not kept pace. Those who are writing programs for information management are in the position of having to catch up with the capacities of the computing machinery.

What principles are guiding the effort? One principle is that the best form of organization depends on the content of the information and on how the information is to be used. For example, programs that maintain a list of names have been written for many purposes; they vary considerably according to what information is associated with each name and how the names are retrieved. A commercial system called Soundex is employed to identify airline passengers with reservations for a particular flight. Soundex stores the names phonetically, which reduces the number of transcription errors and allows a name to be found even if the exact spelling is not known. The Chemical Abstracts Service maintains programs for determining whether a particular substance has already been identified, a task that

in a formal sense is similar to the one done by Soundex. The systems are not phonetic; furthermore, they record considerable information about chemical structure and nomenclature. Information specific to one domain of knowledge can improve the efficiency of a program in accomplishing one job, but the program then becomes less suitable for other purposes. Programs designed for a chemicals data base would be a poor choice for listing airline passengers.

Another consideration is the physical structure of the storage medium. Magnetic disks for storing data became common in the late 1970's. On a disk data are recorded in subunits called blocks, and access is most efficient if the logical divisions of the data approximate the boundaries of the blocks. Thus software aimed at managing large quantities of information is constrained on the one hand by the structure of the machinery and on the other hand by the content of the information. The effort to design software that more fully exploits the capacities of current machines is largely defined by the need to accommodate these two fundamental constraints.

A small group of related items in an electronic data-storage system is generally referred to as a record. For example, in a file describing the stock on hand in a supermarket each record might include the type of product, the general category of goods of which it is

a part, the number of the aisle where the product is to be found and the price. Each item in the record, such as the type of product, is referred to as a field. The record is retrieved from the electronic file by means of a key: a label that can consist of a field, a part of a field or a combination of several fields.

Some types of fields are employed as keys more frequently than others. It is probable that in the supermarket data base the aisle number would serve as a key but the price would not; hence the user could easily find all the products that are for sale in aisle 3 but could not easily find all the products that cost 49 cents. Other information, such as the name of the wholesaler that supplied the product, the shelf life of the product and the quantity of stock on hand might or might not be keyed. The software employed to manage the information should make it easy to search for the record that includes a particular value of the key.

It should be noted that the key need not be taken directly from the record. In formulating a directory-assistance system for AT&T Bell Laboratories I transformed nicknames in the storage record into their formal equivalents in the key. Thus "Chuck" in the record became "Charles" in the key. The transformation, however, was specific to the domain of telephone listings: in a geographic data base Billings, Mont., clearly would not be transformed into Williamings.

The varying relations among records, fields and keys serve to define the three main ways of organizing electronic records: the hierarchical, the network and the relational. The hierarchical system is so named because of the ordering of the fields in the record. In each group of records one field is designated the master field and the other fields are subordinate to it. Groups of records are arranged in a serial order resembling the rungs of a ladder and data can be re-

SHORTEST ROUTES TO THE PLAZA HOTEL from Wall Street were calculated by a computer program, which thereupon drew the map of the southern part of Manhattan Island shown on the opposite page. One route, shown in white, minimizes travel time and the other, shown in orange, minimizes distance. Because of the pattern of one-way streets in lower Manhattan, each route begins with a change of direction. The route that minimizes time passes along West Street to Tenth Avenue, which is followed uptown. Stored information about the average travel time on each street enables the program to specify the route that takes the minimum time. The route that minimizes distance begins on West Street, then passes uptown along Sixth Avenue. Software for storing and processing the information in the map was formulated by the author and his co-workers. The program for identifying the routes that minimize time and distance was written by one of those co-workers, Jane Elliott of AT&T Bell Laboratories.

tried only by traversing the levels according to a path defined by the succession of master fields.

Hierarchical data bases have been employed since the beginning of the modern period of machine computation in the 1940's and examples of their operation could be chosen from many fields. As a simplified example, consider the supermarket data base discussed above. The first level of organization in such a body of information could include a table giving the aisle numbers and the general category of goods available in each aisle. The category of goods serves as the master field. Only by retrieving the table of aisles and their contents and then selecting a category, such as produce, can the tables at the next level be reached.

The second level of the hierarchy might include tables that list the specific products available in an aisle. The tables farther down might include the price of each product, the wholesale supplier and the product's shelf life. Only by traversing several levels of the hierarchy in sequence can information about a particular article, such as its price, be retrieved. Unless additional indexes are constructed within the file it is costly in terms of computer resources to ask questions that deviate from the hierarchical path, such as inquiring about the price directly.

The network model is somewhat more flexible than the hierarchical one because multiple connections can be established between files. Such connec-

tions enable the user to gain access to a particular file without traversing the entire hierarchy above that file. By this means the subsidiary connections modify the vertical structure of the data base in a significant way. For example, in the supermarket data base a connection could be established between the list of aisles and the table of prices, so that it would be possible to find the price of an article without first retrieving the intermediate table that identifies the products for sale in the aisle.

The relational model, which was developed by E. F. Codd of the International Business Machines Corporation in about 1970, is currently the subject of much interest because it promises greater flexibility than the other types of data bases provide. In both the hierarchical form of organization and the network form some questions are answered much more readily than others. Moreover, which questions are difficult to answer and which questions are easy to answer is determined when the data base is constructed. In many instances there is no sound basis for determining in advance what the most frequently asked questions will be.

In the relational data base flexibility is achieved by abolishing the hierarchy of fields. All fields can be utilized as keys to retrieve information. A record is not thought of as a set of discrete entities with one item being designated the master field; instead each record is conceived as a row in a two-dimensional

table and each field becomes a column in the table. The entry

Aisle 2 Dairy Milk .69 quart

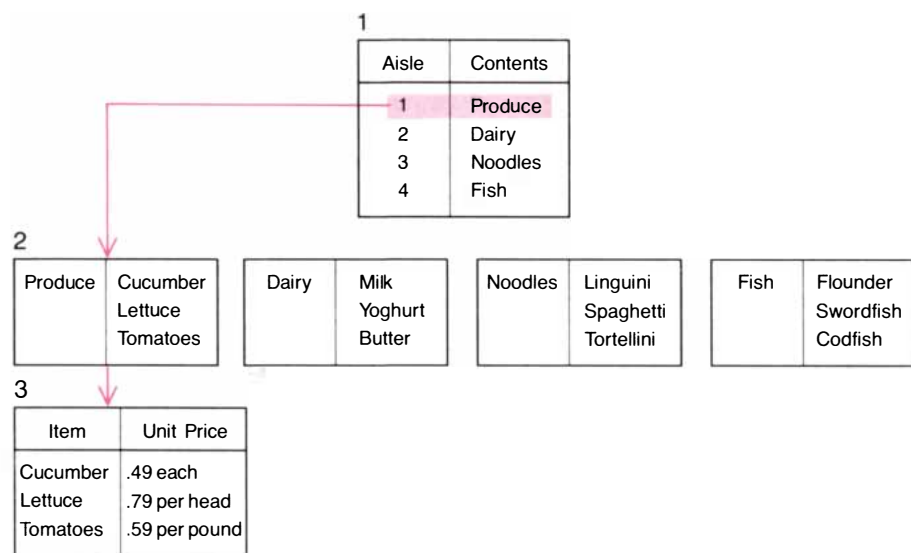
can be thought of as a relation between the aisle field, the general-category field, the product field and the price field. The relation could be augmented with the name of the wholesaler, the shelf life of the product, the quantity of stock on hand and other facts. Shorter relations consisting of two fields can be obtained from the full set of relations by choosing the appropriate fields; the process is referred to as projecting the relation. Thus the price of any product can be retrieved quickly, as could a table listing all the products in one aisle.

The classification of data bases into the hierarchical, network and relational forms is commonly cited in discussions of information management. The classification scheme is not as useful as it might appear, however, because the structure of any data base can be supplemented with secondary indexes that make it possible to answer efficiently queries that do not follow the underlying organization. Furthermore, certain problems are common to all three types of data base. Consider the task of selecting one record from a file made up of many related records. The file could consist of dictionary entries, the lines of a telephone directory or other records, but in all cases the problem is to retrieve one record quickly and efficiently.

The case of picking out a single record depends critically on how the items are arranged in the main memory of the computer or on a secondary storage medium such as a magnetic disk. Several techniques for arranging data can facilitate subsequent retrieval; the techniques can be employed with any of the three types of data base.

For a small body of information it may not be necessary to maintain any particular arrangement. Lines can be stored on a disk in an arbitrary sequence; when an item must be retrieved, a simple pattern-matching program scans the lines sequentially for the appearance of particular combinations of symbols. In the Unix operating system, for example, a program named *grep* is often employed in this way.

Suppose a list of telephone numbers, including those of two seltzer delivery companies, has been stored in a file named *telnos*. In the Unix system the command *'grep seltzer'* results in the printing of the names and numbers of the two companies that deliver seltzer. In the *telnos* file the word "seltzer" was entered before the name of the companies, but the key need not be at the beginning of the line for the *grep* program to operate. The command *grep beverage < telnos* results in the printing of the same two lines because the



HIERARCHICAL DATA BASE consists of tables that must be scanned in a predetermined order to retrieve information. The illustration shows the steps that might be needed to find the price of a product in a file storing data about the stock on hand in a supermarket. The first table in the hierarchy identifies the aisles and the general category of goods available in each aisle (1). The general category can be employed to retrieve a table on the next level that lists the specific products on the shelves of the aisle (2). The third table includes the price of each product (3). Such a form of organization would be convenient for the clerk who must put a price on each article, because the method of access follows the clerk's path through the market. It would be less convenient, however, for directly answering shoppers' inquiries about prices. In the related form of organization called the network data base a subsidiary connection could be established between table 1 and table 3. Such a connection between tables would reduce the time that is needed to find the price, but it would also require additional storage space.

word “beverage” is part of the name of each company.

Maintaining a file of items that are not in a predetermined order has several advantages. There is the saving of time and effort that would be needed to sort the data when the file is set up. In addition considerable flexibility is maintained because it is not necessary to decide in advance which items in the file will serve as keys when the data are retrieved. If a system user decides to find all the numbers in the *telnos* file with the prefix 800, the command *grep 800 < telnos* works immediately, regardless of whether or not such a request was anticipated when the file was established. If the list is not ordered, new data can be added at the end of the file without rearranging the previously stored items. Furthermore, no space is required for storage beyond the space that is taken up by the individual entries. In more complex storage systems some memory capacity is taken up by information needed to organize the data.

The unordered method of storage has a critical drawback: retrieval is very slow. Each line in the file must be scanned separately and as a result searching a file of 10,000 lines takes 100 times as long as searching a file of 100 lines. The algorithm employed by the *grep* program corresponds to finding books in a library by starting at the shelves nearest the entrance and examining each book until the one wanted is found.

To speed up the search process the file can be sorted into serial order, such as alphabetical order or numerical order. Putting the items in serial order makes it possible to retrieve an entry by the technique called binary search. The list is divided in half and the program determines which half includes the item sought. The process is then repeated until the item is found.

Suppose the file consists of the entries in a dictionary and the definition of “cat” is sought. A binary-search program would first identify the middle item in the dictionary, which turns out to be “legality.” By comparing the first letter of “legality” with the first letter of “cat” the program determines that “cat” is in the first half of the file. The midpoint of the first half is “distort” and so “cat” is in the first fourth of the dictionary. After the next division, at “castigator,” the second portion rather than the first must be searched. By continuing through the sequence of midpoints, the definition of “cat” is isolated.

Binary search is quicker than searching an unordered data set. Assume the file to be searched includes n items. On the average finding any one item by searching through an unordered set requires $n/2$ operations. Binary search requires at most about $\log_2 n$ operations to find a particular item (where $\log_2 n$ is

1

Product	Aisle	Category	Price	Unit
Butter	2	Dairy	1.99	pound
Codfish	4	Fish	2.09	pound
Cucumbers	1	Produce	.49	each
Flounder	4	Fish	3.29	pound
Lettuce	1	Produce	.79	head
Linguini	3	Noodles	1.29	pound
Milk	2	Dairy	.69	quart
Spaghetti	3	Noodles	1.29	pound
Swordfish	4	Fish	4.49	pound
Tomatoes	1	Produce	.59	pound
Tortellini	3	Noodles	3.49	pound
Yoghurt	2	Dairy	.79	pint

2

Cucumbers	.49	each
-----------	-----	------

3

Cucumbers	Aisle 1	Produce	.49	each
Lettuce	Aisle 1	Produce	.79	head
Tomatoes	Aisle 1	Produce	.59	pound

IN THE RELATIONAL DATA BASE each entry is made up of a list of connected items; any subset of the items in the full list can readily be retrieved. In a supermarket data base the items might include the type of product, the aisle where the product can be found, the general category to which the product belongs and the price of the product (1). More specific relations, such as the price of an item, can readily be derived from the complete set of relations (2). A table showing the articles available in one aisle and their prices can also be constructed (3). The relational data base is valuable when the most frequently asked questions are not known in advance.

the logarithm of n to the base 2). The serial order has another advantage: When an entry has been located, it is a simple matter to find out about adjacent entries, such as the word after “cat” in the dictionary.

A Adding a new entry to a file set up for binary search, however, is a costly process because the sorted order of the file must be maintained. Since a new item will on the average be inserted in the middle of the list, half of the items in the file must be moved each time a new entry is added. The fact that entries can be found only from the key on which the file is sorted can also be a serious limitation. It is possible to duplicate the items and store multiple files sorted according to different keys, but that consumes much additional space.

Another technique for retrieval from a sorted file relies on subgroups of adjacent data entries called buckets. The first entry in each bucket is stored in a table that serves as an index to the divisions of the file. If n items are divided among \sqrt{n} buckets, with each bucket holding \sqrt{n} items, a linear scan of the bucket index followed by a scan of the appropriate bucket to isolate the needed entry takes only slightly more than \sqrt{n} operations. Although \sqrt{n} operations is not as good as $\log_2 n$, particularly for a large file, it is not unwieldy for a small file. Furthermore, programs for bucket storage are simple to write.

So far the best we have been able to do in searching a file is $\log_2 n$ operations.

It is possible to improve on that figure by means of the procedure known as hashing. To understand the advantages of hashing consider the possibility of assigning a number to each entry in a file. If the number could be computed quickly by means of a simple algorithm each time an item is needed, the item could be fetched directly from the file without searching.

For the entries of a dictionary such an indexed array could be created in principle by a straightforward method. Each letter of the alphabet can be assigned a number, so that the spelling of a word designates a unique number that serves as the word’s address in memory. The trouble with such a scheme is that the memory space required is immense, and it would remain almost entirely empty; most combinations of letters, after all, do not spell an English word. To look at the problem another way, the difficulty is that the first letters of English words are not distributed evenly. For example, words beginning with *c*, *s* or *t* are much commoner than words beginning with *k* or *w*. If 100 words were assigned to 100 numerical addresses on the basis of their first letter, they would pile up in certain slots instead of filling slots 1 through 100 in a smooth distribution.

The most convenient solution is to formulate an algorithm that assigns a “pseudorandom” number to each word. The spelling of the word fully determines the pseudorandom number, but words spelled differently can generate the same number. The algorithm

is based on a mathematical expression called a hash function. In one possible hash function each character in the alphabet is assigned a numerical value and the values for all the characters in a word are summed to yield the pseudorandom number that serves as the address. If an effective hash function is chosen, the entries are distributed fairly smoothly throughout the indexed array, which is referred to as a hash table.

It is generally necessary to leave at least one-fourth of the slots in the hash table empty. Leaving some slots empty reduces the frequency of cases in which the same pseudorandom number is assigned to more than one item. When such a duplication, which is known as a collision, takes place, an additional algorithm is invoked to pick a slot for the second item. The algorithm might call

a

```
UNORDERED FILE
seltzer, excelsior beverage co. newark 242-0412
new york air, nyair 800-221-9300
usair 622-3201
seltzer, elliot beverage somerville 356-0273
united airlines (ua) 624-1500
imagen systems (laser printers) 496-7200
```

b

```
COMMAND: 'grep seltzer
OUTPUT:
seltzer, excelsior beverage co. newark 242-0412
seltzer, elliot beverage somerville 356-0273
```

c

```
COMMAND: grep beverage < telnos
OUTPUT:
seltzer, excelsior beverage co. newark 242-0412
seltzer, elliot beverage somerville 356-0273
```

d

```
COMMAND: grep air < telnos
OUTPUT:
new york air, nyair 800-221-9300
usair 622-3201
united air lines (ua) 624-1500
```

UNORDERED FILE is made up of entries that are stored without being put in any particular arrangement. The top panel shows such a file: a small telephone directory designated *telnos* (a). When the command *grep* in the Unix operating system is given, each line in the file is scanned in sequence and all the lines that include a particular key, or combination of symbols, are retrieved. The names of the companies in the list that deliver seltzer can be retrieved by means of the key *seltzer*, which has been entered at the beginning of both entries (b). The same pair of entries can be retrieved by means of the key *beverage*, which is found in the names of both companies (c). The names of the airline companies included in the list can be found by means of the key *air*, which is part of the name of each airline (d).

for the next slot in the hash table to be filled; as an alternative a second hash function could be computed. If the distribution of entries is known, the table can be kept more than three-fourths full, but it is not common for the distribution to be known.

It should be noted that when a body of information is organized in a hash table, the hash function is computed each time an entry is sought. The hash function is simple to compute, and when it has been computed, the entry is retrieved without additional searching. Hashing is therefore a very fast method of retrieval. Revising the file is also efficient since the items are not stored in a serial order that must be preserved by moving many items each time an entry is made.

The technique of simple hashing has a drawback, however, that makes it unsatisfactory for many applications. Because of the problem of collisions, the approximate number of entries must be known in advance so that a hash table of the appropriate size can be constructed. If many new items arrive unexpectedly, it may be necessary to recompute all the hash functions. In practice it is often not possible to predict the size of a set of data, and so it is inconvenient to have to choose the size of the hash table in advance.

Unordered files, buckets, binary search and hashing are all techniques in use today, particularly in conjunction with small data bases where ease of programming is a more important consideration than ultimate efficiency. Two methods developed in recent years are more commonly employed with large data bases: extensible hashing and *B*-trees.

Extensible hashing was devised to avoid the need to specify the size of the hash table in advance. A hash code longer than necessary is computed and only the part of the code needed to accommodate the number of entries on hand is utilized; the rest of the code forms a reserve against an increase in the size of the file. The details of extensible hashing are beyond the scope of this article, but the outcome of the method is to preserve the speed of hashing at a small cost in additional storage.

In all the variants of hashing, file entries are stored in the arbitrary order determined by the hash function. Any sequential relations that might have existed among items in the original data set are lost in the storage process. For example, if a dictionary is stored in a hash table, words beginning with *c* are scattered at random through the array. As a result, when a word has been retrieved, it is not possible to obtain a quick answer to questions about entries that were adjacent to it when the words were in alphabetical order.

A *B*-tree does make it possible to efficiently answer questions about items

that were neighbors in the original sequence. The *B*-tree is a mechanism for implementing binary search in which the repeated divisions of the file are incorporated into the data structure instead of being calculated by the algorithm. The *B*-tree has the form of an inverted tree: there are many categories (leaves) at the bottom and only one category (the root) at the top. Each category, or node, is made up of a set of keys for data entries.

At the bottom level of the tree each node includes a group of entries arranged in order without omissions. At the next level up each node includes one key from each node of a subset of the nodes at the bottom. The process of reduction continues to the top of the tree, where there is a single node. The tree is traversed from top to bottom with the keys in each node serving as pointers to nodes one level down.

If a dictionary were stored as a *B*-tree, the first node might contain the words "chromophore," "epicycle," "impolite" and so on, which constitute a set of dividing points for the alphabet. "Chromophore," the first key, could point to a node at the second level that includes the words "alfalfa," "apocryphal," "available," "binocular," "bully" and "celery." It can readily be seen that the second list includes dividing points from the beginning of the alphabet to the word "chromophore." Each item in the second group points to a more finely resolved list. At the bottom of the tree the nodes point to the dictionary entries themselves.

The *B*-tree has become popular for several reasons. As noted above, a *B*-tree makes it possible to answer questions about the adjacent items when one item has been retrieved. In addition *B*-tree storage is relatively fast: a search requires roughly $\log_2 n$ operations and adding or deleting entries also requires roughly $\log_2 n$ operations.

One of the main reasons *B*-trees have been so widely applied is related to the physical structure of magnetic-disk storage. It is often assumed for the purpose of approximation that each search in a file takes about the same amount of time. It is only in main memory, however, that each search takes an unvarying period, and data bases large enough to be of interest do not fit in main memory. Large data bases are stored on disk, where there are two types of search with quite different retrieval times. The random-access time is the average time needed to retrieve a record from an arbitrary position on the disk. The sequential-access time is the time needed to retrieve the record following the one most recently accessed. In a typical machine the random-access time might be 30 times greater than the sequential-access time. An efficient program therefore

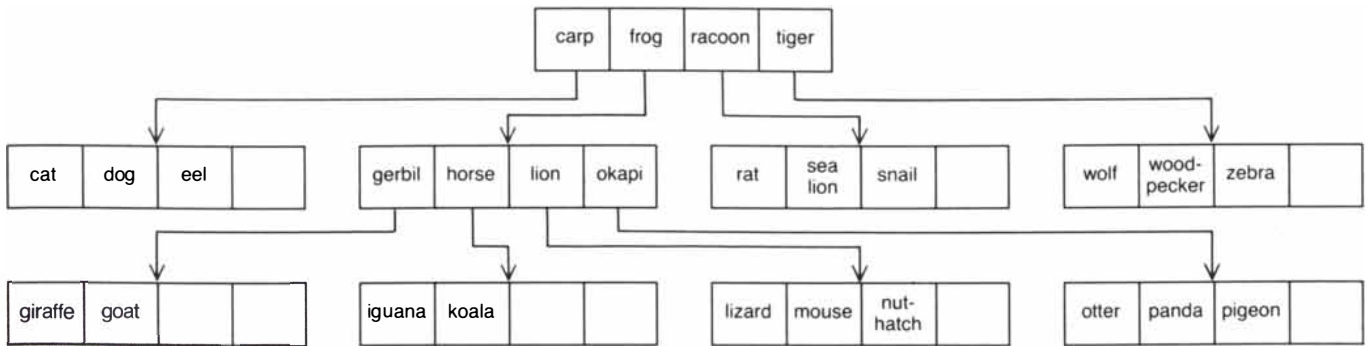
maximizes the number of sequential searches and minimizes the number of random ones by retrieving relatively large groups of data each time a search is done. If the data are stored in a *B*-tree, the size of the nodes at the base of the tree can be adjusted to match the size of

a disk block. Hashing cannot readily exploit the disk structure.

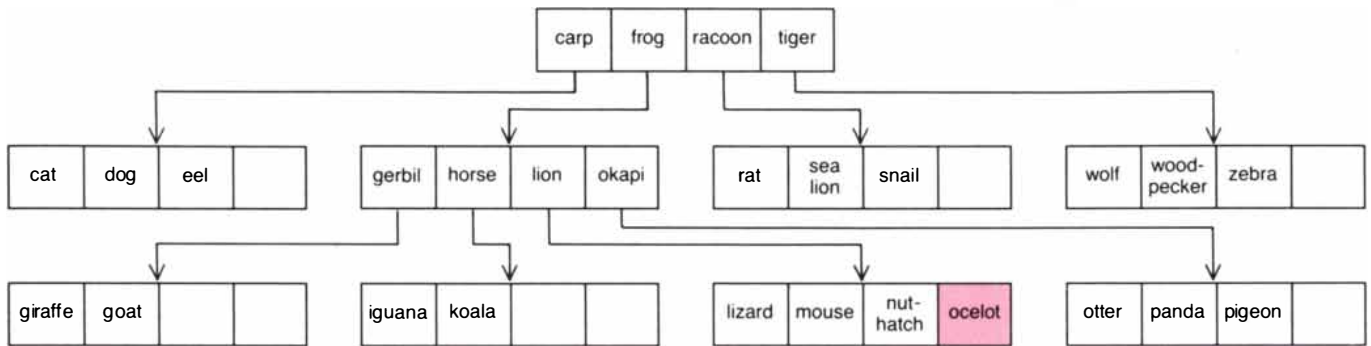
In the preceding discussion the methods for storing and retrieving data have been considered separately. In the design of an actual system, however, it is

often necessary to combine several techniques to achieve the best operational result, as is suggested by several examples drawn from work I have done with my colleagues. One experimental program we have devised gives weather forecasts for any town in the U.S. If the

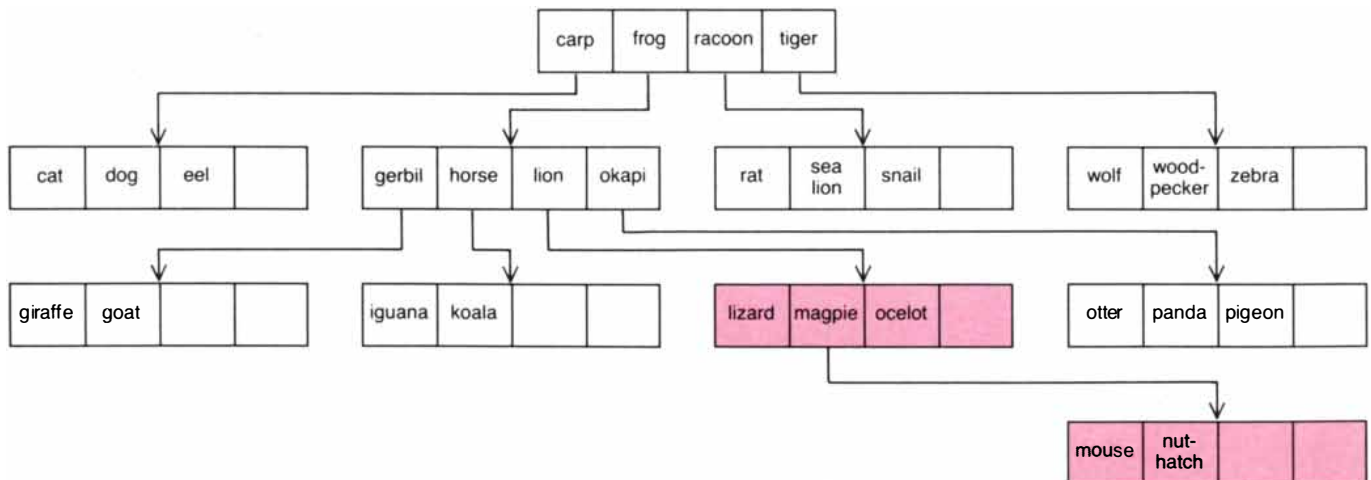
a



b



c



STRUCTURE OF *B*-TREE and strategies for adding items to the tree are shown for a small file made up of the names and descriptions of animals stored in alphabetical order. The *B*-tree is an ordered form of data storage consisting of nodes, or small groups of keys. Each node includes keys that divide the file or part of the file into fractions (a). The top node includes keys that function as dividing points for the entire file. Each key in the top node points to a node on the level below. The keys in each lower node fill the gaps between the keys in the upper node. For example, the gap between *frog* and *racoon* is filled by *gerbil*, *horse*, *lion* and *okapi*; the key *frog* in the top node points to the lower node that includes them. To find the entry

lizard, the top node is scanned and it is found that *lizard* lies between *frog* and *racoon*. When the second node is scanned, it is found that *lizard* lies between *lion* and *okapi*. *Lion* points to the node that begins with *lizard*. The key *lizard* points to the entry describing the organism (not shown). Adding an item to a *B*-tree can be a simple job or a complex one depending on the position of empty slots. To add *ocelot* to the tree is straightforward: the entry is put in the empty slot after *nuthatch* (b). If *ocelot* has been added, however, adding *magpie* requires that the node beginning with *lizard* be divided into an upper and a lower node (c). In that way the arrangement of pointers is preserved: *mouse* and *nuthatch* fill the gap between *magpie* and *ocelot*.

user asks for the weather in a particular town, the program finds the closest place to the town where the weather has been recorded and reports the most recent observations. It then goes on to locate the nearest weather forecasts and reports them.

Presenting such weather information entails the use of three data bases and software for each one. The system relies mainly on a National Weather Service communications circuit that delivers six megabytes of weather information per day, including observations and forecasts. The observations are made at airports, which are identified only by a three-letter code; as a result two data bases are needed in addition to the one that holds the weather information itself. The first supplementary data base, a table that gives the latitude and longitude of each airport, is drawn from records maintained by the Federal Aviation Administration. The second, a table that gives the latitude and longitude of each U.S. town, is drawn from Bureau of the Census records.

The town data are stored as a *B*-tree. The weather reports are stored in a bucketed file where the U.S. is divided into small squares of latitude and lon-

gitude. When a set of weather observations arrives, the airport code is converted into a location by means of the *B*-tree and the information is stored in the appropriate bucket. When a request for weather information is made, the bucket is searched for the closest airport. If the data were in one-dimensional rather than two-dimensional form, the entire job could be done with *B*-trees, but *B*-trees do not accommodate two-dimensional data well. The weather service is often used by my associates who are going on trips and want to know the weather at their destination.

Another service we provide is a means of selecting information from news stories filed by the Associated Press. The program is currently experimental and has about 100 participants. Each day about 200,000 words are stored in the computer system. There are two main modes of access to the information in the news stories. In one mode the user picks current stories from a "menu" display; the stories are selected according to a few words that serve as the title in a computer-terminal display. About 40 people read a total of 840 stories per day on the average by picking them from the menu.

In the second mode of access readers are able to retrieve stories by means of a profile consisting of particular words, phrases or syntactical operations. Someone who wants to read about Mount Everest can ask for all the Associated Press stories in which the word "Everest" appears. Queries based on phrases such as "space shuttle" or on features such as the inclusion of "telephone" and "regulation" in the same sentence can also be answered. About 50 people maintain such standing requests and some 550 stories per day are sent out because they match a particular profile.

The systems I have described so far are all designed to store and retrieve information in the form of numbers or words, but it is also possible to process large quantities of information derived from graphic images. Consider the problem of storing a street map in a computer file and then employing the file to answer questions that would ordinarily be answered by consulting the map. The information in a street map can be reduced to two main types of data: the location of nodes and the location of edges. A node is a point where two streets intersect; an edge is the segment of a street that connects two nodes. The positions of nodes and edges could be converted into digital form by devices called digitizing tablets or scanners, but fortunately the conversion has been done by the Census Bureau. Our system for processing maps is based on data obtained from the bureau.

How should the maps be stored? The data are copious, and what is even worse from the point of view of storage, they are two-dimensional. The storage system must be able to accommodate both these properties. Several storage methods could do so, and to select one we considered the queries that the system should be able to answer efficiently. Four types of query are significant: finding the location of a building when the street number, the street name and the Zip Code are known; finding out whether two streets intersect and, if they do, locating the intersections; finding all the points that can be reached directly from a given point, and finding all the streets within a certain radius of a given point.

Two techniques for storing data plotted in two dimensions are the connection matrix and the *k-d* tree. In the connection matrix the only information that is stored is the list of all pairs of nodes that are linked by an edge. Such a file does not include enough information to support the map system because it is essential for the system to include street names and the locations of the nodes. The street names must be entered so that the program can determine when the route passes from one street to another.

COMMAND: \$ date

OUTPUT: Fri. May 4 12:55:48 EDT 1984

COMMAND: \$ weather
elmira, ny

OUTPUT: Elmira, NY: (42.093 N, 76.807W)

6.3 miles NW at the airport in Elmira, NY (CHEMUNG COUNTY) (11:55 AM EDT):
temperature 55, humidity 96, weather overcast, visibility 15 miles

Next 48 hours at Rochester, NY (ROCHESTER-MONROE COUNTY)

To 8 PM EDT/5: high 62 low 41, prob. precip. to 8 AM 30% to 8 PM 10%

To 8 PM EDT/6: high 66 low 45, prob. precip. to 8 AM 40% to 8 PM 60%

Forecast For Western New York

National weather service buffalo ny

430 am edt fri may 4 1984

Rain..Heavy in spots..Becoming intermittent during the day from west to east and ending tonight. Highs in the mid to upper 50's today and lows tonight about 40. A mix of clouds and sunshine Saturday. Highs 60 to 65.

WEATHER INFORMATION for Elmira, N.Y., is presented by a program devised by the author. After checking the date the user asked for information about the weather in Elmira. The program found the nearest point where the weather had been observed (the Elmira airport) and reported current observations. It then found the nearest weather forecasts. The forecast for Rochester was compiled mechanically; the one for Buffalo was compiled by human beings. The weather-information program, which can report the weather for any town in the U.S., relies mainly on National Weather Service observations that are made at airports. Hence two data bases are required in addition to the one that records the weather information itself: a table giving the latitude and longitude of each airport and a table giving the latitude and longitude of each town. When a request is made, the program identifies the latitude and longitude of the town, finds the nearest airports and reports forecasts and observations from the airports.

**IT'S A SMALL MIRACLE HOW HEWLETT-PACKARD
PUT 656K OF MEMORY, LOTUS 1-2-3, WORD
PROCESSING, A TELECOMMUNICATIONS MODEM
AND COMPLETE IBM CONNECTABILITY INTO**



THE PORTABLE.

For years business people had to choose between the power of a desktop computer and the limited capabilities of the first portables. That problem was solved when Hewlett-Packard introduced The Portable.

The Portable is designed with more total memory than most leading desktop personal computers...656K in fact. That includes 272K of user memory. So, The Portable's built-in business software can work with enormous amounts of data.

1-2-3™ from Lotus™ America's most popular spreadsheet, file management and business graphics program, is permanently built into The Portable. So is Hewlett-Packard's word processing program, MemoMaker. Just press the key and you're ready to work.

The Portable even has a built-in modem and easy-to-use telecommunications software to send

or receive data using a standard telephone jack.

If you use a Hewlett-Packard Touchscreen PC, IBM® PC, XT or an IBM compatible you'll be glad to know that your desktop and The Portable can talk to each other with the simple addition of the Hewlett-Packard Portable-Desktop Link.

The Portable's rechargeable battery gives you 16 hours of continuous usage on every charge.

Finally, you can work comfortably on a full size keyboard and an easy-to-read 16-line by 80-column screen. And it all folds shut to turn The Portable into a simple nine-pound box.

The Portable. A small miracle...perhaps. But then consider where it came from.

See The Portable and the entire family of personal computers, software and peripherals at your authorized Hewlett-Packard dealer. Call (800) FOR-HPPC for the dealer nearest you.

Setting You Free



PG02412 236 A

IBM is a registered trademark of International Business Machines Corporation. 1-2-3 and Lotus are trademarks of Lotus Development Corporation.

© 1984 SCIENTIFIC AMERICAN, INC

**Your Key To
Microcomputer Software!**



**More
software
for more
computers
...and more.**

Whatever your software needs, all you need to know is Westico. We have hundreds of business and professional software programs in formats to fit more than 120 microcomputers, including IBM PC, MS DOS and CP/M-compatible systems. Our large inventory means you get the software you want, when you want it. Plus, our after sales service is designed to keep you smiling. Westico helps you get the most from your microcomputer.

Find out more with our new directory. Detailed descriptions of all our programs help you select the correct software to fit your needs. Start getting more with Westico.

— Order Your Copy Today —

Rush me the brand new Westico software directory.

Name _____
Firm _____
Address _____
City _____ State _____ Zip _____

Mail to:



25 Van Zant Street • Norwalk, CT 06855
(203) 853-6880 • Telex 64-3788

© Copyright 1984 Westico, Inc. WES-37.

er. The locations of the nodes must be entered so that a left turn can be distinguished from a right turn and also so that a physical version of the map can be drawn.

The *k-d* tree is a variant of the *B*-tree that can accommodate data in more than one dimension. In the *k-d* tree the descent from one level to the next-lower level corresponds not only to a progressively finer selection of data but also to a change of dimension. At each step the data are divided along the largest dimension. Thus in storing a digitized map of Chile the first division would be along the north-south axis, whereas in a map of Tennessee the first division would be along the east-west axis.

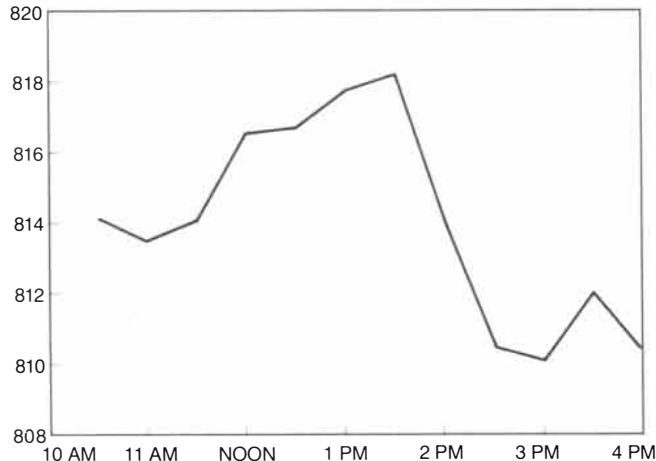
The *k-d* tree is an efficient means of storing two-dimensional data because breaking up the information along its largest dimension reduces the number of decisions that must be made to retrieve an item. The main problem in storing maps, however, is not to reduce the number of decisions in each retrieval but to minimize the number of disk blocks that must be examined. Like the connection matrix, the *k-d* tree cannot store street names with the streets and therefore the names must be supplied from a separate file.

To speed the disk-retrieval operations a "patched file" was constructed that comprises two subfiles, with each subfile holding a different kind of information. One file is the master list of edges, drawn from Census Bureau data and from a table of extra information for each street. The Census Bureau data give the location of each edge and its name; the additional table designates one-way streets and limited-access highways and gives information about speed limits and average travel speed.

In the master segments file, as in the original Census Bureau data, each street is divided into segments short enough so that a segment intersects other segments only at the ends and also short enough so that the segment can be approximated by a straight line. A segment corresponds to a single record in the data base. Such a form of organization implies that for most streets each block requires a separate record. The records are sorted in alphabetical order according to street name.

Each record includes a field that indicates whether the segment is an ordinary street, a limited-access highway, an access ramp or another map feature such as a railroad, a river or a boundary line.

TIME	DOW JONES INDUSTRIAL AVERAGE
10:30	814.12
11:00	813.55
11:30	814.12
NOON	816.59
12:30	816.69
1:00	817.73
1:30	818.21
2:00	814.12
2:30	810.50
3:00	810.12
3:30	812.02
CLOSE	810.41



COMPUTER "The market crept upward early in the session yesterday, but stumbled shortly before trading ended. Stock prices turned in a mixed showing, with the market posting a small loss in moderate trading."

WALL STREET JOURNAL "The stock market finished with mixed results after the attempt to push its rebound into a fourth session faltered in continued active trading."

SOFTWARE FOR THE STOCK MARKET can convey the results of a day's trading in numbers, visual images or words. The Dow Jones Industrial Average is available in machine-readable form; the panel at the top left shows the average at half-hour intervals on June 23, 1982. The panel at the top right shows the output of a simple program that converts the numerical information into a graph of the day's average. A more complex program devised by Karen Kukich of Carnegie-Mellon University gives an English-language summary of the progress of the average through the day. The bottom panel compares a computer-generated summary and the summary of the same day's trading that was published in *The Wall Street Journal*. The stock-market program does not include information about trading on previous days; hence the summary cannot include generalizations about trends lasting for more than one day. The text generator is specialized for the task at hand. For example, it never generates the future tense.

For streets the record also includes the house numbers on each side of the segment, information about travel speed, the locations of the endpoints and the Zip Code on each side of the segment. By means of a binary search through the master file it is a simple matter to retrieve the location of any street address and list the intersections of any two streets. Thus the master file alone can answer the first two types of query.

The second file is a group of segments organized according to a patched format in which the area covered by the map is divided into squares 10,000 feet on a side. To draw a map that includes all the segments in a particular area the program scans the list of patches sequentially and then scans the relevant patches in the same way. A segment that appears in only one patch is stored once, but a segment that appears in more than one patch is stored in all the patches where it has an endpoint. The patched file is entirely derived from the master list. When changes are made, they are entered into the master list only; the patched file is then regenerated automatically.

When techniques for organizing and storing the information from the maps had been selected, the problem of providing an interface between the user and the system had to be solved. Most of the available computer interfaces are inferior to a printed map. On many printed road maps the ratio of the width of the smallest printed character to the width of the map is about one to 1,000. Even on a high-quality computer terminal the ratio of the width of the smallest letter to the width of the screen is generally only about one to 125. Furthermore, most computer devices can print only horizontally; a few can also print vertically but almost none can print at intermediate angles. Because of these limitations many street names must be omitted from the maps that are produced from the digitized information.

To determine which labels should be omitted the program utilizes information about how big the labels are and also about which streets are important. It is assumed that the longer a street is, the more important it is for routing purposes; the assumption works well in practice. Information about the relative importance of streets is also employed to excerpt the maps so that large areas can be represented without excessive detail and to plot routes that traverse only large thoroughfares. For rapid processing it is convenient to assume that each street follows a straight path between the intersections that survive the excerpting.

The system we have constructed can efficiently answer all four types of query given above. When the data base is com-



FUZZY, BLURRY VISION

A simple eye examination can correct today's problems, prevent tomorrow's!

If the world around you seems out of focus, more distant than it really is—or if you're experiencing headaches or dizziness—or squinting, blinking or rubbing your eyes more than usual—it's been too long between eye exams.

Years of training and experience enable your eye care professional to correct most vision problems simply and quickly. And to spot the

beginnings of more serious problems like cataracts or glaucoma.

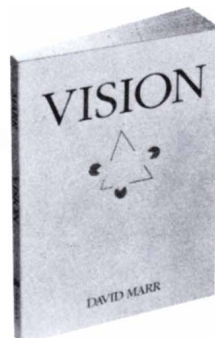
An eye examination can give warnings of other problems, too. Like diabetes, high blood pressure, kidney disease or arteriosclerosis which can be treated medically. So it is important that you and your family have regular eye exams.

Ask about new ways of correcting imperfect vision. Millions now wear soft contacts who couldn't just a few years ago. Including people who have astigmatism—who wear bifocals—or who have had cataract surgery. Call your eye care professional for an appointment today.

©1983

Presented as a Public Service by
BAUSCH & LOMB

Makers of the most prescribed soft contact lenses in the world



VISION
David Marr, *Late of the Massachusetts Institute of Technology*
397 pages, 155 illustrations
ISBN 0-7167-1567-8; paper \$21.95

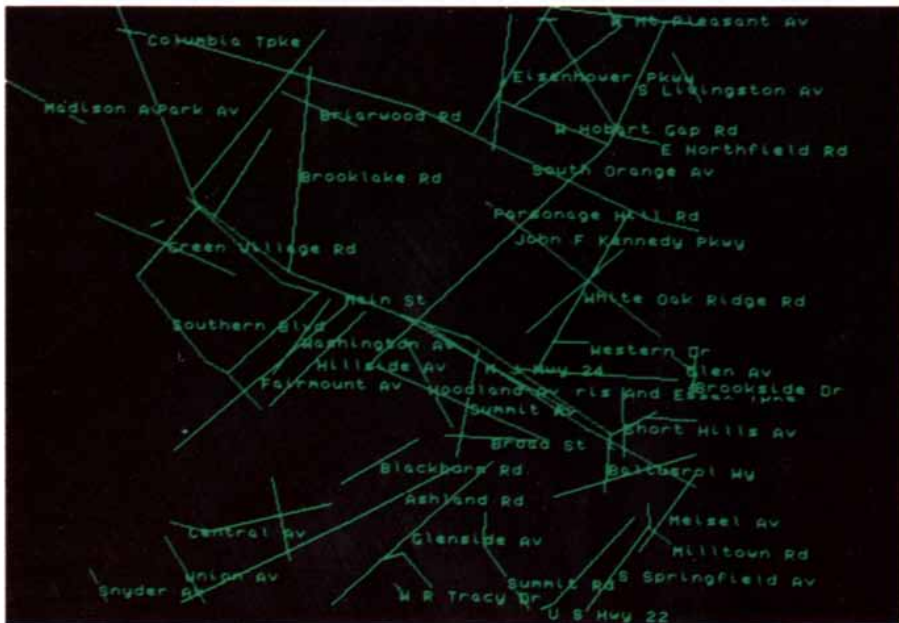
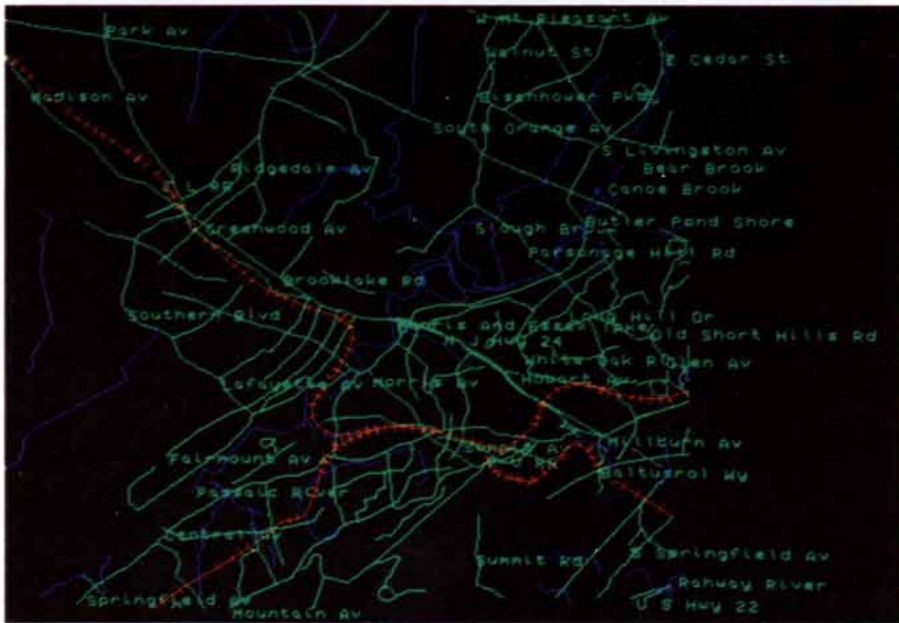
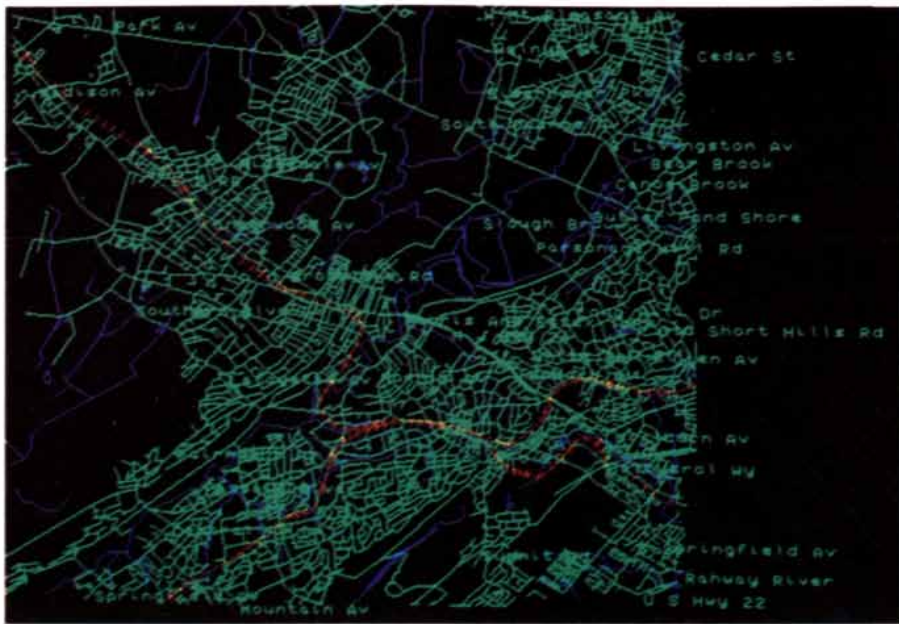
A MILESTONE IN THE HISTORY OF THE SUBJECT*

In **VISION**, David Marr presents an entirely new way of looking at sight. The author theorizes that one can understand what seeing is and how it works by comprehending the underlying information-processing tasks initiated by the eye. **VISION** is an amplification of the landmark computational theory developed by Marr at M.I.T.

"Even if no one of Marr's detailed hypotheses ultimately survives, which is unlikely, the questions he raises can no longer be ignored, and the methodology he proposes seems to be the only one that has any hope of illuminating the bewildering circuitry of the central nervous system."

—H. C. Longuet-Higgins, *Science**

To order, send check or money order to:
■ W. H. Freeman and Company, 4419 West 1980 South
Salt Lake City, Utah 84104



bined with a program written by Jane Elliott of AT&T Bell Laboratories, it is possible to find the shortest route between two points on the map in terms of time or distance. Most of the map processing utilizes the patched-file structure, which is analogous to the bucketed file employed for one-dimensional data. The patched file is somewhat slower than the *B*-tree but it exploits the disk structure better than the *B*-tree does. The list of patches for a single map generally fits in one disk block; hence each retrieval can be done quickly. Moreover, the patched structure is easy to understand, to use and to update.

Few standard routines exist for processing two-dimensional data, but by combining several techniques the map problem can be solved. Many other intriguing examples of data-base management could be cited as a result of the proliferation of devices that generate information in machine-readable form. That increase, however, has not been matched by the development of software to provide access to the information. In the next few years more efficient and more imaginative software for information management will undoubtedly be developed. Although the results of the process cannot be predicted, it is probable that such development will be guided by the principles set forth in the introduction to this article: the need to tailor each program to the content of the information and to how the information will be employed and the need to fully exploit the structure of the machinery in which the software operates.

GEOGRAPHIC DATA BASE stored in digital form generates maps with various levels of detail. Each panel shows the same area: a square four miles on a side centered on an intersection in Chatham, N.J. The map at the top shows all the streets in the square. The map in the middle has been excerpted to show only important streets. The map at the bottom has been further excerpted and also simplified by assuming that each street follows a straight path between the intersections shown. The excerpting can considerably reduce the computer time that is needed for processing a map. The full map requires 56 seconds of processing on a Digital Equipment Corporation VAX 11/750 computer (a large minicomputer). The excerpted map requires 34 seconds and the excerpted, straight-line map requires five seconds. The software for processing the maps relies on a file that comprises two subfiles. The main file includes data in digital form from the Bureau of the Census concerning the location of nodes and edges. A node is an intersection and an edge is the portion of a street that connects two nodes. The second file is a "patched file" in which the map is divided into small squares and the appearance of nodes and edges in each square is recorded. The two subfiles are employed together to excerpt the maps and to answer questions about routes.



**NOMINATIONS REQUESTED
BY 30 JANUARY 1985 FOR THE TWELFTH**

**MARCONI
INTERNATIONAL
FELLOWSHIP**

**TO COMMISSION CREATIVE WORK IN
COMMUNICATIONS SCIENCE OR TECHNOLOGY**

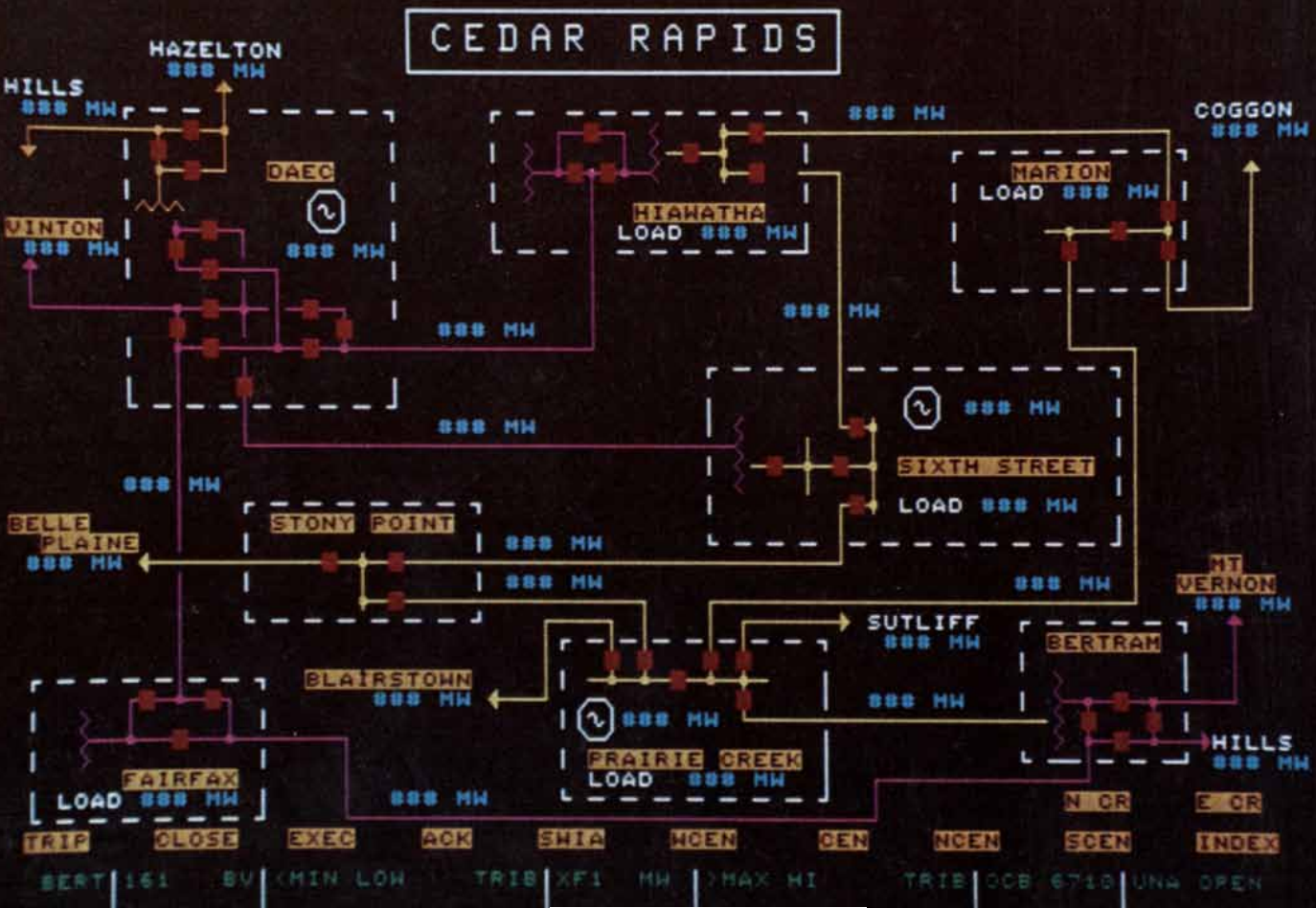
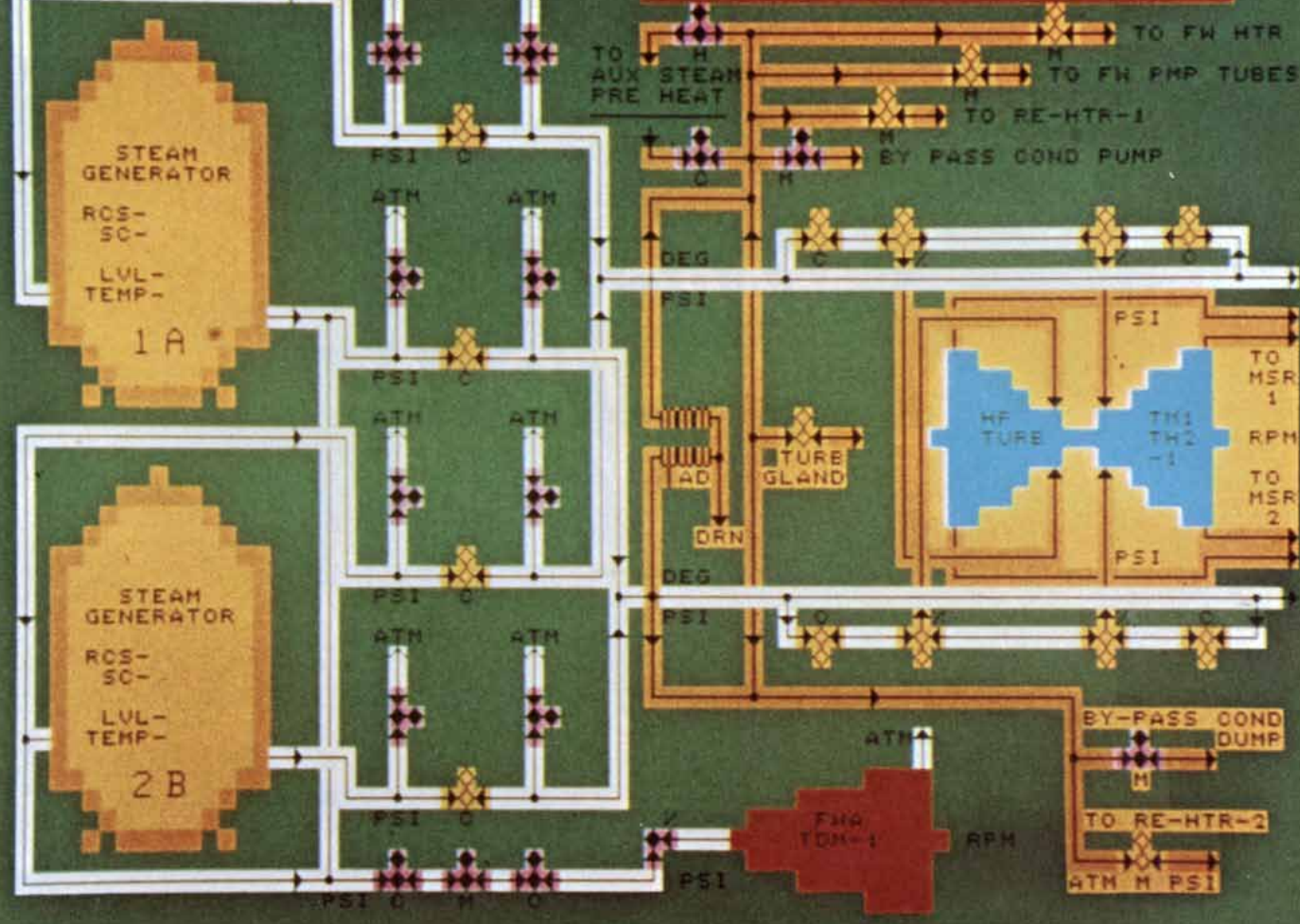
A \$35,000 AWARD

Criteria for Eligibility:

1. The importance of the nominee's contributions to communications science or technology;
2. The present and future value to society of these contributions; and
3. The degree to which the nominee's work exemplifies commitment to the betterment of the human condition.

For Information:

Marconi International Fellowship Council
Polytechnic Institute of New York
333 Jay Street
Brooklyn, New York 11201, U.S.A.
(212) 643-5500



Computer Software for Process Control

Software of this kind has the primary function of communicating with and governing physical devices. A process-control computer does not set its own pace but responds to events in the real world

by Alfred Z. Spector

Computer systems that monitor or control processes in the real world are rapidly growing in number. They deal with such things as air traffic control, the equivalent routing and signaling system for the railroads, the operation of nuclear power plants, the distribution of electric power, the telephone network, the autopilot and other control systems of an aircraft, the operation of elevators, the control of robots and machine tools, the indoor environment of buildings, production lines in manufacturing plants, the flight of spacecraft and so on. One distinctive characteristic of a process-control computer is that its primary function is to communicate with the physical world rather than with a human operator (although it may display information about the state of the process to the operator). Another feature is that a process-control computer cannot set its own pace; it must respond on cue to events in the world at large.

A typical system might be put to work controlling a fractionating column that distills light chemical components from heavier ones, as in a petroleum refinery. In this application the computer, directed by software, receives information on the level and the rate of flow of the various fluids and on temperatures and pressures in the column; it issues commands to control these factors and thereby determines the quantity and quality of the products. The control system might also be programmed to minimize the use of energy in the plant.

Whatever the application is, the links between the computer and the process

are sensors and actuators. Typically a sensor monitors analogue data, such as changes in temperature, that must be transformed into digital data before being presented to the computer. With some types of sensor the software periodically calls for information; with other types it is the sensor that interrupts the software at irregular intervals to present information. A program for controlling a process is also likely to include a timing device—a clock—that can be regarded as a sensor. An actuator manipulates the real-world process either electrically or electromechanically. In controlling temperature an actuator might turn a fan on or off.

The links between the computer and the human operators are input and output devices. A keyboard is the standard input device. Modern computer systems often have additional means of input, such as a light pen or a "mouse," whereby the operator can make choices by pointing at a display screen. The screen itself is an output device, displaying textual and graphic information on the state of the process. Another form of output is an alarm indicating that some part of the process needs attention.

At the heart of a process-control computer is a model of the real-world process. The model has three components that I call the model state, the state-update function and the predictor function. The model state consists of data giving a complete description of the real-world process at each instant. The state-update function transforms one model state into another based on

information supplied by the sensors. The predictor function, if it is given an accurate model state, yields a set of computer commands that achieve some desired condition in the process being controlled. What the formal terms describe is a system of feedback control: the software receives data from the sensors, implements the state-update and predictor functions and issues commands to the actuators. The results of those commands then influence further data received from the sensors.

Separate from the model but also crucial to the operation of the system is a strategic plan. It specifies the sequence of states the controlled process should pass through. For example, in a city traffic-control system the plan specifies the state of the traffic lights as a function of time and traffic flow. The plan may be supplied by human operators or it may be generated by the software from a set of more abstract goals established by the designers of the system.

A fairly simple arrangement for controlling the supply of heat to a building illustrates the structure of a process-control system. The hardware includes a sensor to monitor the outdoor temperature, sensors in several rooms to monitor indoor temperatures, a clock and two actuators, which are switches for a heat pump and a furnace. Assume the software is given two goals: to maintain a temperature that depends on the time of day and to minimize energy consumption.

The model state includes the inside and outside temperatures and the time of day. The most important part of the state-update function calculates a weighted average of the data from the various indoor-temperature sensors. The predictor function utilizes the model state, together with information about the heat loss of the building and the thermal output of the two heaters, to predict when a heater should be started or stopped. The strategy calls for a deter-

ELECTRIC POWER SYSTEMS are monitored by computer software, which displays the state of the system at any given time on the computer screen. In the top photograph on the opposite page the software is reporting the conditions in a generating plant with two steam generators and one turbine. The bottom photograph shows the high-voltage transmission system of the Iowa Electric Light and Power Company. By means of this display the operator at the company's control center in Cedar Rapids can open and close circuit breakers to reroute power among the substations named in the display. The program was installed by Aydin Controls.

mination of whether the furnace or the heat pump is most cost-effective at a particular time; the determination might well be based on the outdoor temperature and the cost of fuel.

One could extend the system by adding more sensors (say to monitor the fuel levels), by taking into account other aspects of the model state and by arranging to give some notice of exceptional conditions (such as a malfunctioning heater or an open window). The fundamental nature of the system would not be changed, however, by these enhancements. It would still be based on a model of the controlled process, and it would still employ a predictor function to reach new states.

Most process-control systems are more complex than this example might suggest. The main reason is the complexity of the internal model. Consider a vehicular-control system, even an elementary one that can sense only acceleration. Just to maintain the correct velocity the state-update function must do a mathematical integration for each reading of the acceleration. If a camera were added to detect obstacles and follow roadways, analyzing the view in order to update the model state would demand the most elaborate techniques of artificial intelligence.

Complex procedures may also be

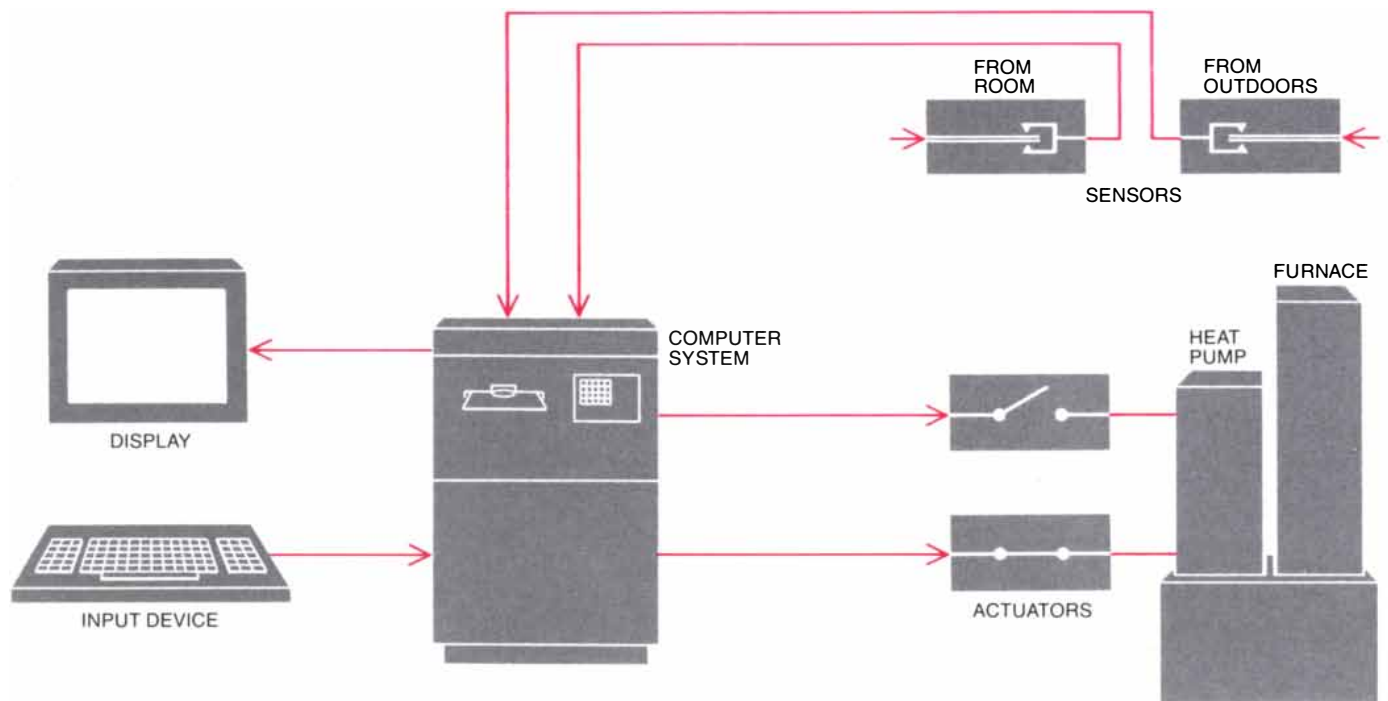
needed in carrying out predictor functions and in developing strategic plans. The predictor function that calculates the angles for a six-joint robot arm in order to position and orient its hand calls for substantial amounts of linear algebra. Planning the sequence of intermediate states necessary for a robot arm to move smoothly from one position to another is even more difficult.

Computation and planning are tasks that must be done in many computer applications, and process-control software employs techniques to do them that are common to other types of software. On the other hand, process-control systems have requirements that differ from those of other computer applications. One requirement has to do with speed. For a computer playing chess or calculating a payroll precise timing is seldom critical; higher speed would be advantageous, but a result obtained after a delay is still valid. A system controlling a jet aircraft, however, has to make decisions quickly; it must act in "real time."

Similarly, the chess-playing or payroll-calculating computer can do one task at a time and can schedule its tasks in whatever way is convenient. The aircraft-control system must meet multiple demands as they are presented, care-

fully synchronizing its work on various tasks. Reliability is also more important in the aircraft, where the consequence of a programming error could be a loss of life and not merely the loss of a game or a financial loss. In many cases the need for speed, synchronization and reliability is complicated by the physical organization of the system, in which computers, sensors and actuators may be spatially separated and operating in a harsh environment.

The problems of synchronization and timing are suggested by the simple program fragment in the top illustration on the opposite page. It is part of a program for controlling the heating of a building, and it takes the form of a loop: a sequence of instructions that can be executed repeatedly. The only timing requirement is that the procedures must be carried out fast enough to allow for a specified frequency of operation. If there were multiple sensors to be sampled or actuators to be commanded, however, the control flow through the software would be far more complex. Furthermore, if the system had to handle asynchronous interruptions from sensors and to issue commands in response to such events, the software could no longer be organized as a loop or even as a set of loops but would need to have some more complex topology.



PROCESS-CONTROL SYSTEM for regulating the supply of heat to a building is shown schematically. The system consists of two heaters (a furnace and a heat pump), an array of sensors to monitor the outdoor temperature and the temperature in the rooms of the building, actuators to turn the heating units on and off, a computer and a program given two goals: to maintain a temperature that depends on the time of day and to minimize the consumption of energy at all times. The program operates according to a three-part model of the

real-world situation. The model state includes the inside and outside temperatures and the time of day; the state-update function calculates a weighted average of the various temperatures and revises the model state accordingly, and the predictor function takes into account such matters as the state of the system and the rate of heat loss from the building to predict when one of the heaters should be turned on or off. The computer program is based on a strategy of using only the more economical heater unless both heaters are needed.

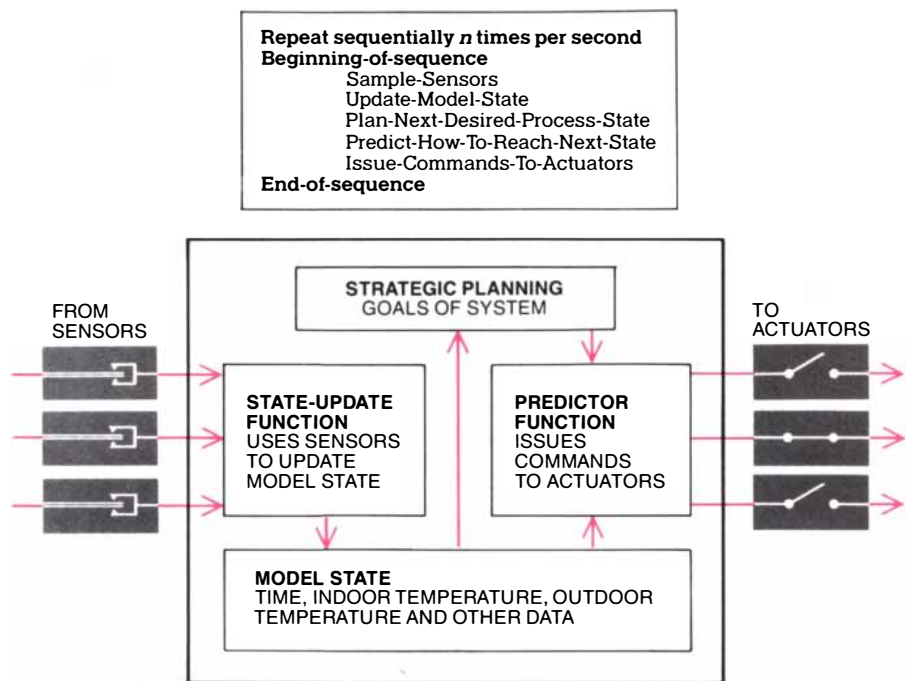
Process-control software is usually constructed as a collection of cooperating but separate tasks. A task is defined as an independent sequence of computer instructions that can call on data at least partially segregated from the data relating to other tasks. Multiple tasks can be carried out on multiple processors, with each processor executing a single task. A commoner arrangement, however, employs a multitasking operating system to schedule the execution of many tasks on a single computer. Careful analysis is needed to ensure the tasks receive service in a way that meets the overall goals of the system.

One simple technique schedules tasks in a round-robin way: each task gets a turn during which it is executed to completion. With this technique a task may meet with severe delays if long tasks are scheduled ahead of it. A second possibility is a preemptive round-robin technique, in which each task gets only a short time to execute before the processor turns to another one. If there is more work for the first task to do, it gets another period of execution on its next turn. A third approach is priority-based scheduling, in which tasks of higher priority get longer or more frequent periods of execution. A final important approach is called deadline scheduling. A deadline by which each task must be completed is established, and the system attempts to schedule the tasks so that they all meet their goals.

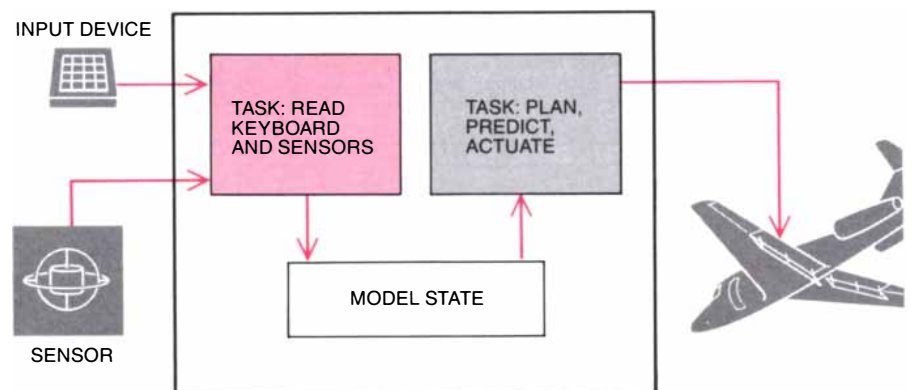
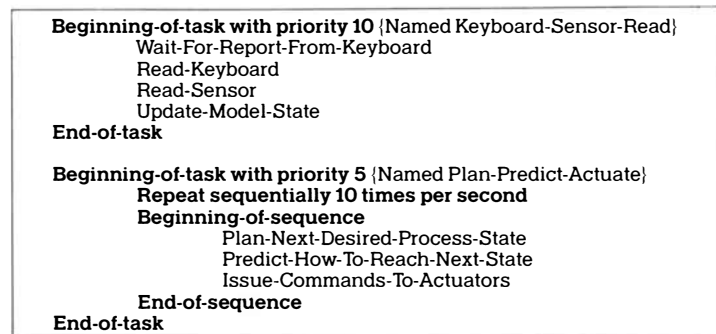
A fragment of a simple multitask control program is shown in the bottom illustration at the right. Two tasks are coordinated: Keyboard-Sensor-Read, with a high priority, and Plan-Predict-Actuate, with a lower priority. The high-priority task receives input asynchronously (that is, at unpredictable intervals) from a human operator and from sensors; the other task periodically adjusts actuators in response to recent inputs. If the low-priority task is executing when the keyboard or a sensor reports that new data are available, the operating system interrupts the task and allows it to resume only after the data have been read. Almost all process-control systems have this multitask organization.

It is significant in this arrangement that the Keyboard-Sensor-Read task transfers data to the Plan-Predict-Actuate task by way of the model state. The first task simply loads information from the keyboard into the area of memory where the model state is stored, an area to which the second task also has access. The sharing of memory in this way is a powerful and efficient technique for intertask communication. It is the standard procedure for process-control applications confined to a single computer.

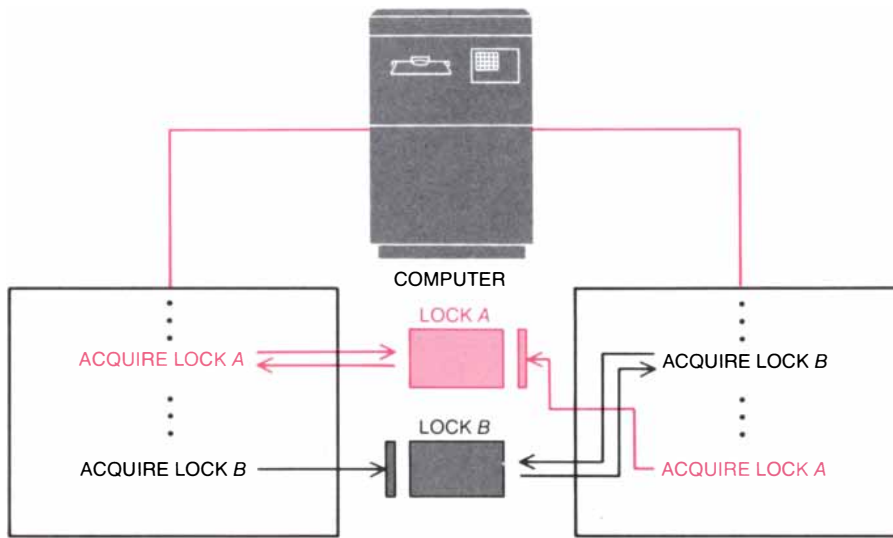
On the other hand, communication between tasks by means of shared memory complicates multitasking and



SOFTWARE STRUCTURE of the process-control scheme for the heating system is suggested by the program fragment in the upper part of this illustration. The commands in boldface type are key words in the programming language; the other commands are calls on software procedures that are specific to the system being controlled. The program has the form of a loop: it is a sequence of commands that can be executed repeatedly. The lower part of the illustration shows the major components of the computer program for controlling the heating system.



MULTITASK CONTROL SYSTEM is based on a single program that attends to and coordinates several functions; in the example shown here it operates a remotely controlled aircraft. A fragment of the program (*top*) includes two tasks; one task has a high priority (*color*) and the other a lower priority (*gray*). The high-priority task receives inputs at irregular intervals from a human operator, who works with a keyboard, and from sensors on the aircraft. The low-priority task periodically adjusts actuators that operate the aircraft's controls (here a wing flap). The high-priority task cannot be interrupted. If the low-priority task is executing when inputs come from the keyboard or a sensor, it is interrupted by the program. Both tasks have access to the representation of the model state in the computer's memory. Memory sharing is standard for intertask communication in process-control systems directed by one computer.



PROBLEM OF DEADLOCK can arise in a program that employs the technique called locking to synchronize the sharing of memory by multiple tasks. In this technique a task cannot gain access to data in memory until it acquires a lock, meaning that it asks the operating system for access and receives it. When the task no longer needs access, it releases the lock. Here deadlock is shown in a two-task system. The task on the left is executing a sequence of instructions (represented by dots) when it encounters a command to acquire lock *A*. It does so and continues executing. Meanwhile the second task has followed instructions to acquire lock *B*. Now the first task issues an instruction to acquire lock *B*; it cannot do so because the other task already has that lock, and so the first task is suspended until the lock is made available. If the second task now issues a request to acquire lock *A*, the system is deadlocked and neither task can execute.

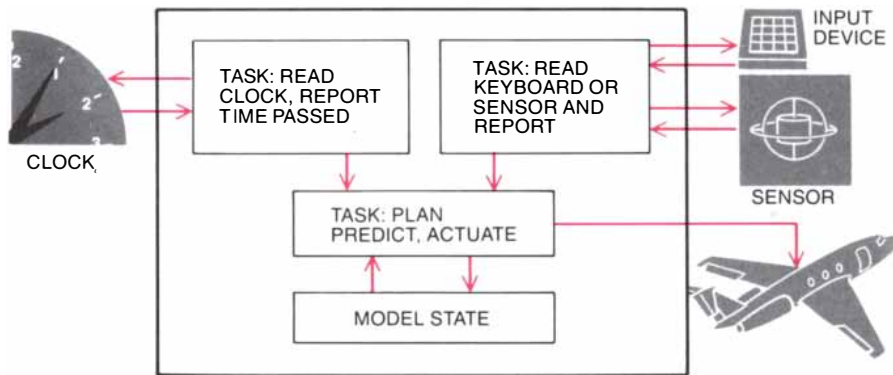
```

Beginning-of-task with priority 10 {Clock}
Repeat sequentially 10 times per second
Beginning-of-sequence
  Send-Message (Plan-Predict-Actuate-Task, Time-Passed)
End-of-sequence
End-of-task

Beginning-of-task with priority 10 {Named Keyboard-Sensor-Read}
Wait-For-Report-From-Keyboard
Read-Keyboard
Read-Sensor
Send-Message (Plan-Predict-Actuate-Task, New-Data)
End-of-task

Beginning-of-task with priority 5 {Named Plan-Predict-Actuate}
Repeat sequentially 10 times per second
Beginning-of-sequence
  Await-Message
  If message is from Timer-Task then
    Beginning of sequence
      Predict-How-To-Reach-Next-State
      Issue-Commands-To-Actuators
    End-of-sequence
  Else Update-Local-State-With-New-Data
End-of-sequence
End-of-task

```



MESSAGE PASSING is an alternative to memory sharing for synchronizing the work of multiple tasks in a process-control system. It is the only workable method when the system has several separate computers. Part of a message-passing program for controlling an airplane appears at the top; the organization of the program is diagrammed below the program fragment.

leads to some interesting problems. One problem is synchronization. Suppose in the two-task program the low-priority task is interrupted after it has read some elements of the model state but before it has read others. When the task resumes, it reads the remaining values and calculates a result determined by all the readings. During the interruption, however, the model state may have been changed by the other task, so that the computation is based on a chimera of old and new data. Suppose the interrupted task was measuring a position in an *x-y-z* coordinate system and was interrupted after it had read *x* but before it had read *y* and *z*; it would base its calculation on a position that never existed in the real world.

Several techniques are available for synchronizing access to shared memory, but two simple ones are commonly used. One technique depends on designating segments of program code in which a task cannot be interrupted; the facilities for interrupting a task can readily be disabled by the operating system in conjunction with the computer hardware. Turning again to the two-task example, the system might specify that the Plan-Predict-Actuate task is to be free of interruption while it is reading the *x*, *y* and *z* coordinates. The other major technique is called locking. Before a task can gain access to data it must ask the operating system for the right to do so. When the task has finished with the data, it tells the operating system that the data are free for the use of other tasks. The task is said to acquire a lock before it can read data in shared memory, and it must release the lock when it has finished. By this mechanism the operating system ensures that only one task at a time has access to the data.

Although the synchronization of access to shared memory is conceptually simple, it is a major source of bugs in multitask programs. In part the reason is that synchronization bugs are extremely difficult to find; they may exhibit no symptoms except in rare circumstances. The computer malfunction that delayed the first mission of the space shuttle was a synchronization problem (a highly complex one) that would be likely to arise only once every 65 times the system was started up.

Problems connected with the synchronization of shared memory can affect the reliability of a system in other ways. What happens if a programming error leads a task into an endless loop while it is in a no-interruption period? Unless great care has been taken in the design of the system, other tasks may be shut out entirely. A problem that can arise with locks is that a collection of tasks may form a cycle in which each task is waiting for another task to release a lock, with the result that none of the tasks is executed. Such a deadlock can

Artificial intelligence is the focus of a new advanced technology center at Hughes Aircraft Company. The facility brings research and development efforts under one roof. Scientists and engineers will work closely with universities throughout the country to develop software and equipment to build the so-called expert systems. Studies will center on knowledge representation, symbolic reasoning and inference, natural language processing, and knowledge acquisition and learning. Technology will be developed for image understanding for missile targeting and geological surveys from space, smart avionics to reduce pilot workload, self-controlled systems, simulation and training, fault diagnosis and maintenance, and manufacturing resource allocation and planning.

The sights, sounds, motion, and urgency of combat await pilots who learn to fly the F/A-18 Hornet strike fighter in the first computerized simulators of their kind. A pilot wears full flying gear and sits in an exact replica of an F/A-18 cockpit located inside of a 40-foot-diameter sphere. High-resolution pictures of earth, sky, and targets are projected onto the inner surface of the sphere and matched with appropriate sounds and vibration. Pilots thus experience runway vibration, aircraft stalls, buffeting, missile launches, cannon fire, dazzling aerial maneuvers, and enemy aircraft and missiles approaching at supersonic speeds. The Hughes simulator will save the U.S. Navy and Marine Corps millions of dollars by providing combat training without costly flight operations.

Computers will be troubleshooting hybrid microcircuits used in new sophisticated missiles at Hughes. Computer-aided troubleshooting (CATS) will cut troubleshooting time, improve effectiveness by automatically locating faults, eliminate mistakes and wasted time, and simplify the technician's decision-making. CATS also will be able to use past repair records as a key to speed up troubleshooting. A typical case: An automatic bar-code reader identifies a failed part and data about the failed test is retrieved from a main computer. A probe then automatically takes measurements at key internal circuit nodes so the fault can be isolated. Next, the computer displays a schematic of the failed circuit area and compares actual and ideal signal values. The technician then determines which component has most likely failed and selects rework instructions accordingly.

Development times for semicustom very large-scale integrated (VLSI) circuits have been cut from greater than one year to 20 weeks at an ultramodern computer-aided training and design center at the Hughes facility in Newport Beach, California. Utilizing advanced design automation software, a comprehensive library of predesigned logic functions (called Macros), and preprocessed wafers, the new facility is helping engineers design chips with 2,000 to 8,000 gates and with as many as 180 pins. New 3-micron dual-layer metal HCMOS processes are applied to both standard cell products and state-of-the-art gate arrays. Skilled design engineers and education specialists at the Newport Design Center provide training and technical support for IC designers throughout the company.

Hughes Research Laboratories needs scientists for a spectrum of long-term sophisticated programs. Major areas of investigation include: masked ion beam lithography, liquid-crystal materials and applications, submicron microelectronics, ion propulsion, computer architectures for image and signal processors, gallium arsenide device and integrated circuit technology (analog and digital), and new electronic materials. For immediate consideration, please send your resume to Professional Staffing, Hughes Research Laboratories, Dept. S2, 3011 Malibu Canyon Road, Malibu, CA 90265. Equal opportunity employer. U.S. citizenship required.

For more information write to: P.O. Box 11205, Marina del Rey, CA 90295



"Excuse Me,



Make Way For Hayes' Please.

An advanced, easy-to-use data management system for the IBM® PC and compatibles.

Want to get your paperwork out of a clumsy file cabinet and onto your PC's screen, where you can manage it better? Frustrated with data base software that's either too limited or too difficult to use? Hayes offers you a simple word of kindness.

Please™. A powerful, yet easy-to-use, system for organizing and managing your information. *Please* is flexible enough to store any data you enter, and it'll return data to you in exactly the form you need. *Please* does more.

It does it all faster. And it's sure to please!

"The menu, Please?"

Menus list all your options and tell you exactly which keys to press for every *Please* feature.

That's to be expected. As the telecomputing leader, Hayes built its reputation on quality design, reliability

and customer support. Now these same standards have been applied to a new data management system that is going to instantly change the way you do business!

Say you're looking for an efficient way to maintain sales data. *Please* leads you every step of the way in creating a sales database that might include names, addresses, dates and figures. These categories are called "fields" in database lingo, and they're the very heart of your database structure.

Want last month's total in a particular region? Press a few keys and it's yours! A few more keystrokes and you'll know who's moving product, and what's your biggest seller.

Please will supply you with labels for a mailing to selected customers. It can send customer information to your word processor for a promotional letter. And it can receive data from

your spreadsheet program. *Please* will even look up a name and company for you, your Hayes Smartmodem* will dial the phone number, and you're ready to talk!

Taking this same sales database, you might also want to define special

fields for a custom Output Plan. With a defined field for "COMMISSIONS DUE,"

Please can automatically compute each salesman's commissions, and print them out in a report of your own design. All this and more, just for saying "Please!"

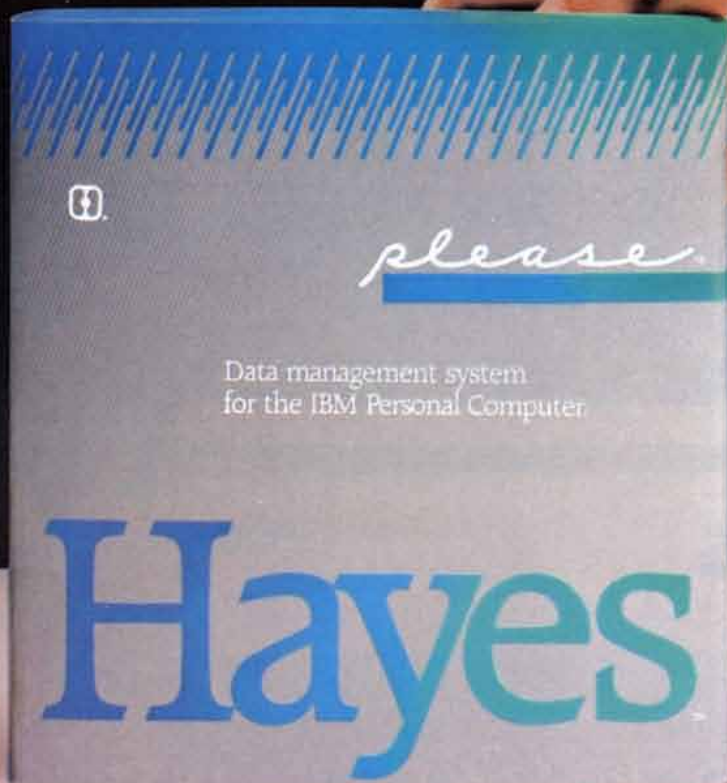
And if you ever change your mind and want to change the structure of your database, please feel free. Step-by-step instructions show you how.

You have the same flexibility with any database you and *Please* design. You can store up to 16 million records and 200 custom Output Plans for each database! More than you're likely ever to require. But isn't it nice

"Make it snappy, Please!"

Need a report fast? You and *Please* can put together a Quick List in a matter of seconds.

Please™



“Put it here, Please!”

Design a special screen format to position data in a particular place.

Just in case you ever need it?

Now you might think that a data management system that does all this must be difficult to use. Right? Rest assured. *Please* works hard so you don't have to. An easy-to-follow sample disk shows you everything you need to know to create your first database. Three *Please* menus show you which keys to press to access every feature. And whenever you need it, *Please* provides on-screen HELP messages, tailored to a specific task. So you needn't waste time reading through a list of unrelated instructions on your screen. Or stop what you're doing to consult a manual. In no time at all, and with no assistance at all, you'll be a *Please* database pro!

knowing all that storage power is there?

“Merge these, Please!”

Combine data from one database into another, without changing your original.

Everything about *Please* is designed to save you time and effort. So what could make data management even easier? *Please Application Templates*, that's what!

To help you get up-and-running immediately, we've developed a series of practical, pre-designed templates. You'll appreciate their well-thought-out structure, and “fill-in-the-blank” ease. Choose several! For business and personal use.

Including *Mailing List*, for storing names and addresses and producing mailing lists. *Contacts*, for man-

aging facts and figures about your sales contacts. *Applicants*, for following applicants throughout the interviewing process. *Appointments*, for maintaining your calendar and tracking all of your business expenses. *Household Records*, a complete home management system. And more! Your dealer has details!

Buy *Please* now! Get a FREE *Mailing List* template from your dealer.

Second FREE template of your choice, direct from Hayes!

Help yourself. *Please!* And take advantage of these two valuable offers. See your dealer right away!



Hayes

Hayes Microcomputer Products, Inc.,
5923 Peachtree Industrial Blvd.,
Norcross, Georgia 30092. 404/441-1617.



WHAT A CLOCKMAKER IN 18TH CENTURY ENGLAND TAUGHT US ABOUT MAKING QUALITY PERSONAL PRINTERS.

Each gear finely honed. Each pendulum carefully balanced. Each timepiece a combination of precision and function. He was a craftsman, building a quality product to stand the test of time.

Today, Okidata makes the most technologically advanced dot matrix printers the same way. With quality, performance, and a healthy respect for value.

Function with affordability. All printers print, but an Okidata does it with more performance and versatility than any other. There are seven models to choose from, priced at \$349 to \$2,595.

Affordability with flexibility. Okidata offers three print modes, too. The data processing mode lets you print up to 350 characters per second. That's five pages a minute. Another mode lets you print emphasized or enhanced text for more effective presentations, while the correspondence mode prints letter

quality at up to 85 characters per second, with clarity that rivals daisywheel. And an Okidata can print graphics and charts, which a daisywheel can't. This allows you to fully use the latest integrated software packages like Lotus 1-2-3™ and Symphony™.

Flexibility with compatibility. Each Okidata printer is fully compatible with all popular software packages and personal computers. Special interfaces are available for IBM and Apple, including the Apple Macintosh.

Compatibility with reliability. Here's where Okidata quality really shows. With a warranty claim rate of less than 1/2 of 1%. With printheads that last well beyond 200,000,000 characters and come with a full year guarantee. With service available nationwide through Xerox Service Centers.

Precision and performance. Quality

and value. That old English clockmaker would have been very proud of us.

Call 1-800-OKIDATA (609-235-2600 in New Jersey) for the name of the Authorized Okidata Dealer nearest you.



OKIDATA
an OKI AMERICA company
Technological Craftsmanship.

Lotus 1-2-3 and Symphony are trademarks of Lotus Development Corp.

appear in a system with as few as two tasks and two locks.

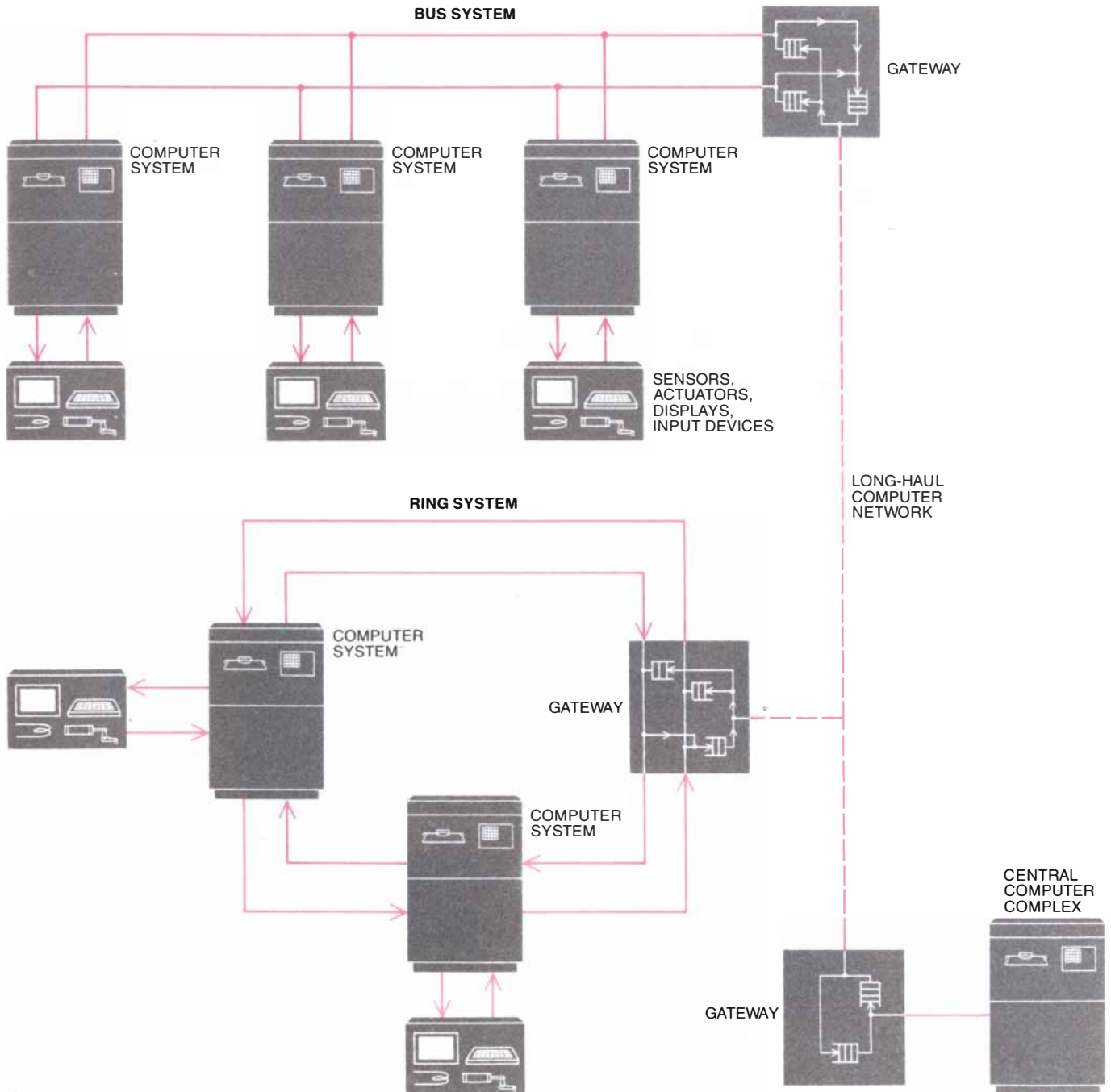
Difficulties unrelated to synchronization are also possible in a shared-memory system. Because the memory is shared, tasks are not isolated from one another, and it is difficult to limit the effects of a failure. The faulty execution can alter the model state and thereby disrupt the operation of many other tasks. In a network of geographically dispersed computers the sharing of memory becomes awkward for still an-

other reason, having to do with the hardware rather than the software: it is simply not feasible to build a memory to be shared by several distant computers.

For these reasons among others some process-control systems are organized as a collection of tasks that do not implicitly share the model state but instead explicitly exchange information by transmitting messages. Each task keeps track of those elements of the model state it needs in its own, non-

shared, segment of memory. The operating system provides facilities for sending and receiving messages.

For simple process-control applications the message-passing arrangement is more complex and usually less efficient than shared memory. Still, the explicit flow of information between tasks and the greater isolation of tasks offer certain advantages. A message-passing organization is the only feasible choice when a process is being controlled by a system of several cooperating comput-



DISTRIBUTED COMPUTER SYSTEM is a common arrangement when the sensors and actuators of a process-control system are widely scattered. The system shown encompasses three clusters of computers that are geographically dispersed. One cluster is made up of three computers that communicate with one another by means of a

local-area network organized as a "bus" (a set of parallel straight-line conductors). Another cluster employs a ring-shaped network. The third component of the system is a central computer that has the functions of coordination, the collection of data and human input. The three systems exchange messages over a long-distance network.

ers without shared memory. Notwithstanding the virtues of message passing, a program can still become deadlocked because tasks can be waiting to receive messages from one another.

Systems made up of several computers working in concert are increasingly prevalent, not only for process control but also for scientific calculation, data processing and artificial intelligence. In some arrangements of this kind a number of processors share a common memory; the sharing is possible, however, only if the computers are physically close. (Transmitting a signal one kilometer takes several microseconds, which is an intolerable delay for the heavy traffic between a central processor and its main memory.)

When the computers are physically separated, the process-control system is organized as a collection of processors with their own local memory; the processors are connected by communication channels to form a network. These distributed systems, as they are called, necessarily rely on message passing. The distance between processors, sensors and actuators distinguishes different types of distributed systems because of the effect of distance on bandwidth and latency. Bandwidth is a measure of the number of bits that can be transmitted per second; latency is the delay between the dispatch and the receipt of informa-

tion. Shorter distances make for higher bandwidth and lower latency, thus allowing closer interaction between tasks being executed on the computers.

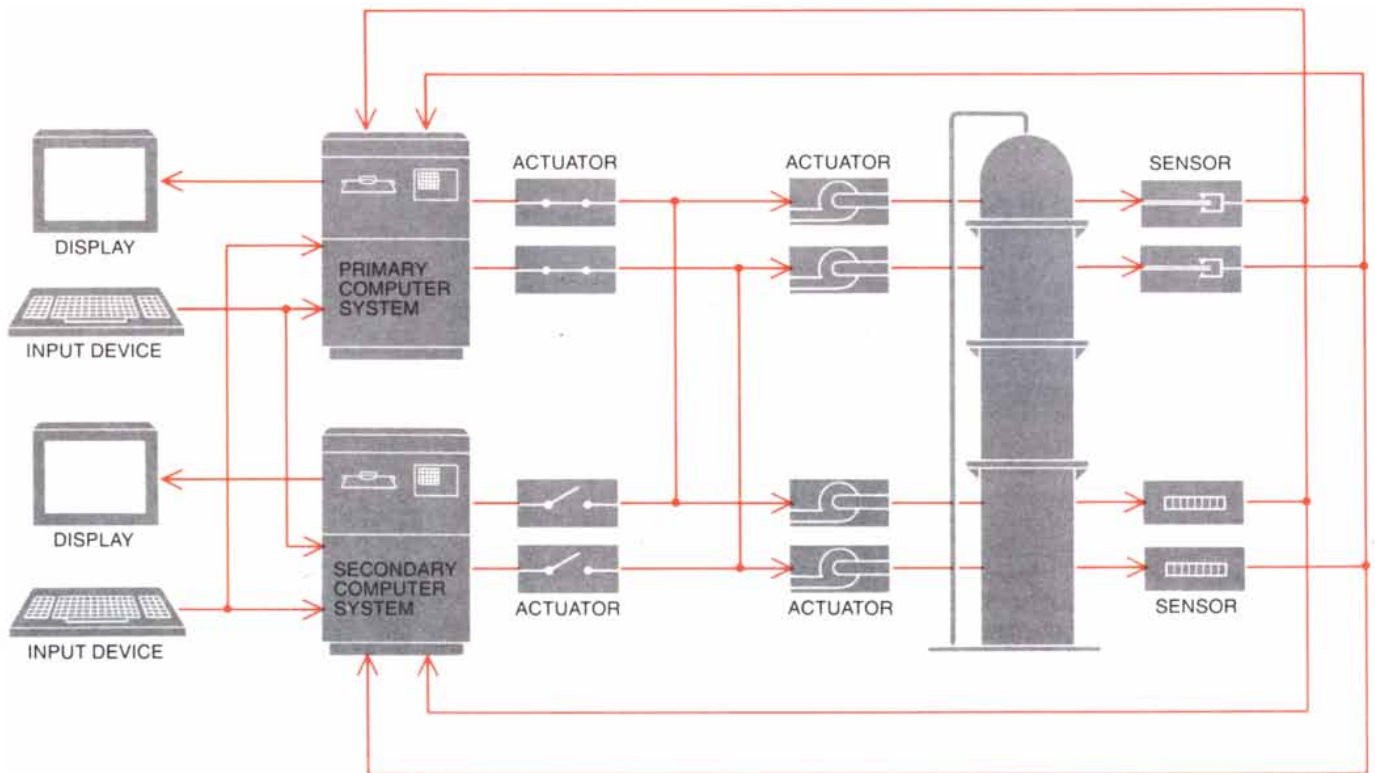
A common reason for setting up a distributed system in process control is that the sensors and actuators are themselves geographically distributed. In such a situation it is often possible to organize the system so that most of the processing is done near the relevant sensors and actuators and communication between computers is minimized. An example is a process-control system for a large factory with many semiautonomous computers in different buildings. The systems occasionally exchange messages for the purpose of plantwide coordination and scheduling, but mainly they operate independently. The system in one building might independently control the manufacture of a certain product but would communicate with other systems to gather information about production quotas or the supply of raw materials.

Distributed processing can also simplify the design and installation of the system (say by reducing the amount of cabling), and it encourages an organizational structure in which the people responsible for a process also have charge of the associated computing tasks. Potentially higher performance is another major motivation for multicomputer systems. In principle more computa-

tion can be done if many processors are working simultaneously. A process-control system organized as a set of tasks that communicate by means of messages is a natural application of such parallel processing. A computationally intensive planning task might be executed on a high-speed processor that receives data from tasks running on other computers and passes strategic plans back to them.

Perhaps the most important reason for adopting distributed computing in a process-control system is that it is a useful means of achieving reliability. When the system is divided into subsystems that operate autonomously, a failure in one machine should not cause the entire system to fail. The factory setup I described above offers an example: even if one subsystem fails, the other subsystems can continue, at least until they run out of raw materials.

If continued operation of the entire system is the objective, redundant processing is a necessary but not a sufficient means to that end. The entire process-control system from sensors to actuators, not just the computer and its software, must go on working in spite of a malfunction. A computer working correctly but receiving incorrect data from sensors and therefore issuing faulty commands to actuators could do more



PROTECTION AGAINST FAILURE is more important in process-control systems than it is in many other computer applications. One way to improve reliability is to duplicate key components of the system. Here one computer is designated primary and the other secondary. Each computer receives all the inputs, but only the primary

is connected to the actuators. The secondary merely does the computations as if it were controlling the process. If the primary fails, control of the actuators is switched to the secondary. The system also has redundancy in the sensors, the actuators, the displays and the input devices, since they must operate as reliably as the computers.

damage than a computer that stopped completely.

The highest goal in the quest for reliability is continued operation completely unaffected by failures. The nature of the compromises made when that goal cannot be attained is suggested by the approach to reliability taken in the control systems of the space shuttle. Because of redundancy in the shuttle's primary spaceborne computer system, a single failure does not force changes in the mission. A second failure does not jeopardize the crew or the vehicle, but the shuttle is brought back to the earth as soon as possible because further failures could be hazardous. The shuttle is said to be "fail operational, fail safe."

If an entire system cannot be kept running after a failure, it may still retain partial function, a property given the rather stately name of graceful degradation. For example, a system that can no longer control a process automatically might still accept keyboard commands from an operator, so that the process could be controlled manually. If even partial operation is not possible, the control system should at least bring about an orderly shutdown of the process in the event of a major failure.

Failures of software can occur in three ways. First, the requirements of the process-control system may be misstated, so that even if the software meets its specifications exactly, it leads to erroneous operation. On the first mission of the space shuttle insufficient knowledge of the vehicle's flight characteristics led to an ascent trajectory that might not have allowed an emergency landing in Spain as part of the plan for dealing with crises. Second, the logical design underlying the software or the programming statements in which the design is embodied may not meet the specifications. The flaw may be something as simple as a typographical error. Third, a human operator may err in using the software.

Protecting against failure is a key topic in all computer applications, but in process control the high probability of bugs in synchronization and timing and the potentially greater cost of such bugs make the task of protecting against them both more important and harder. Sometimes formal analytical methods can be brought to bear to prove that a program meets the requirements stated for it. In most cases, however, less formal analytical methods must serve. A common method is to have experts in requirements and experts in programming jointly study a specification of requirements and a program devised to meet them.

Regardless of how thoroughly a program is analyzed, it is still necessary to test it. Analytical methods are generally not powerful enough to catch all the conceivable bugs in software design

and in programming. Furthermore, no amount of comparison of programs with requirements can ensure that the requirements themselves are correct and sufficient.

Sometimes the people who verify programs are organized as a group separate from the programmers, and they independently test the operation of the program. During the development of the primary spaceborne operating system of the shuttle the independent group of program verifiers was about the same size as the programming group. In an effort to reduce the risk of programming errors still further, two programming groups are sometimes formed to independently devise software for the same task. The assumption is that at least one version of the software will work.

Protection against errors by operators calls for what is irreverently termed idiot-proofing. Careful design of the software can minimize the likelihood of such errors. Input from the operator can also be checked for plausibility, for example by having the program determine whether numerical values are in the proper range. Another technique is to give the operator an opportunity to rethink crucial actions. The program can raise a question such as "Did you mean to halt the process?"

An interesting property of software is that once a program has been proved to operate correctly, it will continue to work indefinitely; software does not "break down." In contrast, hardware that is reliable at one time may not be reliable later. Processors may stop altogether or compute erroneous results; memory and sensors may return incorrect values; communication lines may garble information or lose it, and actuators may stop working or become inaccurate. A single component might fail or an entire computer with its memory and communication lines might be disabled, as in an airplane after a fire. Even if the computing system itself is flawless, it may still fail because of environmental effects such as power fluctuations or excessive heat. If a system is to be reliable, therefore, it should tolerate the faults that still arise in spite of all efforts to eliminate them. Redundancy in many forms is the means of keeping such faults from interfering with the system's planned reliability.

A certain amount of redundancy can be built into the hardware. For example, many computers store extra information with each item in memory so that some storage errors can be corrected automatically. Some processors automatically retry an instruction that has failed, a design feature founded on the reasonable likelihood that the effort will succeed the second time. Major logic modules—even entire processors—can be replicated. The results obtained by

the units operating in parallel are compared, and the result accepted is determined by a majority vote. The technique is called n -replica modular redundancy. When n is more than 3, the technique yields a correct value if there are no more than $(n - 1)/2$ failures. As long as modules fail independently the reliability of the system increases as n increases.

Software techniques are also employed to gain reliability. Like hardware, software can retry a procedure after a failure. In most communications systems the software retransmits data until it has received an acknowledgment that the data got through. Software can also turn to a redundant source after one source has failed. In a typical operation the software would detect a malfunction in an actuator by noting inconsistencies in data coming from a sensor; commands would then be redirected to another actuator.

In a multicomputer system similar software procedures can be carried out on multiple processors to diminish the consequences of a processor failure. The software for each processor can be programmed independently to increase the likelihood that at least one version of the software is correct. If the system has autonomous processors that are separately powered, spread over a distance and connected by redundant communication channels, there is little chance that the entire system can fail. With the addition of redundant sensors and actuators a system of this kind can provide extremely high reliability.

One way of organizing a system with extensive redundancy is to designate one computer the primary, with the other computers being considered secondary. The primary computer receives data from the sensors and commands the actuators. The secondary computers may also receive data from the sensors so that they will have the correct model state if one of them has to take over from the primary, but they do not command the actuators. The secondaries check the primary computer periodically, both implicitly by keeping watch on the consistency of the data from the sensors and explicitly by sending messages to the primary that ask it to perform some test function. Both the primary and the secondaries must filter information from redundant sensors, and they may have to vote on the readings from the sensors in order to ensure that the computations are based on valid data.

Perhaps the most difficult part of this technique is reliably determining when a primary computer has failed. It is easy to envision a situation in which a malfunctioning secondary could seize control from a properly functioning primary. Clearly it is necessary to have an agreement among multiple secondaries on the necessity for one of them to take over from the primary. Whenever there

is enough time, it is simpler for a human operator to make the decision.

An alternative way of organizing a multicomputer system is based on voting. The procedure is similar to *n*-modular redundancy except that the voting is done by software modules rather than by hardware. The modules perform calculations, exchange results with one another and vote on the results. Because independent processors run at slightly different speeds, a group of them must wait until the last one has finished a computation before they can vote.

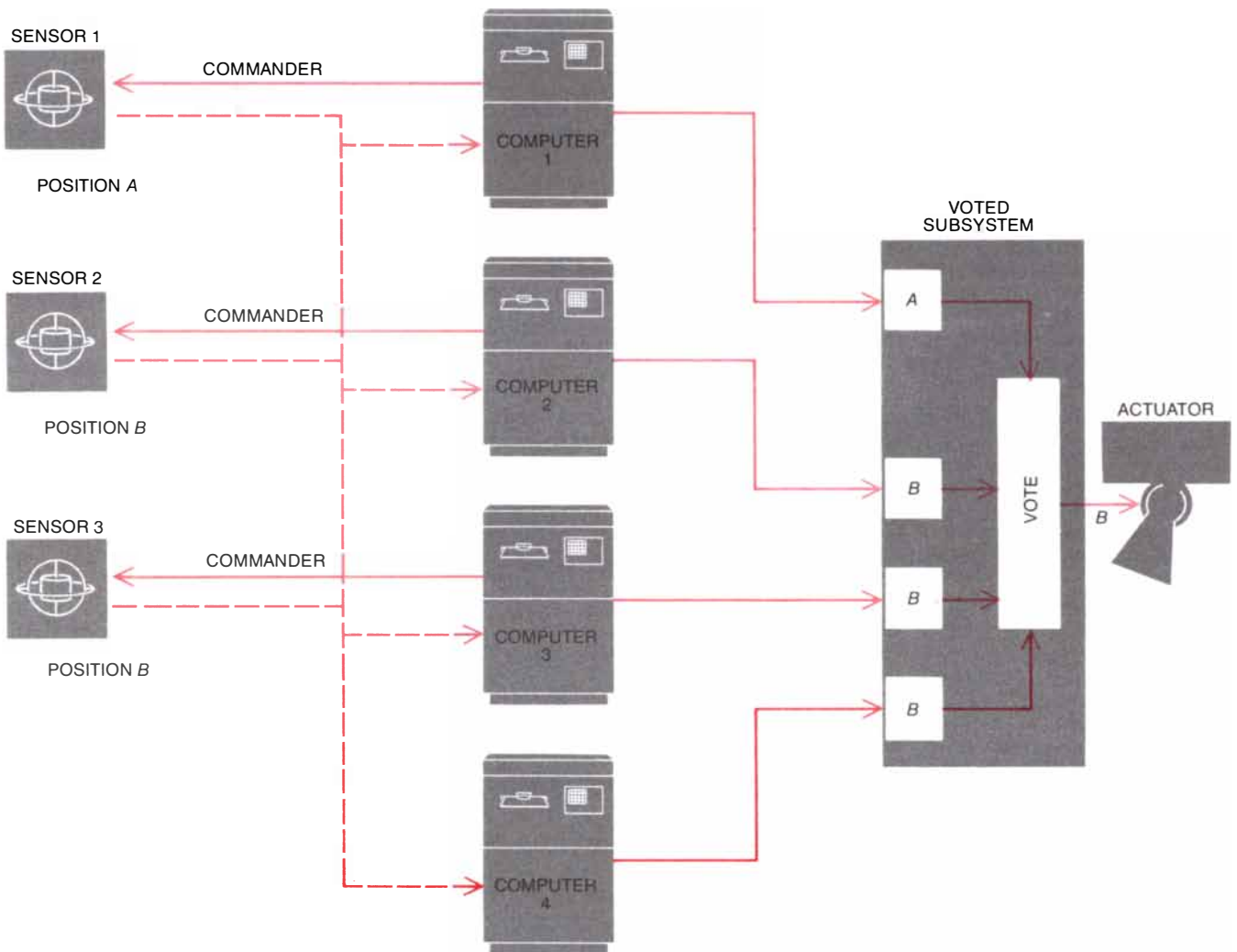
A system somewhat like the one described controls critical flight segments on the space shuttle. The system has four autonomous processors. Each processor commands a separate actuator for each function, such as moving airfoil surfaces during flight in the atmosphere. The voting is done hydraulically:

three actuators can overpower one actuator. A description of hydraulic actuators that fight against one another may seem out of place in a discussion of computer software, but it is quite germane. The hydraulic voting scheme exists because process-control software, by its very nature, must directly influence processes in the real world. It is this direct control that most distinguishes process-control systems from other computer systems, which merely report the results of computations.

Some process-control systems are inherently quite simple. The heating-control arrangement I have described is an example. Other systems, such as the ones that control the shuttle, jet aircraft and telephone switches, are among the most complex computer systems ever built. They demand advanced strategic planning, high performance, high reliability and precise timing. Meeting

those objectives calls for the use of techniques from the fields of software engineering, hardware architecture, systems design, operating systems and artificial intelligence.

A simple system can be designed in a matter of days. A complex one such as the onboard system for the space shuttle calls for years of effort by thousands of workers. Perhaps the problem most in need of solution is how to reduce the time required to construct a complex control system. It is almost always possible to devise hardware and software to meet a process-control objective, but the task can be formidably expensive. Although certain new software engineering techniques and programming languages (Ada is one) will help somewhat, the real need is for better methods of specification and programming, easier means of analysis and testing and better organization of systems.



VOTING ARRANGEMENT is often provided in process-control systems to protect against failure. The scheme of hydraulic voting shown here is employed on the space shuttle to set the position of the rocket gimbal. The system has three sensors, four computers and four hydraulic actuators. Only one computer issues commands for a reading from each sensor, but all the computers receive data from all the sensors. The computers exchange enough information to ensure that

they agree on the data. If they do not agree, they discard the data. Otherwise they independently compute the appropriate output by means of identical algorithms. Each computer commands a separate actuator; if the commands are in conflict, three actuators can overpower the fourth actuator, so that there is effectively a principle of majority rule. If a component fails, a member of the crew can deal with the problem, as is indicated by the lines labeled "Commander."



We're light on your feet.

Our casting material is up to one-third lighter than plaster. It's three times stronger and we hope you never need it.

But on most any injury requiring a cast, you'd find Scotchcast® Casting Tape more comfortable than plaster and easier to get around in. It's thinner, so it fits under clothing. It also breathes. That reduces itching, odor and helps you stay cooler in warm weather.

All the comforts of Scotchcast tape were inspired by a 3M employee who

asked for something "less barbaric" than his plaster cast.

Hearing the needs of the medical profession and the patients it serves has helped 3M pioneer over 700 health care products. We now make everything from stethoscopes and pharmaceuticals to x-ray film that reduces the danger of radiation.

And it all began by listening.

3M hears you...

For your free brochure on 3M's health care products plus all other 3M products, write: Dept. 010709/3M, P.O. Box 22002, Robbinsdale, MN 55422.

Name _____

Address _____

City _____ State _____ Zip _____

Or call toll-free: **1-800-336-4327.**

3M

Computer Software in Science and Mathematics

Computation offers a new means of describing and investigating scientific and mathematical systems. Simulation by computer may be the only way to predict how certain complicated systems evolve

by Stephen Wolfram

Scientific laws give algorithms, or procedures, for determining how systems behave. The computer program is a medium in which the algorithms can be expressed and applied. Physical objects and mathematical structures can be represented as numbers and symbols in a computer, and a program can be written to manipulate them according to the algorithms. When the computer program is executed, it causes the numbers and symbols to be modified in the way specified by the scientific laws. It thereby allows the consequences of the laws to be deduced.

Executing a computer program is much like performing an experiment. Unlike the physical objects in a conventional experiment, however, the objects in a computer experiment are not bound by the laws of nature. Instead they follow the laws embodied in the computer program, which can be of any consistent form. Computation thus extends the realm of experimental science: it allows experiments to be performed in a hypothetical universe. Computation also extends theoretical science. Scientific laws have conventionally been constructed in terms of a particular set of mathematical functions and constructs, and they have often been developed as much for their mathematical simplicity as for their capacity to model the salient features of a phenomenon. A scientific law specified by an algorithm, however, can have any consistent form. The study of many complex systems, which have resisted analysis by traditional mathematical methods, is consequently being made possible through computer experiments and computer models. Computation is emerging as a major new approach to science, supplementing the long-standing methodologies of theory and experiment.

There are many scientific calculations, of course, that can be done by conventional mathematical means, without the aid of the computer. For example,

given the equations that describe the motion of electrons in an arbitrary magnetic field, it is possible to derive a simple mathematical formula that gives the trajectory of an electron in a uniform magnetic field (one whose strength is the same at all positions). For more complicated magnetic fields, however, there is no such simple mathematical formula. The equations of motion still yield an algorithm from which the trajectory of an electron can be determined. In principle the trajectory could be worked out by hand, but in practice only a computer can go through the large number of steps necessary to obtain accurate results.

A computer program that embodies the laws of motion for an electron in a magnetic field can be used to perform computer experiments. Such experiments are more flexible than conventional laboratory experiments. For example, a laboratory experiment could readily be devised to study the trajectory of an electron moving under the influence of the magnetic field in a television tube. No laboratory experiment, however, could reproduce the conditions encountered by an electron moving in the magnetic field surrounding a neutron star. The computer program can be applied in both cases.

The magnetic field under investiga-

tion is specified by a set of numbers stored in a computer. The computer program applies an algorithm that simulates the motion of the electron by changing the numbers representing its position at successive times. Computers are now fast enough for the simulations to be carried out quickly, and so it is practical to explore a large number of cases. The investigator can interact directly with the computer, modifying various aspects of a phenomenon as new results are obtained. The usual cycle of the scientific method, in which hypotheses are formulated and then tested, can be followed much faster with the aid of the computer.

Computer experiments are not limited to processes that occur in nature. For example, a computer program can describe the motion of magnetic monopoles in magnetic fields, even though magnetic monopoles have not been detected in physical experiments. Moreover, the program can be modified to embody various alternative laws for the motion of magnetic monopoles. Once again, when the program is executed, the consequences of the hypothetical laws can be determined. The computer thus enables the investigator to experiment with a range of hypothetical natural laws.

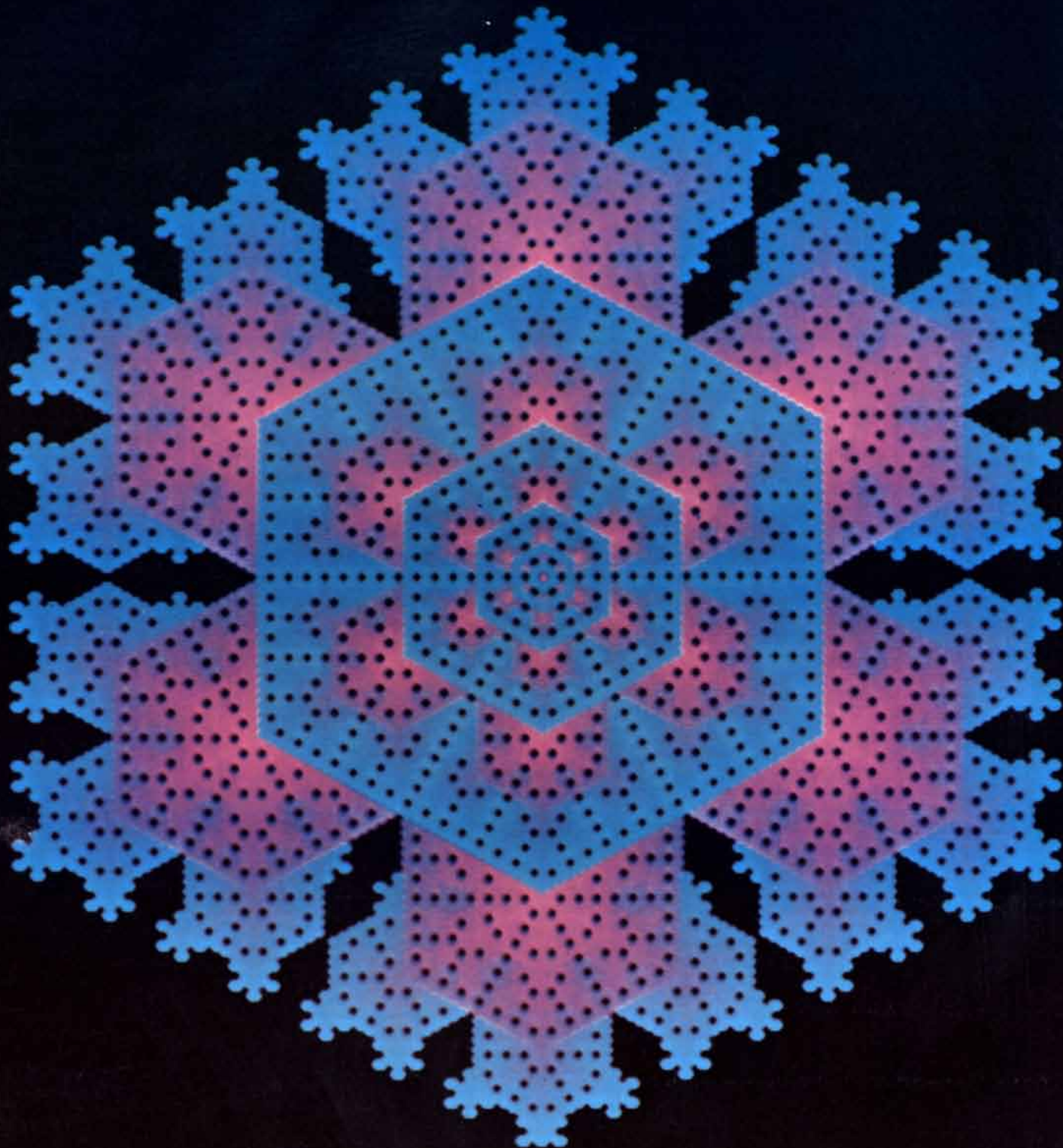
COMPUTER SIMULATION has made it practical to consider many new kinds of models for natural phenomena. Here the stages in the formation of a snowflake are generated by a computer program that embodies a model called a cellular automaton. According to the model, the plane is divided into a lattice of small, regular hexagonal cells. Each cell is assigned the value 0, which corresponds to water vapor (*black*), or the value 1, which corresponds to ice (*color*). Beginning with a single red cell in the center of the illustration, the simulated snowflake grows in a series of steps. At each step the subsequent value of any cell on the boundary of the snowflake depends on the total value of the six cells that surround it. If the total value is an odd number, the cell becomes ice and takes on the value 1; otherwise the cell remains vapor and keeps the value 0. The successive layers of ice formed in this way are shown as a sequence of colors, ranging from red to blue every time the number of layers doubles. The calculation required for each cell is simple, but for the pattern shown more than 10,000 calculations were needed. The only practical way to generate the pattern is by computer simulation. The illustration was made with the aid of a program written by Norman H. Packard of the Institute for Advanced Study.

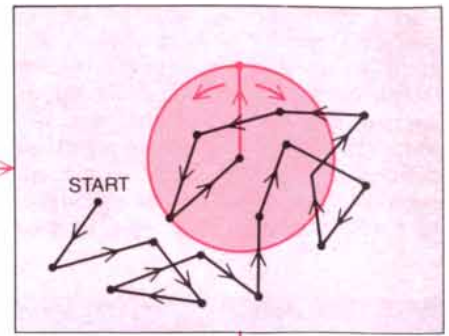
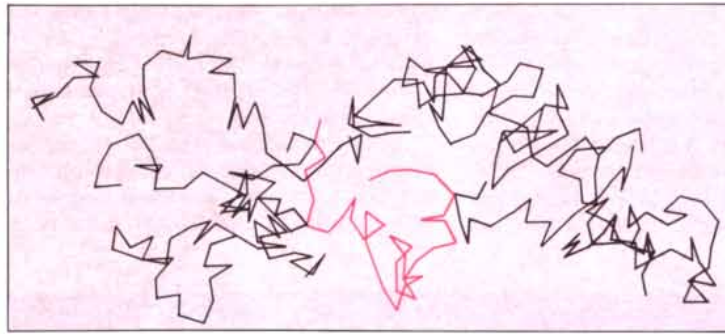
The computer can also be used to study the properties of abstract mathematical systems. Mathematical experiments carried out by computer can often suggest conjectures that are subsequently established by conventional mathematical proof. Consider a mathematical system that can be introduced to model the path of a beam of elec-

trons traveling through the magnetic fields in a circular particle accelerator. The transverse displacement of an electron as it passes a point on one of its revolutions around the accelerator ring is given by some fraction x between 0 and 1. The value of the fraction corresponding to the electron's displacement on the next revolution is then $ax(1 - x)$,

where a is a number that can range between 0 and 4. The formula gives an algorithm from which the sequence of values for the electron's displacement can be worked out.

A few trials show how the properties of the sequence depend on the value of a . If a is equal to 2 and the initial value of x is equal to .8, the next value of x ,

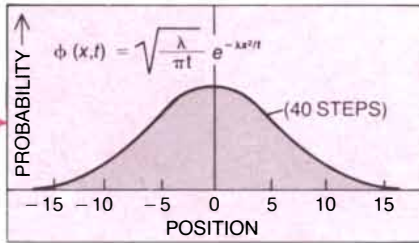




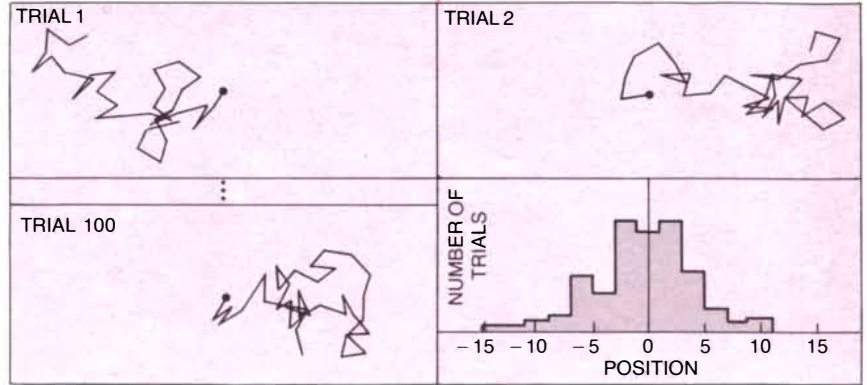
DIFFERENTIAL EQUATION

$$\frac{\partial \phi}{\partial t} = \lambda \left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} \right)$$

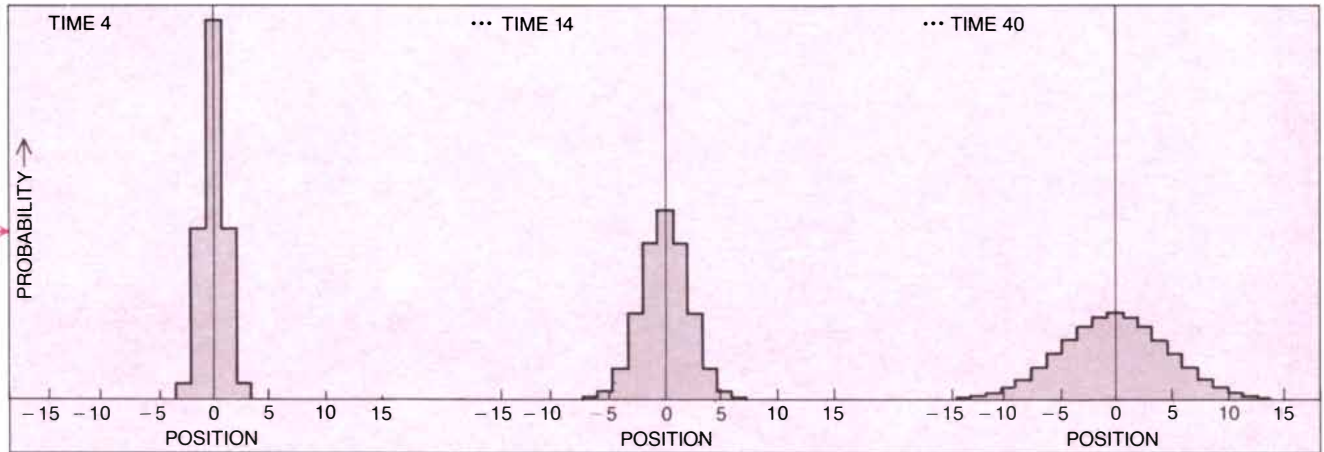
EXACT SOLUTION



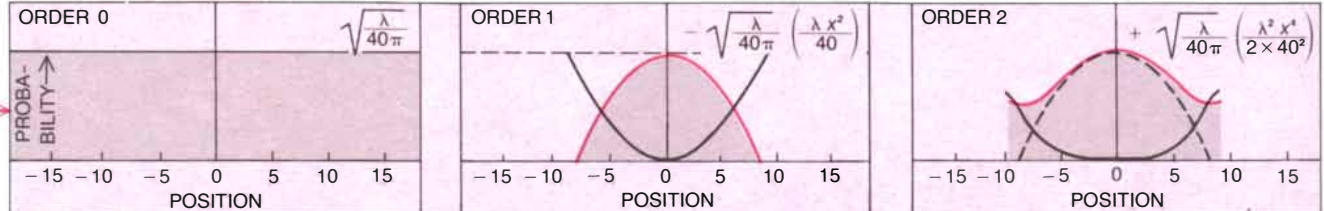
COMPUTER EXPERIMENT (40 STEPS)



NUMERICAL APPROXIMATION



ALGEBRAIC APPROXIMATION (40 STEPS)



MATHEMATICAL AND COMPUTATIONAL METHODS are applied in various ways in the study of random walks. A random walk is a model for such physical processes as the Brownian motion of a small particle suspended in a liquid. The particle undergoes random deflections as it is bombarded by the molecules in the liquid; its path can thus be described as a sequence of steps, each taken in a random direction. The most direct way to deduce the consequences of the model is by a computer experiment. Many random walks are simulated on a computer and their average properties are measured. The diagram shows a histogram in which the height of each bin records the number of simulated random walks that were found to have reached a particular range of positions after a certain time. As more trials are included, the shape of the histogram approaches that of the exact distribution of positions. For an ordinary random walk it is possible to derive the exact distribution directly. A differential equation can be

constructed for the distribution, and the equation is simple enough for an exact solution to be given. For most differential equations, however, no such exact solution can be obtained, and approximations must be made. In numerical approximations the smooth variation of quantities in the differential equation is approximated by a large number of small increments. The results shown in the diagram were obtained by a computer program in which the spatial and temporal increments were small fractions of the lengths and times for individual steps in the random walk. Algebraic approximations to the differential equation are found as a series of algebraic terms. The diagram shows the first three terms in such a series. The contribution of each term is shown as a solid black line or curve. The line or curve is added by superposition to the broken black line or curve that represents the previous order in the approximation. The result of the superposition is the current order in the approximation (solid colored curves).

which is given by $ax(1 - x)$, is equal to .32. If the formula is applied again, the value of x obtained is .4352. After several iterations the sequence of values for x converges to .5. Indeed, when a is small and x is any fraction between 0 and 1, the sequence quickly settles down to give the same value of x for each revolution of the electron.

As a increases, however, a phenomenon called period doubling can be observed. When a reaches 3, the sequence begins to alternate between two values of x . As a continues to increase, first four, then eight and finally, when it reaches about 3.57, an entire range of values for x appear. This behavior could not readily be guessed from the construction of the mathematical system, but it is immediately suggested by the computer experiment. The detailed properties of the system can then be established by a conventional proof.

The mathematical processes that can be described by a computer program are not limited to the operations and functions of conventional mathematics. For example, there is no conventional mathematical notation for the function that reverses the order of the digits in a number. Nevertheless, it is possible to define and apply the function in a computer program. The computer makes it practical to introduce scientific and mathematical laws that are intrinsically algorithmic in nature. Consider the chain of events set up when an electron accelerated to a high energy is fired into a block of lead. There is a certain probability that the electron emits a photon of a particular energy. If a photon is emitted, there is a certain probability that it gives rise to a second electron and a positron (the antiparticle of the electron). Each member of the pair can in turn emit more photons, so that a cascade of particles is eventually generated. There is no simple mathematical formula that can describe even the elements of the process. Nevertheless, an algorithm for the process can be incorporated into a computer program, and the outcome of the process can be deduced by executing the program. The algorithm serves as the basic law that describes the process.

The mathematical basis of most conventional models of natural phenomena is the differential equation. Such an equation gives relations between certain quantities and their rates of change. For example, a chemical reaction proceeds at a rate proportional to the concentrations of the reacting chemicals, and that relation can be expressed by a differential equation. A solution to the equation would give the concentration of each reactant as a function of time. In some simple cases it is possible to find a complete solution to the equation in terms of standard mathematical functions. In most cases, however, no

such exact solution can be obtained, and one must resort to approximation.

The commonest approximations are numerical. Suppose one term of a differential equation gives the instantaneous rate of change of a quantity with time. The term can be approximated by the total change in the quantity over some small interval and then substituted into the differential equation. The resulting equation is in effect an algorithm that determines the approximate value of the quantity at the end of an interval, given its value at the beginning of the interval. By applying the algorithm repeatedly for successive intervals, the approximate variation of the quantity with time can be found. Smaller intervals yield more accurate results. The calculation required for each interval is quite simple, but in most cases it must be repeated many times to achieve an acceptable level of accuracy. Such an approach is practical only with a computer.

The numerical methods embodied in computer programs have been employed to find approximate solutions to differential equations in a wide variety of disciplines. In some cases the solutions have a simple form. In many cases, however, the solutions show complicated, almost random behavior, even though the differential equations from which they arise are quite simple. For such cases experimental mathematics must be used.

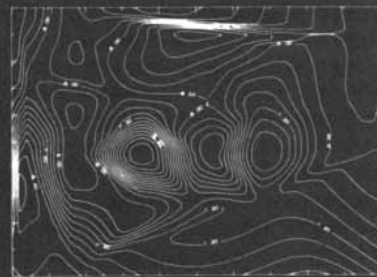
In practical applications one often finds not only that differential equations are complicated but also that there are many of them. For example, the theoretical models of nuclear explosions employed in the design of weapons and the study of supernovas involve hundreds of differential equations that describe the interactions of many isotopes. In practice such models are always used in the form of computer programs: only a computer can follow the interrelations among so many quantities.

The results of some numerical calculations, such as the abundance of helium in the universe, can be stated as single numbers. In most cases, however, one is concerned with the variation of certain quantities as the parameters of a calculation are changed. When the number of parameters is only one or two, the results can be displayed as a graph. When there are more than two parameters, however, the results often can be stated succinctly only as a mathematical formula. Exact formulas usually cannot be found, but it is often possible to derive approximate formulas. Such formulas are particularly convenient because, unlike graphs or tables of numbers, they can be inserted directly into other calculations.

A common form for an approximate formula is a series of terms. Each term includes a variable raised to some pow-

Contouring

Mainframe Power on the IBM Personal Computer with the In-Situ CONTUR Software



CONTUR is useful for a variety of applications, including geology, biology, chemistry, physics, mathematics, civil engineering, hydrology, and mining. The program supports Hewlett-Packard and Houston Instruments plotters.

For more information about CONTUR and other In-Situ software please call us.



In-Situ Inc.

P.O. Box 1
Laramie, WY 82070 USA

Telephone: 307 742-8213
TWX: 910-949-4944

AUDIO-FORUM® offers

the best in self-instructional foreign language courses using audio cassettes — featuring those used to train U.S. State Dept. personnel in Spanish, French, German, Portuguese, Japanese, Greek, Hebrew, Arabic, Chinese, Italian, and more.

Learn a foreign language on your own! Free Catalog

Call (203) 453-9794, or fill out and send this ad to —
Audio-Forum

Room 604, On-the-Green
Guilford CT 06437

Name _____

Address _____

City _____

State/Zip _____

I am particularly interested in (check choice):

- Spanish French German Polish
 Greek Russian Vietnamese
 Bulgarian Turkish Hausa
 Other

er; the power is larger in each successive term. When the value of the variable is small, the terms in the series become progressively smaller; thus for small values of x the sum of the first few terms in an infinite series such as $1 - x + x^2 - x^3 + \dots$ gives an accurate approximation to the sum of the entire series, which is $1/(1+x)$. The first few terms in a series are usually easy to evaluate, but the complexity of the terms increases rapidly thereafter. In order to evaluate terms that include large powers of x the computer becomes essential.

In principle computer programs can operate with any well-defined mathematical construct. In practice, however, the kinds of construct that can be used in a particular program are largely determined by the computer language in which the program is written. Numerical methods require only a limited set of mathematical constructs, and the programs that embody such methods can be written in general-purpose computer languages such as C, FORTRAN or BASIC. The derivation and manipulation of formulas require operations on higher-level mathematical constructs such as algebraic expressions, for which new computer languages are needed. Among the languages of this kind now in use is the SMP language that I have developed.

SMP is a language for manipulating symbols. It operates not only with numbers but also with symbolic expressions that can represent mathematical formu-

las. For example, in SMP the algebraic expression $2x - 3y + 5x - y$ would be simplified to the form $7x - 4y$. This transformation is a general one, valid for any possible numerical values of x and y . The standard operations of algebra and mathematical analysis are among the fundamental instructions in SMP [see illustration on page 196].

The SMP language also includes operations that allow higher-level mathematical constructs to be defined and manipulated, much as they are in ordinary mathematical work. Real numbers (which include all rational and irrational values) as well as complex numbers (which have both a real and an imaginary part) are fundamental in SMP. The mathematical constructs known as quaternions, which are generalizations of the complex numbers, are not fundamental. They can nonetheless be defined in SMP, and rules can be specified for their addition and multiplication. In this way the mathematical knowledge of SMP can be extended.

Some of the advantages of a language such as SMP can be compared to the advantages of using a calculator instead of a table of logarithms. By now the widespread availability of electronic calculators and computers has made such tables obsolete: it is far more convenient to call on an algorithm in a computer to obtain a logarithm than it is to look up the result in a table. Similarly, with a language such as SMP it has become pos-

sible to make the entire range of mathematical knowledge available in algorithmic form. For example, the calculation of integrals, conventionally done with the aid of a book of tables, can increasingly be left to a computer. The computer not only carries out the final calculations quickly and without error but also automates the process of finding the relevant formulas and methods.

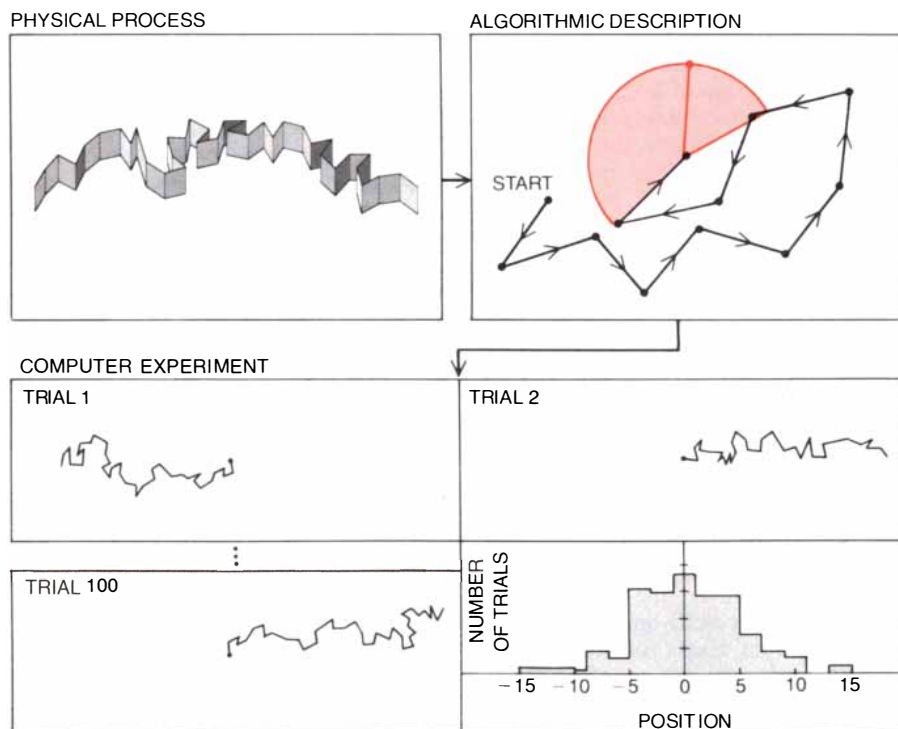
In SMP an expanding collection of definitions is being assembled in order to provide for a wide variety of mathematical calculations. One can now find in SMP the definition of variance in statistics, and one can immediately apply the definition to calculate the variance in a particular case. Such definitions enable programs written in the SMP language to call on increasingly sophisticated mathematical knowledge.

Differential equations give adequate models for the overall properties of physical processes such as chemical reactions. They describe, for example, the changes in the total concentration of molecules; they do not, however, account for the motions of individual molecules. These motions can be modeled as random walks: the path of each molecule is like the path that might be taken by a person in a milling crowd. In the simplest version of the model the molecule is assumed to travel in a straight line until it collides with another molecule; it then recoils in a random direction. All the straight-line steps are assumed to be of equal length. It turns out that if a large number of molecules are following random walks, the average change in the concentration of molecules with time can in fact be described by a differential equation called the diffusion equation.

There are many physical processes, however, for which no such average description seems possible. In such cases differential equations are not available and one must resort to direct simulation. The motions of many individual molecules or components must be followed explicitly; the overall behavior of the system is estimated by finding the average properties of the results. The only feasible way to carry out such simulations is by computer experiment: essentially no analysis of the systems for which analysis is necessary could be made without the computer.

The self-avoiding random walk is an example of a process that can apparently be studied only by direct simulation. It can be described by a simple algorithm that is similar to the ordinary random walk. It differs in that the successive steps in the self-avoiding random walk must not cross the path taken by any previous steps. The folding of long molecules such as DNA can be modeled as a self-avoiding random walk.

The introduction of the single con-



COMPUTATIONAL METHODS alone are used in the study of self-avoiding random walks. Self-avoiding random walks, which arise as models for physical processes such as the folding of polymer molecules, differ from ordinary random walks in that each step must avoid all previous steps. The complication makes it impossible to construct a simple differential equation that describes the average properties of the walk. Conventional mathematical approaches are thus ineffective. Properties of the self-avoiding random walk are found by direct simulation.



Who does a 12-year-old turn to when his dad's on drugs?

What happens to a youngster with problems, when the parents who should help have problems of their own?

Like drugs or alcoholism.

Or emotional crises that threaten a family's very existence.

It's hard to deal with painful situations like this at any age. But especially when you're underage.

And speaking practically, these family problems have a way of becoming business problems.

In economic terms, workers crippled by drug or alcohol addiction cost U.S. industry and society over 135 billion dollars a year.

That's why we at ITT created the Employee Assistance Program. To try and give constructive help to people who need it.

We maintain telephone hotlines around the world—24 hours a day, 7 days a week.

When employees or members of their families in participating ITT companies

call us, our Program people hear them out. Then we refer them to somebody nearby who can help.

It's all done in strict confidence, without anybody knowing but the employees themselves.

A number of our people have turned to this ITT program since it began. And most are still with us, productive and happy members of our corporate family.

But more important, happy members of their own families.

The best ideas are the ideas that help people.

ITT

For details, write: Director—ITT Employee Assistance Program, Personnel Dept., 320 Park Ave., NY, NY 10022. Or call 212-940-2550.
© 1984 ITT Corporation, 320 Park Avenue, New York, NY 10022

straint makes the self-avoiding random walk much more complicated than the ordinary random walk. Indeed, there is no simple average description, analogous to the diffusion equation, that is known for the self-avoiding random walk. In order to investigate its prop-

erties it seems one has no choice but to carry out a direct computer experiment. The procedure is to generate a large number of sample random walks, choosing a random direction at each step. The properties of all the walks are then averaged. Such a procedure is an

example of the Monte Carlo method, so called because its application depends on the element of chance.

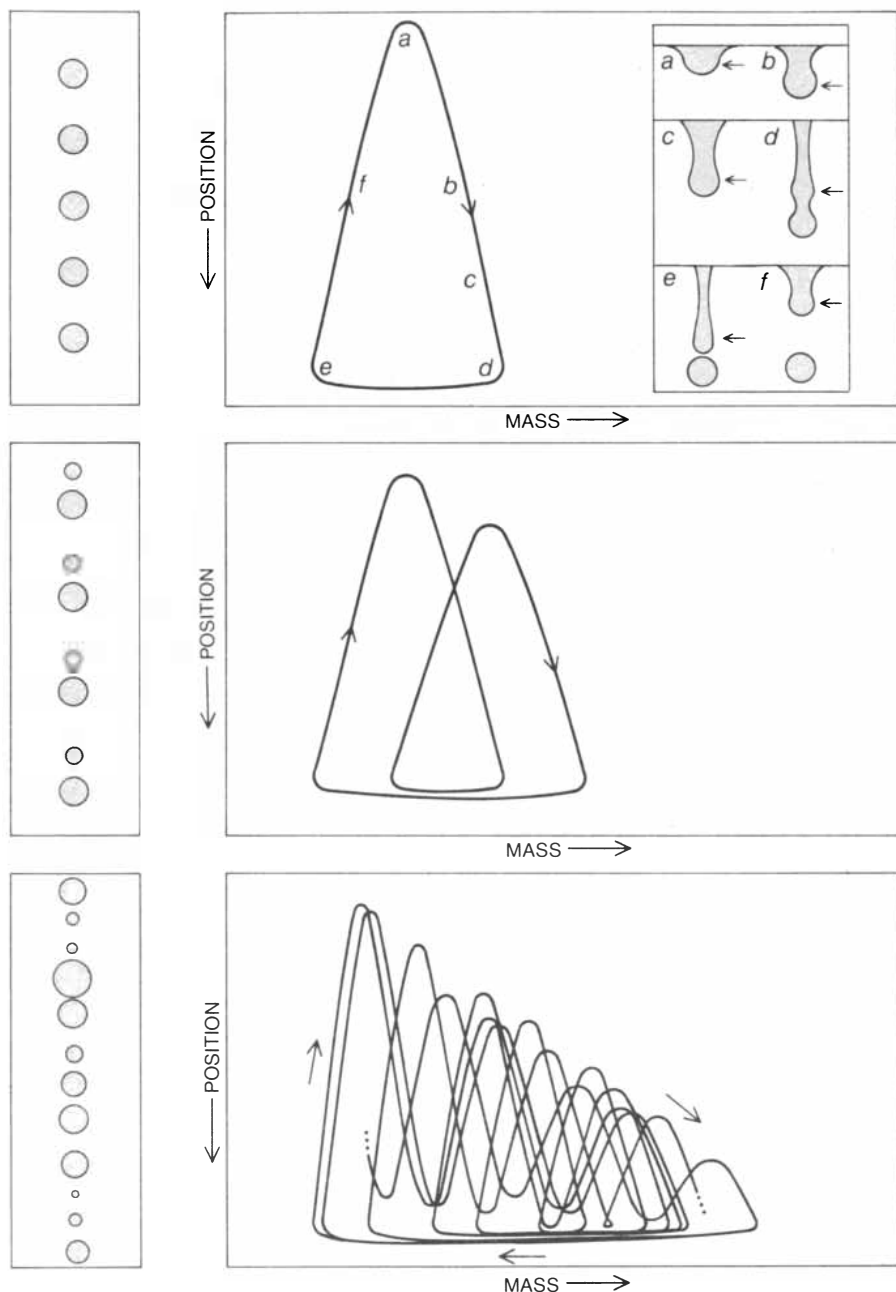
Several examples have been given of systems whose construction is quite simple but whose behavior is extremely complicated. The study of such systems is leading to a new field called complex-systems theory, in which the computational method plays a central role. The archetypal example is fluid turbulence, which develops, for example, when water flows rapidly around an obstruction. The set of differential equations satisfied by the fluid can easily be stated. Nevertheless, the patterns of fluid flow to which the equations give rise have largely defied mathematical analysis or description. In practice the patterns are found either through observation of the actual physical system or, as far as possible, through computer experiment.

It is suspected there is a set of mathematical mechanisms common to many systems that give rise to complicated behavior. The mechanisms can best be studied in systems whose construction is as simple as possible. Such studies have recently been done for a class of mathematical systems known as cellular automata. A cellular automaton is made up of many identical components; each component evolves according to a simple set of rules. Taken together, however, the components generate behavior of essentially arbitrary complexity.

The components of a cellular automaton are mathematical "cells," arranged in one dimension at a sequence of equally spaced points along a line or in two dimensions on a regular grid of squares or hexagons. Each cell carries a value chosen from a small set of possibilities, often just 0 and 1. The values of all the cells in the cellular automaton are simultaneously updated at each "tick" of a clock according to a definite rule. The rule specifies the new value of a cell, given its previous value and the previous values of its nearest neighbors or some other nearby set of cells.

Consider a one-dimensional cellular automaton in which each cell can have the value 0 or 1. Even in such a simple case the overall behavior of the cellular automaton can be quite complex; the most effective way to investigate the behavior is by computer experiment. Most of the properties of cellular automata have in fact been conjectured on the basis of patterns generated in computer experiments. In some cases they have later been established by conventional mathematical arguments.

Cellular automata can serve as explicit models for a wide variety of physical processes. Suppose ice is represented on a two-dimensional hexagonal grid by cells with the value 1 and water vapor is represented by cells with the value 0. A cellular-automaton rule can then be



CHAOTIC BEHAVIOR is seen in many natural systems. A familiar example is the dripping faucet, described by a mathematical model formulated in terms of a differential equation by Robert Shaw of the Institute for Advanced Study. When the rate at which water flows through the faucet is very low, drops of equal size are formed at regular intervals (left). The model implies that if the position of the top of each drop that forms (arrows) is plotted against the mass of the system is represented by a point that traces the curve with time. If the flow is increased, the behavior of the system suddenly becomes more complicated. A phenomenon known as period doubling occurs, and pairs of drops, often of different sizes, are formed in each cycle. If the flow is further increased, there is a sequence of additional period doublings. Finally, just before the water flowing from the faucet becomes continuous an irregular stream of drops is produced. The drops have an entire range of sizes, and the intervals between the formation of consecutive drops appear to be random. The behavior of the system is then described by an irregular curve called a strange or chaotic attractor. The form of the curve is implied by the differential equation, but in practice it can be found only by numerical-approximation techniques.

PHYSICIAN...

LET CME = DISCOTEST™

THE PACE-SETTING COMPUTER CASE SIMULATION SERIES FROM
SCIENTIFIC AMERICAN *MEDICINE*

10 INSTANT SCORING AND CRITIQUE OF YOUR TEST RESPONSES

Load the 5 1/4" disk into your computer and the simulation of a clinical encounter begins to unfold in real time. At the touch of a key, you can interview the "patient" and obtain a detailed history, perform diagnostic procedures, carry out the therapeutic regimen you've selected and instantly receive details on the "patient's" progress. At the moment you complete the test, while the details of the case are still fresh in your mind, the computer generates your score and a full critique of your answers.

20 RE-USE THE PROBLEM TO TRY OUT ALTERNATE THERAPEUTIC PATHWAYS

Unlike a paper CME program or a review course, DISCOTEST™ is truly a continuing education tool that you or a colleague can use again and again, whether you immediately want to strengthen your understanding and improve your score, or months later, review patient management techniques.

30 ECONOMICAL AND FLEXIBLE WAY TO EARN VALUABLE CME CREDITS

Accredited by the Stanford University School of Medicine, the program provides eight tests per year (two per disk) worth up to 32 Category 1 or prescribed credits.* You complete the tests when you choose, without sacrificing valuable patient-care time. And, you can save the time and expense of review courses.

40 AN ENJOYABLE, USER-FRIENDLY PROGRAM

DISCOTEST's interactive, user-friendly software makes it easy and enjoyable to earn CME credits on your personal computer. Available on 5 1/4" floppy disks, DISCOTEST™ can be run on an IBM PC® or Apple® IIe or II+ (or compatible models); it requires a minimum of 64K RAM, a single disk drive, and an 80-column screen.

50 IF 10, 20, 30, 40 = YOUR NEEDS, GO TO 60

60 HERE'S HOW IT WORKS

Your first year's subscription includes four disks, containing two patient management problems each; a handsome binder to store your disks; and a printed user's manual, giving an introduction to DISCOTEST™, step-by-step operating instructions, and answers to common questions. Disks are sent on a quarterly schedule. In addition, we will send you a bonus review disk, with a sample test similar to those you will be taking for credit.

DISCOTEST™ costs a modest \$148, or less than \$5 per credit. The software is backed by a money-back guarantee, with free replacement of defective disks. Take this opportunity to subscribe to DISCOTEST™ and combine the advantages of high-caliber CME instruction with the immediate interaction of the computer.

70 END

SCIENTIFIC AMERICAN **MEDICINE**

415 MADISON AVENUE, NEW YORK, N.Y. 10017

7Q

Yes, please enroll me in DISCOTEST™ at a price of \$148 per year. My one-year subscription includes:

- A bonus review disk
- Four disks containing two CME tests each
- A comprehensive user's manual
- A handsome binder for the disks and manual
- 8 certification cards (one per test) to record and submit my computer-generated code for CME credit
- A total of up to 32 Category 1 or prescribed credits

Check enclosed* Bill me MasterCard VISA

Account No. _____ Exp. Date _____

*Please add applicable sales tax if resident of California, Illinois, Michigan, Massachusetts, or New York. All payments must be in U.S. dollars.

DISCOTEST™ will be compatible with several personal computer operating systems. Please indicate which one of the following personal computers you would use or with which your system is compatible.

- IBM PC with DOS 2.0, or compatible
- Apple IIe with Apple DOS 3.3, or compatible
- Apple II+ with Apple DOS 3.3, or compatible

Name _____

Address _____

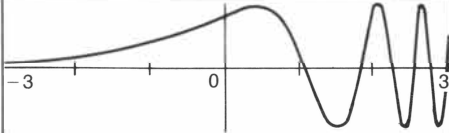
City _____ State _____ Zip _____

Medical Specialty _____

Signature _____

*As an organization accredited for continuing medical education, the Stanford University School of Medicine designates this continuing medical education activity as meeting the criteria for 32 credit hours in Category 1 for Educational Materials for the Physician's Recognition Award of the American Medical Association, provided it has been completed according to instructions. This program has been reviewed and is acceptable for 32 prescribed hours by the American Academy of Family Physicians.

Apple II+ and Apple IIe are registered trademarks of Apple Computer, Inc.

INPUT	OUTPUT	COMMENT
6+17	23	Evaluate a numerical expression.
6/7+8/9	110/63	Evaluate a numerical expression with exact fractions.
2x-3x+1	1-x	Simplify an algebraic expression.
Ex[(x-1)(x+1)]	-1+x ²	Expand algebraic expressions made up of products of terms. The notation x ^y stands for x raised to the power of y. A space between two non-numerical expressions stands for multiplication.
Ex[(x-a) ² (x+2a) ⁵]	8a ⁶ x ⁶ +21a ⁵ x ⁵ +10a ⁴ x ⁴ -40a ⁴ x ³ -48a ⁵ x ² +16a ⁶ x+32a ⁷ +x ⁷	
Fac[x ² -1]	(-1+x)(1+x)	Factor algebraic expressions.
Fac[x ⁶ -6x ⁴ +4x ³ +9x ² -12x+4]	(-1+x) ⁴ (2+x) ²	
Sol[x ² -3x+1=0,x]	$\{x \rightarrow \frac{3-5^{1/2}}{2}, x \rightarrow \frac{3+5^{1/2}}{2}\}$	Solve an equation for the variable x.
Sol[{x+3a y=4,y-15x=6b},{x,y}]	$\{x \rightarrow \frac{4}{1+45a} - \frac{18ab}{1+45a}, y \rightarrow \frac{60}{1+45a} + \frac{6b}{1+45a}\}$	Solve a pair of simultaneous equations for the variables x and y.
Ps[(1+x ³) E ^{x,x,0,6}]	$1+x+\frac{x^2}{2}+\frac{7x^3}{6}+\frac{25x^4}{24}+\frac{61x^5}{120}+\frac{121x^6}{720}$	Find a power-series approximation for the expression e ^x (1+x) ³ for x close to 0, keeping terms up to order x ⁶ .
t:x-2a t ² -2t+1	1+4a-2x+(-2a+x) ²	Assign the value x-2a to the symbol t; simplify the expression t ² -2t+1 for this value.
f[2]:6x+1 f[3]:4-x a f[2]+b f[3]+c f[1]	a(1+6x)+b(4-x)+c f[1]	Assign the value 6x+1 to f[2] and the value 4-x to f[3]; evaluate an expression involving f[1], f[2] and f[3], where f[1] is not yet specified.
f	{[2]:1+6x,[3]:4-x}	Print the object f, which is a list whose elements are indexed by the numbers shown in the brackets.
f[1]:7 f	{7,1+6x,4-x}	Assign the value 7 to f[1]; print the object f, which is now given as a vector, or ordered list of elements.
f ² -8	{41,-8+(1+6x) ² , -8+(4-x) ² }	Find the square and then subtract 8 from each element of the vector f; the result is a new vector.
f[p]:5x f[p ²]:6x f	{[p ²]:6x,[p]:5x,[1]:7,[2]:1+6x,[3]:4-x}	Assign values for elements of f that have non-numerical indexes; print the object f.
f[\$x]:\$x ² f	{[p ²]:6x,[p]:5x,[1]:7,[2]:1+6x,[3]:4-x,[\$x]:\$x ² }	Assign a value for f[\$x], where \$x is any expression; the general definition is placed at the end of the list f and is used only when none of the preceding special cases apply. Print the object f.
f[p]+f[2]+f[a]	1+11x+a ²	Evaluate the expression f[p]+f[2]+f[a]; the general definition for f[\$x] is applied in order to evaluate f[a].
g[\$x_ = Natp[\$x]]:\$x g[\$x-1] g[1]:1 g	{[1]:1,[\$x_ = Natp[\$x]]:\$x g[\$x-1]}	Define the factorial function g[x] for natural numbers x, where g[N] is equal to 1 x 2 x ... x N. The definition is given by a recursive formula in which g[x] is specified in terms of g[\$x-1]. The expression \$x_ = Natp[\$x] indicates that \$x must be a natural, or positive whole, number.
g[5]	120	Evaluate g[5], the factorial of 5.
Abs[3] Abs[-3] Abs[-x]	3 3 Abs[x]	Find the absolute values of -3, 3 and -x.
Abs[\$x \$x]: Abs[\$x] Abs[\$x]		Define the absolute value of the product of two arbitrary expressions \$x and \$\$x to be the product of their absolute values.
Abs[\$x^(n_ = Natp[\$n])]: Abs[\$x]^\$n		Define the absolute value of the arbitrary expression \$x raised to the natural-number power \$n to be the absolute value of \$x to the power \$n.
Abs[a b^2 c]	Abs[a] Abs[b] ² Abs[c]	Find the absolute value of the product a x b ² x c according to the standard rules of algebra and the definitions given for the absolute-value function.
Graph[Sin[E ^x],x,-3,3]		Plot a graph of the function sin(e ^x) for values of x from -3 to 3.

MATHEMATICAL CALCULATIONS are carried out by a computer in this example of a dialogue in the SMP computer language developed by the author. The computer manipulates algebraic formulas and other symbolic constructs as well as numbers. The commands in

the language include all the operations of standard mathematics. The last few panels show how new operations can be defined. Properties of the absolute-value function are defined and then are applied by the computer to simplify any expression that includes the function.

used to simulate the successive stages in the freezing of a snowflake. The rule states that once a cell is frozen it does not thaw. Cells exposed at the edge of the growing pattern freeze unless they have so many ice neighbors that they cannot dissipate enough heat to freeze. Snowflakes grown in a computer experiment from a single frozen cell according to this rule show intricate treelike patterns, which bear a close resemblance to real snowflakes. A set of differential equations can also describe the growth of snowflakes, but the much simpler model given by the cellular automaton seems to preserve the essence of the process by which complex patterns are created. Similar models appear to work for biological systems: intricate patterns of growth and pigmentation may be accounted for by the simple algorithms that generate cellular automata.

Simulation by computer is the only method now used for investigating many of the systems discussed so far. It is natural to ask whether simulation, as a matter of principle, is the most efficient possible procedure or whether there is a mathematical formula that could lead more directly to the results. In order to address the question the correspondence between physical and computational processes must be studied more closely.

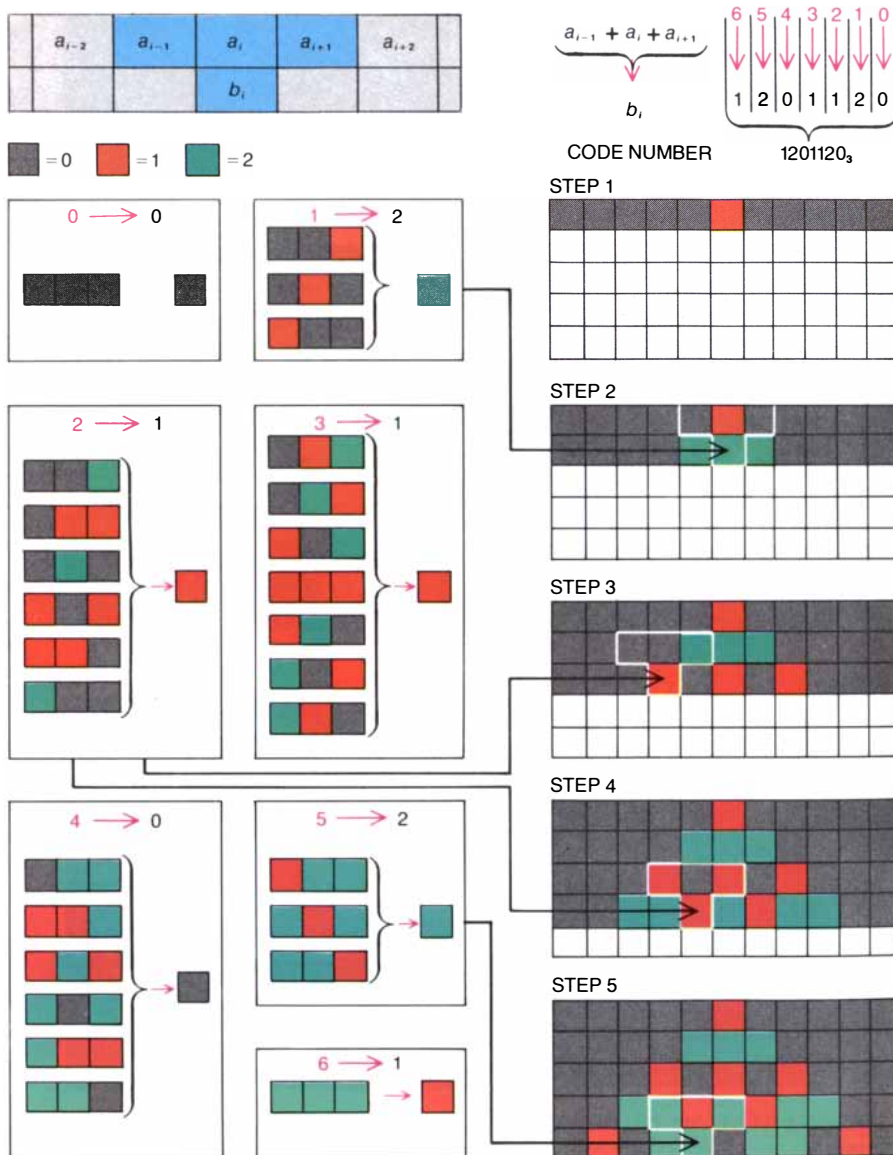
It is presumably true that any physical process can be described by an algorithm, and so any physical process can be represented as a computational process. One must determine how complicated the latter process is. In cellular automata the correspondence between physical and computational processes is particularly clear. A cellular automaton can be regarded as a model of a physical system, but it can also be regarded as a computational system closely analogous to an ordinary digital computer. The sequence of initial cell values in a cellular automaton can be understood as abstract data or information, much like the sequence of binary digits in the memory of a digital computer. During the evolution of a cellular automaton the information is processed: the values of the cells are modified according to definite rules. Similarly, the digits stored in the memory of the digital computer are modified by rules built into the central processing unit of the computer.

The evolution of a cellular automaton from some initial configuration may thus be viewed as a computation that processes the information carried by the configuration. For cellular automata exhibiting simple behavior the computation is a simple one. For example, it may serve only to pick out sequences of three consecutive cells whose initial values are equal to 1. On the other hand, the evolution of cellular automata that show complicated behavior may correspond to a complicated computation.

It is always possible to determine the outcome of a given number of steps in the evolution of a cellular automaton by explicitly simulating each step. The problem is whether or not there can be a more efficient procedure. Can there be a short cut to step-by-step simulation, an algorithm that finds the outcome after many steps in the evolution of a cellular automaton without effectively tracing through each step? Such an algorithm could be executed by a computer, and it would predict the evolution of a cellular automaton without explicitly simulating it. The basis of its operation would be that the computer could carry out a more sophisticated computation than the cellular automaton could and so

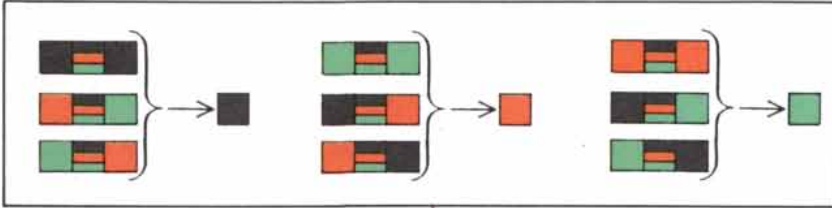
achieve the same result in fewer steps. It would be as if the cellular automaton were to calculate 7 times 18 by explicitly finding the sum of seven 18's, while the computer found the same product according to the standard method for multiplication. Such a short cut is available only if the computer is able to carry out a calculation that is intrinsically more sophisticated than the calculation embodied in the evolution of the cellular automaton.

One can define a certain class of problems called computable problems that can be solved in a finite time by following definite algorithms. A simple computer such as an adding machine can solve only a small subset of these.

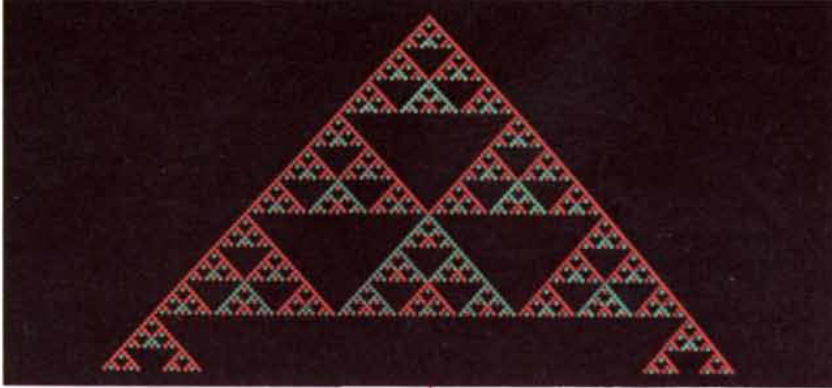


CELLULAR AUTOMATA are simple models that appear to capture the essential features of a wide variety of natural systems. A one-dimensional cellular automaton is made up of a line of cells, shown in the diagram as colored squares. Each cell can take on a number of possible values, represented by different colors. The cellular automaton evolves in a series of steps, shown as a sequence of rows of squares progressing down the page. At each step the values of all the cells are updated according to a fixed rule. In the case illustrated the rule specifies the new value of a cell in terms of the sum of its previous value and the previous values of its immediate neighbors. Such rules are conveniently specified by code numbers defined as shown in the diagram; the subscript 3 is given because each cell can take on one of three possible values.

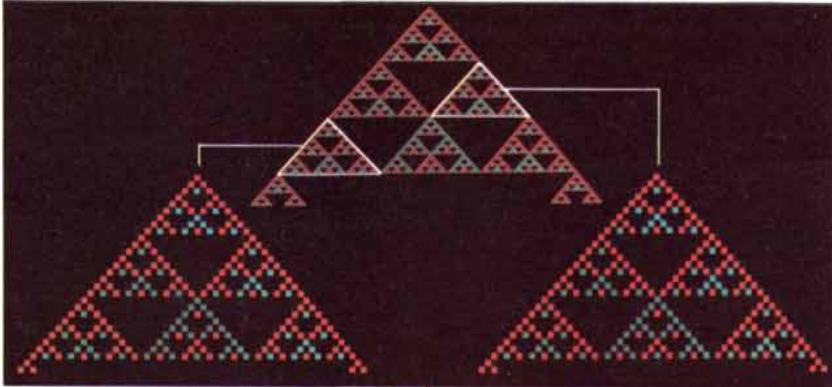
RULE



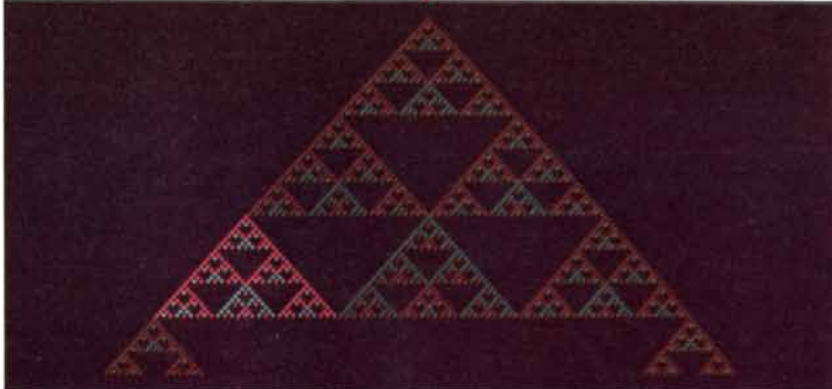
SIMULATION



CONJECTURE



PROOF



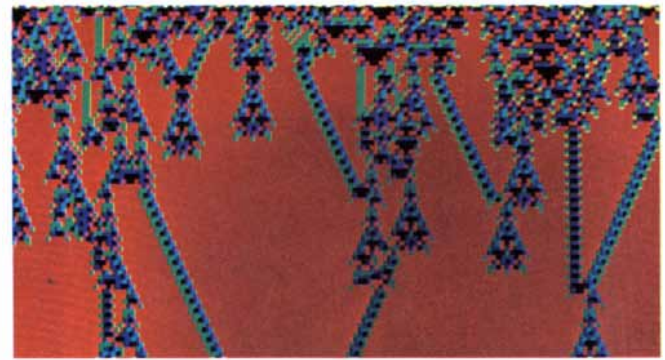
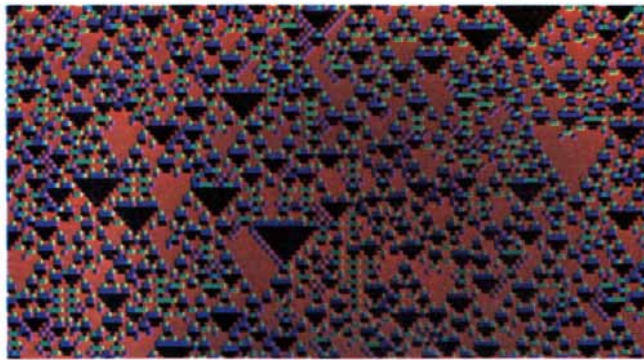
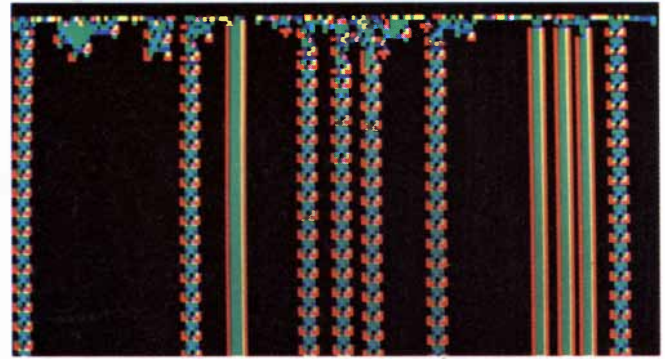
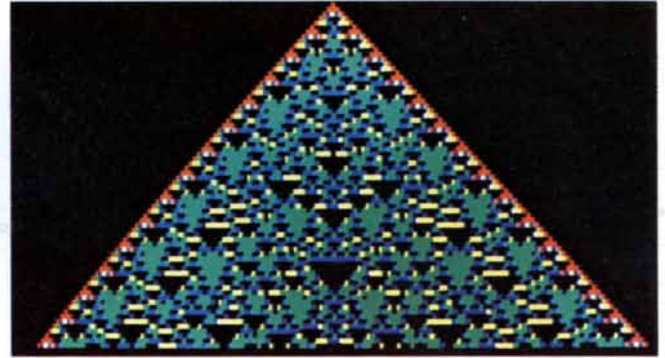
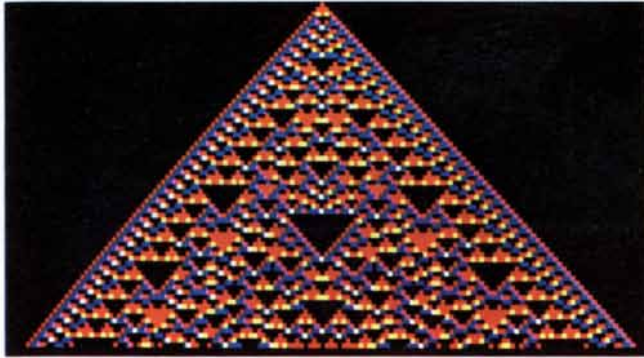
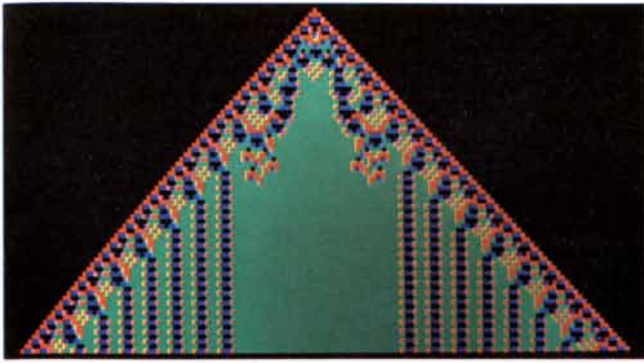
EXPERIMENTAL MATHEMATICS is an exploratory technique made possible largely through the use of computers. Any set of mathematical rules can be applied repeatedly by a computer and their consequences explored in an experimental fashion. For example, in order to study a pattern generated by the cellular automaton defined by the rule shown, one begins by explicitly simulating on a computer many steps in the evolution of the cellular automaton. Inspection of the pattern obtained then leads to the conjecture that it is fractal, or self-similar, in the sense that parts of it, when enlarged, have the same overall form as the whole. The conjecture, once made, is comparatively easy to prove by conventional mathematical techniques. The proof can be based on the fact that the initial conditions for growth from certain cells in the pattern are the same as the conditions for growth from the very first cell. There are an increasing number of mathematical results that were discovered in computer experiments. Some of them have subsequently been reproduced by conventional mathematical arguments.

problems. There exist universal, or general-purpose, computers, however, that can solve any computable problem. A real digital computer is essentially such a universal machine. The instructions that can be executed by the central processing unit of the computer are rich enough to serve as the elements of a computer program that can embody any algorithm. A number of systems in addition to the digital computer have been shown to be capable of universal computation. Several cellular automata are among them: for example, universal computation has been proved for a simple two-dimensional cellular automaton with a 0 or a 1 in each cell. It is strongly suspected that several one-dimensional cellular automata are also universal computers. The simplest candidates have three possible values at each cell and rules of evolution that take account only of the nearest-neighbor cells.

Cellular automata that are capable of universal computation can mimic the behavior of any possible computer; since any physical process can be represented as a computational process, they can mimic the action of any possible physical system as well. If there were an algorithm that could work out the behavior of these cellular automata faster than the automata themselves evolve, the algorithm would allow any computation to be speeded up. Because this conclusion would lead to a logical contradiction, it follows there can be no general short cut that predicts the evolution of an arbitrary cellular automaton. The calculation corresponding to the evolution is irreducible: its outcome can be found effectively only by simulating the evolution explicitly. Thus direct simulation is indeed the most efficient method for determining the behavior of some cellular automata. There is no way to predict their evolution; one must simply watch it happen.

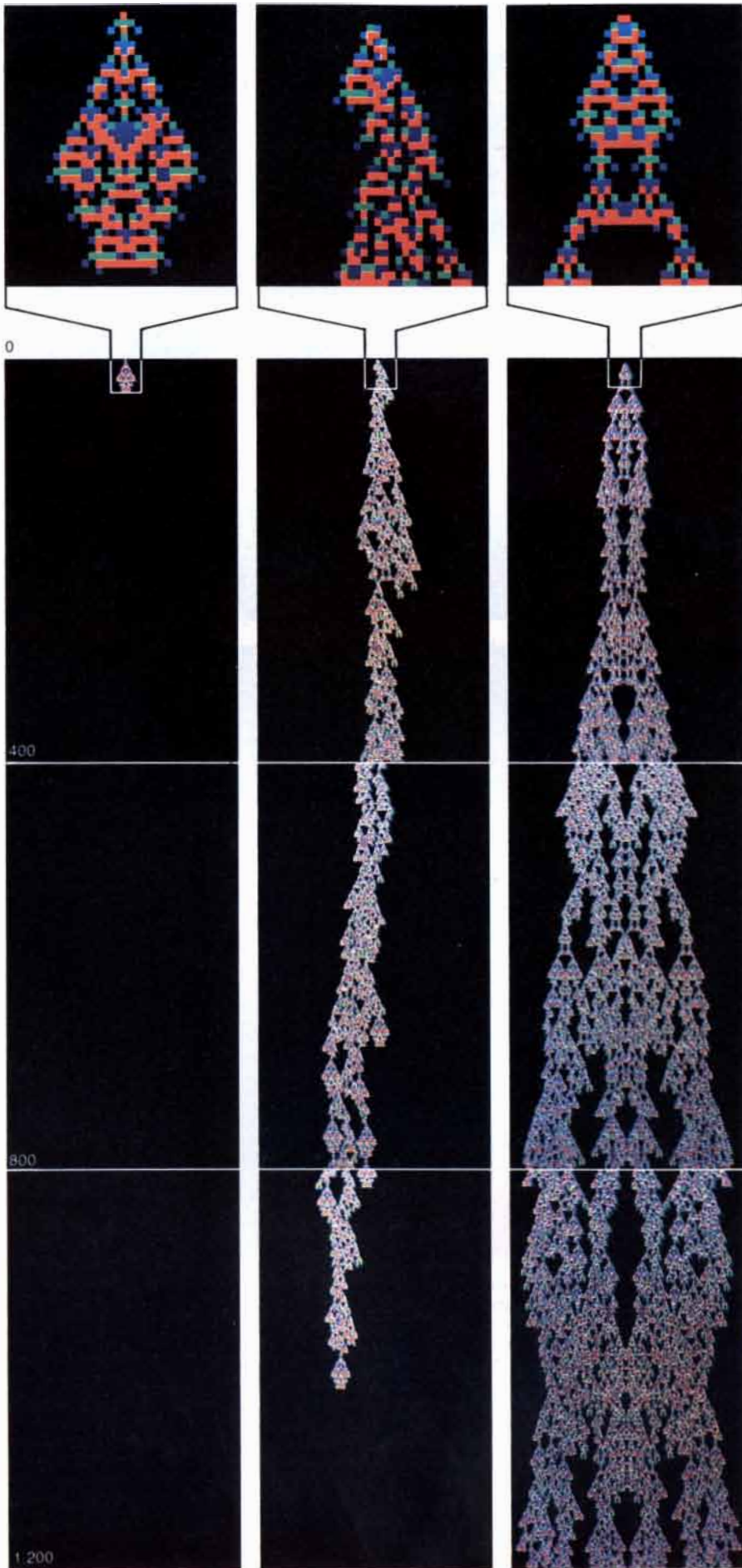
It is not yet known how widespread the phenomenon of computational irreducibility is among cellular automata or among physical systems in general. Nevertheless, it is clear that the elements of a system need not be very complicated for the overall evolution of the system to be computationally irreducible. It may be that computational irreducibility is almost always present when the behavior of a system appears complicated or chaotic. General mathematical formulas that describe the overall behavior of such systems are not known, and it is possible no such formulas can ever be found. In that case, explicit simulation in a computer experiment is the only available method of investigation.

Much of physical science has traditionally focused on the study of computationally reducible phenomena, for which simple overall descriptions can be given. In real physical systems, however,



2213310 _s	4200410 _s	= 0
331240 _s	2024310 _s	= 1
1100400 _s	2231000 _s	= 2
131210 _s	3211310 _s	= 3
		= 4

COMPLEX BEHAVIOR can develop even in systems with simple components. The eight cellular automata shown in the photographs are made up of lines of cells that take on one of five possible values. The value of each cell is determined by a simple rule based on the values of its neighbors on the previous line. Each pattern is generated by the rule whose code number is given in the key (see illustration on page 197). The patterns in the upper four photographs are grown from a single colored cell. Even in this case the patterns generated can be complex, and they sometimes appear quite random. The complex patterns formed in such physical processes as the flow of a turbulent fluid may well arise from the same mechanism. Complex patterns generated by cellular automata can also serve as a source of effectively random numbers, and they can be applied to encrypt messages by converting a text into an apparently random form. The patterns in the lower four photographs begin with disordered states. Even though the values of the cells in these initial states are chosen at random, the evolution of the cellular automata gives rise to structures of four basic classes. In the two classes shown in the third row of photographs the long-term behavior of the cellular automata is comparatively simple; in the two classes shown in the bottom row it can be highly complex. The behavior of many natural systems may well conform to this classification.



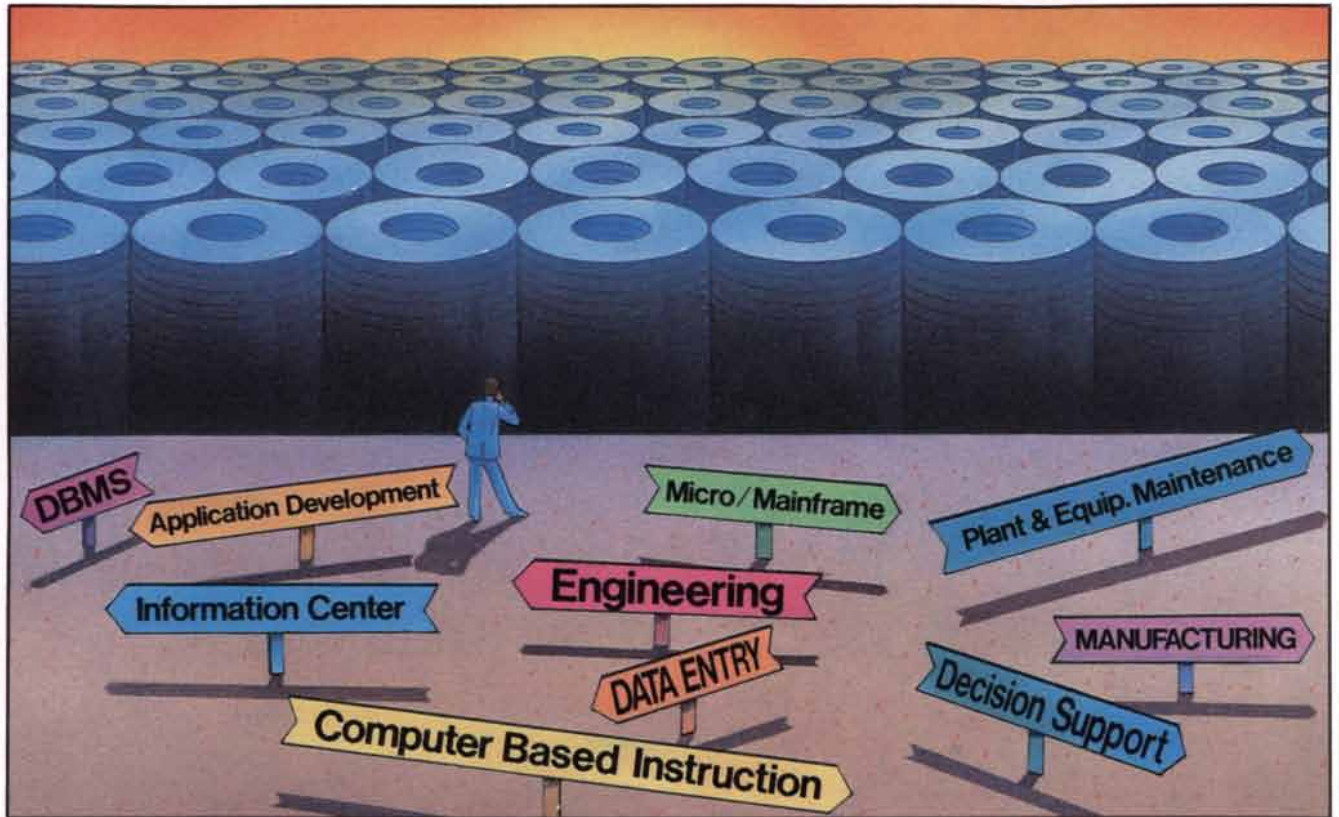
computational reducibility may well be the exception rather than the rule. Fluid turbulence is probably one of many examples of computational irreducibility. In biological systems computational irreducibility may be even more widespread: it may turn out that the form of a biological organism can be determined from its genetic code essentially only by following each step in its development. When computational irreducibility is present, one must adopt a methodology that depends heavily on computation.

One of the consequences of computational irreducibility is that there are questions that can be asked about the ultimate behavior of a system but that cannot be answered in full generality by any finite mathematical or computational process. Such questions must therefore be considered undecidable. An example of such a question is whether a particular pattern ever dies out in the evolution of a cellular automaton. It is straightforward to answer the question for some definite number of steps, say 1,000: one need only simulate 1,000 steps in the evolution of the cellular automaton. In order to determine the answer for any number of steps, however, one must simulate the evolution of the cellular automaton for a potentially infinite number of steps. If the cellular automaton is computationally irreducible, there is no effective alternative to such direct simulation.

The upshot is that no calculation of any fixed length can be guaranteed to determine whether a pattern will ultimately die out. It may be possible to tell the fate of a particular pattern after tracing only a few steps in its evolution, but there is no general way to tell in advance how many steps will be re-

UNDECIDABLE PROBLEMS can arise in the mathematical analysis of models of physical systems. For example, consider the problem of determining whether a pattern generated by the evolution of a cellular automaton will ever die out, so that all the cells become black. The patterns generated by the cellular automaton shown above are so complicated that the only possible general approach to the solution of the problem is to explicitly simulate the evolution of the cellular automaton. The pattern obtained from the initial state shown at the left is found to die out after just 16 steps. The initial state in the center yields a pattern that takes 1,016 steps to die out. The initial state at the right gives rise to a pattern whose fate remains unclear even after a simulation carried out over many thousands of steps. In general no finite simulation of a fixed number of steps can be guaranteed to determine the ultimate behavior of the cellular automaton. Hence the problem of whether or not a particular pattern ultimately dies out, or halts, is said to be formally undecidable. The cellular automaton shown here follows a rule specified by the code number 3311100320.

Try to find software that solves your problem.

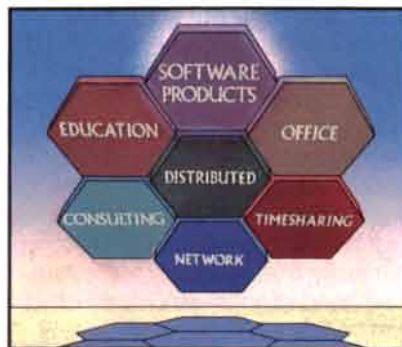


Or call BOEING.

Acquiring mainframe and micro software that best fits your needs isn't easy. Today's software landscape seems unending. So to obtain software that actually achieves your specific objectives, you need programs with proven problem-solving capabilities. Like software from Boeing.

Every software package from Boeing Computer Services is backed by Boeing expertise and experience. That's why both users and data processing professionals appreciate our solutions to a myriad of computing needs. Executives in many industries depend on our financial modeling and decision support software for accurate, up-to-the-minute pictures of business activity and for reliable forecasts. Production managers turn to Boeing for on-line manufacturing software that can keep track of all elements in the production cycle . . .

even in exacting make-to-order plants. Engineers increase their productivity with dynamic analysis and simulation using Boeing software. Boeing computer-based instruction software and courseware is central to the education and training programs of many companies, large and small. It is used cross-company and cross-discipline.



One of the newest relational data base management systems on the scene comes from Boeing. Its cost is low; its function is extensive. It runs on IBM, CDC, DEC VAX, Data General and Prime computers, and interfaces with a micro version.

For more information about Boeing software solutions, call (206) 763-5000. Or write BOEING COMPUTER SERVICES, M/S 7K-11, P.O. Box 24346, Seattle, WA 98124. Ask about our "TRY IT" evaluations.

For information about Boeing's other integrated information services—including enhanced remote computing, distributed processing, network services, office automation, consulting, and education and training—call toll free 1-800-447-4700. Or write BOEING COMPUTERSERVICES, M/SCV-26-18C, 7980 Gallows Court, Vienna, VA 22180.

BOEING COMPUTER SERVICES A Division of The Boeing Company



Casio brings space-age technology to spaceship earth.

Casio's solar-powered scientific calculators put space-age technology easily within your reach.

Our FX-910 is the logical choice for students and engineers alike. At only \$24.95, it gives you algebraic logic, 48 functions and an 8-digit + 2-digit exponent display—in a size that will fit as easily in your pocket as its price will suit your pocketbook.

At the same time, our credit card-size FX-90 (\$29.95) has members of the scientific community flipping—over its 49-function flip-open keyboard. Made possible

by Casio's innovative sheet key technology, this handy feature makes complicated scientific equations easier to solve because the major function keys are displayed oversize on their own keyboard.

Like our FX-90, our FX-450 (\$34.95) has a 10 + 2-digit LCD display and a keyboard with touch-sensitive keys. But the keys are double size and the number of functions increases to 68. Most importantly, it lets you calculate with the speed of light—and eight other commonly used physical constants, including

Plank's constant and atomic mass. It also makes computer math calculations and conversions in binary, octal and hex equally easy to use.

Casio has more space-age instruments at down-to-earth prices than there is space for here. However your Casio dealer will gladly let you get your hands on technology that, until now, only the future held.

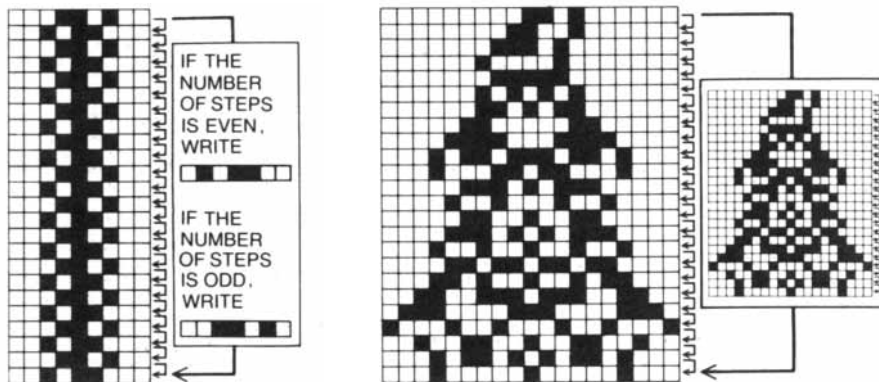
CASIO
Where miracles never cease®

quired. The ultimate form of a pattern is the result of an infinite number of steps, corresponding to an infinite computation; unless the evolution of the pattern is computationally reducible, its consequences cannot be reproduced by any finite computational or mathematical process.

The possibility of undecidable questions in mathematical models for physical systems can be viewed as a manifestation of Gödel's theorem on undecidability in mathematics, which was proved by Kurt Gödel in 1931. The theorem states that in all but the simplest mathematical systems there may be propositions that cannot be proved or disproved by any finite mathematical or logical process. The proof of a given proposition may call for an indefinitely large number of logical steps. Even propositions that can be stated succinctly can require an arbitrarily long proof. In practice there are many simple mathematical theorems for which the only known proofs are very long. In addition the cases that must be examined to prove or refute conjectures are often quite complicated. In number theory, for example, there are many cases in which the smallest number having some special property is extremely large; the number can often be found only by testing each whole number in turn. Such phenomena are making the computer an essential tool in many mathematical investigations.

Computational irreducibility implies many fundamental limitations on the scope of theories for physical systems. It may be possible to model a system at many levels, from simulating the motions of individual molecules to solving differential equations for overall properties. Computational irreducibility implies there is a highest level at which abstract models can be made; above that level results can be found only by explicit simulation.

When the level of description becomes computationally irreducible, undecidable questions also begin to appear. Such questions must be avoided in the formulation of a theory, much as the simultaneous measurement of the position and velocity of an electron—impossible according to the uncertainty principle—is avoided in quantum mechanics. Even if such questions are eliminated, there is still the practical difficulty of answering questions that in principle can be answered. The degree of difficulty depends strongly on the nature of the objects involved in the simulation. If the only way to predict the weather were to simulate the motions of every molecule in the atmosphere, no practical calculations could be carried out. Nevertheless, the relevant features of the weather can probably be studied by considering the interactions of large volumes of the



COMPUTATIONAL IRREDUCIBILITY is a phenomenon that seems to arise in many physical and mathematical systems. The behavior of any system can be found by explicit simulation of the steps in its evolution. When the system is simple enough, however, it is always possible to find a short cut to the procedure: once the initial state of the system is given, its state at any subsequent step can be found directly from a mathematical formula. For the system shown schematically at the left, the formula merely requires that one find the remainder when the number of steps in the evolution is divided by 2. Such a system is said to be computationally reducible. For a system such as the one shown schematically at the right, however, the behavior is so complicated that in general no short-cut description of the evolution can be given. Such a system is computationally irreducible, and its evolution can effectively be determined only by the explicit simulation of each step. It seems likely that many physical and mathematical systems for which no simple description is now known are in fact computationally irreducible. Experiment, either physical or computational, is effectively the only way to study such systems.

atmosphere, and so useful simulations should be possible.

The efficiency with which a computationally irreducible system can be simulated depends on the computational sophistication of each step in its evolution. The steps in the evolution of the system can be simulated by instructions in a computer program. The fewer the instructions needed to reproduce each step, the more efficient the simulation. Higher-level descriptions of physical systems typically call for more sophisticated steps, such as single instructions in higher-level computer languages correspond to many instructions in lower-level ones. One time step in the numerical approximation of a differential equation that describes a jet of gas requires a computation more sophisticated than the one needed to follow a collision between two molecules in the gas. On the other hand, each step in the higher-level description given by a differential equation accounts for an immense number of steps in the lower-level description of molecular collisions. The resulting gain in efficiency more than makes up for the fact that the individual steps are more sophisticated.

In general the efficiency of a simulation increases with higher levels of description, until the operations needed for the higher-level description are matched with the operations carried out directly by the computer doing the simulation. It is most efficient for the computer to be as close an analogue to the system being simulated as possible.

There is one major difference between most existing computers and physical systems or models of them: computers process information serially, whereas

physical systems process information in parallel. In a physical system modeled by a cellular automaton the values of all the cells are updated together at each time step. In a standard computer program, however, the simulation of the cellular automaton is carried out by a loop that updates the value of each cell in turn. In such a case it is straightforward to write a computer program that performs a fundamentally parallel process with a serial algorithm. There is a well-established framework in which algorithms for the serial processing of information can be described. Many physical systems, however, seem to require descriptions that are essentially parallel in nature. A general framework for parallel processing does not yet exist, but when it is developed, more effective high-level descriptions of physical phenomena should become possible.

The introduction of the computer in science is comparatively recent. Already, however, computation is establishing a new approach to many problems. It is making possible the study of phenomena far more complex than the ones that could previously be considered, and it is changing the direction and emphasis of many fields of science. Perhaps most significant, it is introducing a new way of thinking in science. Scientific laws are now being viewed as algorithms. Many of them are studied in computer experiments. Physical systems are viewed as computational systems, processing information much the way computers do. New aspects of natural phenomena have been made accessible to investigation. A new paradigm has been born.

Computer Software for Intelligent Systems

The key to intelligent problem solving lies in reducing the random search for solutions. To do so intelligent computer programs must tap the same underlying "sources of power" as human beings do

by Douglas B. Lenat

On your way to San Francisco one summer evening you come to an intersection in Nebraska. The road continues straight ahead. To your left the crossroad trails off through the cornfields; shielding your eyes from the sun, you see it doing the same to the right. Having no map, you might decide to pick one of the three roads at random and turn, say, to the left. Soon you would reach another intersection, and then another, and would be forced to make a series of random choices; at some point you would hit a dead end and would have to return to the preceding intersection and strike out on a different route. If you were both long-lived and extremely lucky, you might eventually reach San Francisco, but the odds against it would probably be on the order of 10^{30} to one. Because you know something about the world, however, you are not forced to pick a random path through the countryside, and at the first intersection you take a right.

Most problems, including many far more interesting than this one, can be cast in the same form: as the search for a path from some initial state to a desired final state. Most interesting problems also share the characteristic that they are too complex to be solved by random search, because the number of choices increases exponentially as one proceeds from the first intersection, or decision point. The classic example of this is chess, in which the number of possible board positions has been estimated at 10^{120} . A good player, however, reduces the problem of choosing his next move to manageable proportions by considering only 100 or so positions, corresponding to the most promising lines of attack. Therein, as I see it, lies the essence of intelligence: finding ways to solve otherwise intractable problems by limiting the search for solutions.

For about 30 years a small community of investigators has been trying, with varying degrees of success, to program

computers to be intelligent problem solvers. By the mid-1970's, after two decades of humbly slow progress, workers in the new field of artificial intelligence had come to a fundamental conclusion about intelligent behavior in general: it requires a tremendous amount of knowledge, which people often take for granted but which must be spoon-fed to a computer. Standing at the Nebraska intersection, a human traveler would know that San Francisco lies to the west of Nebraska, that in the evening the sun is in the western sky and that by heading toward the sun one would be heading in the right general direction. He would thus not have to try the other two possible paths.

Moreover, the relative simplicity of this problem is not representative of other everyday tasks that people complete without a second thought. Understanding even the easiest passages in common English, for example, requires a knowledge of the context, the speaker and the world at large that is far beyond the capabilities of present-day computer programs. The central role of knowledge in intelligence explains why the most successful programs so far have been "expert systems," which operate in highly specialized domains, such as the diagnosis of meningitis, and game-playing programs. In contrast, early efforts to design a "general problem solver" assumed that the core of intelligence lay in

a reasoning ability that could be applied across all domains. These efforts proved less fruitful and for the most part have now been abandoned.

In attacking a complex problem people draw on various methods—of using their knowledge of the world's regularities to constrain the search for a solution. They may invoke mathematical theorems or less formal rules of thumb; they may break up the problem into more tractable subproblems, or they may reason by analogy to problems that have already been solved. To the extent that computer programs already exhibit intelligence it is because they draw on some of these same sources of power. The future of artificial intelligence lies in finding ways to tap those sources that have only begun to be exploited.

Many programs written in the first two decades of artificial-intelligence research depended heavily on formal reasoning methods. When a task is well defined in a highly constrained domain, such methods can provide powerful ways of pruning or even eliminating the search tree. For example, no one need ever again waste time searching for a way to trisect an angle or seeking the best method of calculating the motion of a projectile, because well-established theorems and algorithms have settled those questions once and for all.

INTELLIGENT PROGRAM is seen running on a machine that displays various aspects of its operation in different windows. The program, called EURISKO, was written by the author and his colleagues and has been applied to a number of topics, including those named in the small window near the bottom of the display; here it is engaged in designing a fleet of ships to compete in Traveller T.C.S., a war game. The program's knowledge base includes the complex rules of the game as well as general heuristics to guide it in its search for ever better designs. EURISKO has just finished simulating a battle in which "side1" decisively defeated "side2," and the current heuristic has directed it to learn from the results of the battle by analyzing the differences between the two fleets to determine the cause of side1's victory. Because the main difference is that side1 has only one type of ship, EURISKO hypothesizes that it is desirable to minimize the number of ship types and suggests an experiment to test the hypothesis. In competition EURISKO's fleet, made up predominantly of small, fast ships, defeated the fleets of human players.

Prompt Window

This task is taking a long time; type a few ^T's if you would like me to move on to a new one.

Top level typescript window

Beginning task 459-110: Analyze the DifferenceBetween side1 and side2 in the recent TravellerFleetBattleGame played, looking for Cause.

The main difference is that side1 has ships of one type, while side2 has ships of nine types. 4 other hypotheses.

1 is more likely to be special than 9. So consider: If designing a fleet for TFBG, minimize the types of ship.

Experiment: reduce side2's number of ships, and increase side1's number of ship types.

Current-Heuristic

H161: (IF the current task was to find an Applic of a Game, THEN try to learn from the results)

*** CONDITIONS ***

IfPotentiallyRelevant: (Playing (a Game))
IfFinishedWorkingOnTask: (a GamePlaying)
IfResultsSatisfied: (a Decisive Victory)

*** ACTIONS ***

ThenCompute: (DifferenceBetween side1 side2)
ThenPrintToUser: (Guessed the causes in the recent --)
ThenAddToAgenda: (Analyze the differences for Cause)

*** DESCRIPTIONS ***

IsA: (Heuristic Op Anything MultiValuedOp AbstractOp)
Worth: 542
Abbrev: (It's worth finding out why one --)
Arity: 1
InitialWorth: 400
LastRunOn: PlayTravellerFleetBattle
ThenAddToAgendaRecord: (7560 . 6)
ThenPrintToUserRecord: (6726 . 21)
OverallRecord: (351537 . 21)
ThenComputeFailedRecord: (501 . 1)
ThenComputeRecord: (314972 . 15)
Generalizations: (ProtoOp)
FocusTask: (FocusOnH61)

- TOPICS
ElemMathematics
Heuristics
Representation
OilSpills
Programming
Games
DevicePhysics
Plumbing
PlaneTessellation

Current-Agenda

NumberOfTasks: 1307

- (RunMore TravellerFleetBattle)
7 Reasons
(Analyze Battle821)
7 Reasons
(Analyze Battle815)
2 Reasons
(Mutate ShipType37 (Increase Agility))
7 Reasons
(Analyze Battle816)
5 Reasons
(Mutate TicTacToe (Increase Complexity))

Current-Concept

Worth: 613
NPlayers: 2
InitialWorth: 650
ToPlay: (PlayTravellerFleetBattle)
Rules: (TravellerRules
RulesOfFairPlay)
IsA: (Game Anything TwoPersonGame
FairGame WarGame FleetBattle
DiceGame)

... at the moment ...

Adding a task to the Agenda, to Analyze the differences (i.e., between side1 and side2) for Cause.

Current-Task

RunMore-TravellerFleetBattle

Priority: 873
IsA: (Task SimulationTask
GameTask)
ConceptToWorkOn: TravellerFleetBattle
AspectToWorkOn: Play
CreditTo: (Heuristic85
Heuristic13 TheUser)
PastHistory: ((Run112 Task 1203
Created (ShipType30 ShipType31
& 10 others)))
NReasons: 7
Reasons: ((Because it is a
valuable concept) (Because the
user is interested in it) (
Because in the past it led to
useful new --) (Because there
is not much else interesting to
do --))

Plus 4
properties which are not slot names:
(NReasons PastHistory)

One of the most popular formal methods has been logical deduction by means of a proof technique called resolution, which proves by refutation. To apply resolution one must first convert the statement to be proved into the logical formalism of predicate calculus. The

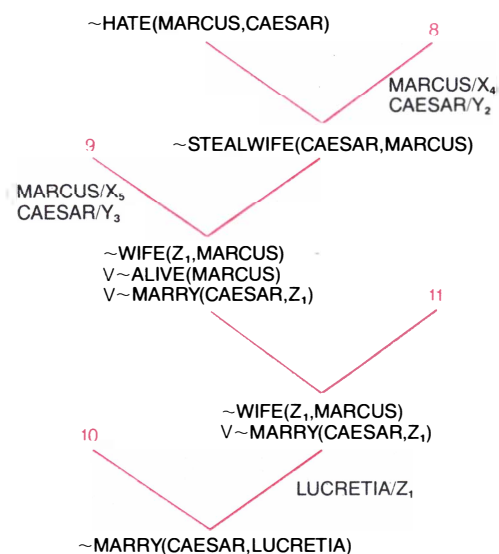
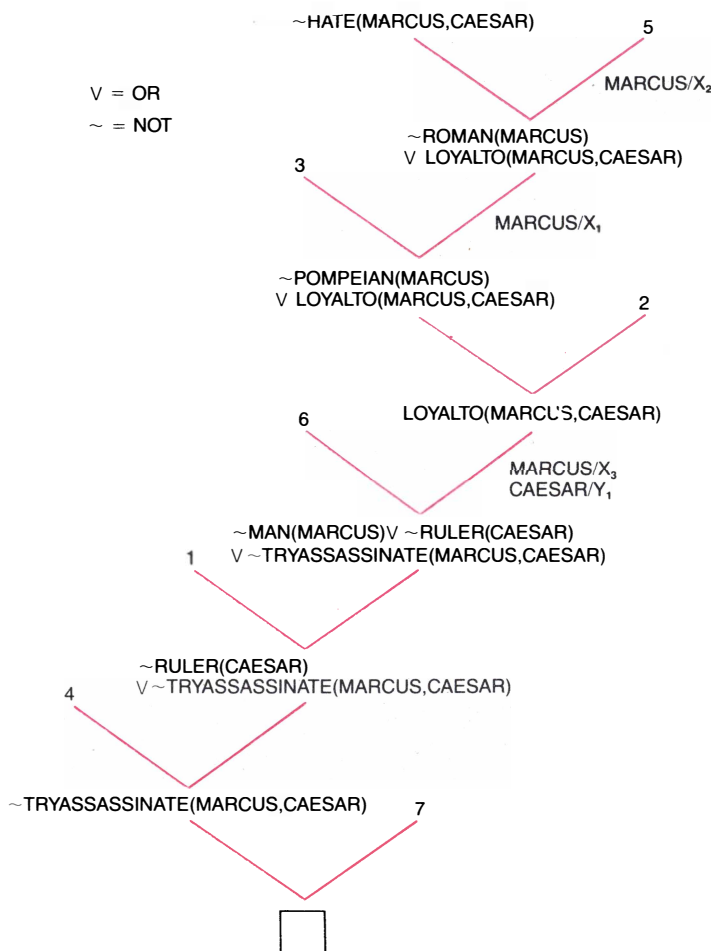
statement is then negated, and the negation is "resolved" with a series of axioms: statements known to be true for the particular problem area. If the inferences drawn by combining the negation with the axioms yield a contradiction, the negation must be false and the original

statement therefore must be true.

In 1964 J. A. Robinson showed that the resolution method is "complete": in every case it will eventually generate a contradiction if the original theorem is true. (If the theorem is false, the series of inferences produced by resolution is not

"DID MARCUS HATE CAESAR?"

<ol style="list-style-type: none"> 1 MAN(MARCUS) 2 POMPEIAN(MARCUS) 3 \sim POMPEIAN(X_1) \vee ROMAN(X_1) 4 RULER(CAESAR) 5 \sim ROMAN(X_2) \vee LOYALTO(X_2, CAESAR) \vee HATE(X_2, CAESAR) 6 \sim MAN(X_3) \vee \sim RULER(Y_1) \vee \sim TRYASSASSINATE(X_3, Y_1) \vee \sim LOYALTO(X_3, Y_1) 7 TRYASSASSINATE(MARCUS, CAESAR) 8 \sim STEALWIFE(Y_2, X_4) \vee HATE(X_4, Y_2) 9 \sim WIFE(Z_1, X_5) \vee \sim ALIVE(X_5) \vee MARRY(Y_3, Z_1) \vee STEALWIFE(Y_3, X_5) 10 WIFE(LUCRETIA, MARCUS) 11 ALIVE(MARCUS) 	<ol style="list-style-type: none"> 1 Marcus was a man. 2 Marcus was a Pompeian. 3 All Pompeians were Romans. 4 Caesar was a ruler. 5 All Romans were either loyal to Caesar or hated him. 6 People only try to assassinate rulers they are not loyal to. 7 Marcus tried to assassinate Caesar. 8 A person hates someone who steals his wife. 9 If the wife of a man who is alive marries a second man, then the second man stole the first man's wife 10 Lucretia was Marcus's wife 11 Marcus was alive
---	--



RESOLUTION, a proof technique of formal logic, can be used to deduce answers to questions, but it is prohibitively time-consuming when the problem is complex. Resolution proves by refutation: a hypothesis is shown to be valid by showing that its negation, when it is compared with axioms, or statements known to be true, leads to a contradiction. First the negated hypothesis and the axioms are converted into logical notation: clauses consisting of a disjunction of terms called literals. The set of axioms is then searched for one including a literal that, after appropriate substitutions have been made for variables, contradicts a literal in the negated hypothesis. When the two statements are "resolved," the contradictory literals cancel. This procedure is subsequently repeated with the resulting statement;

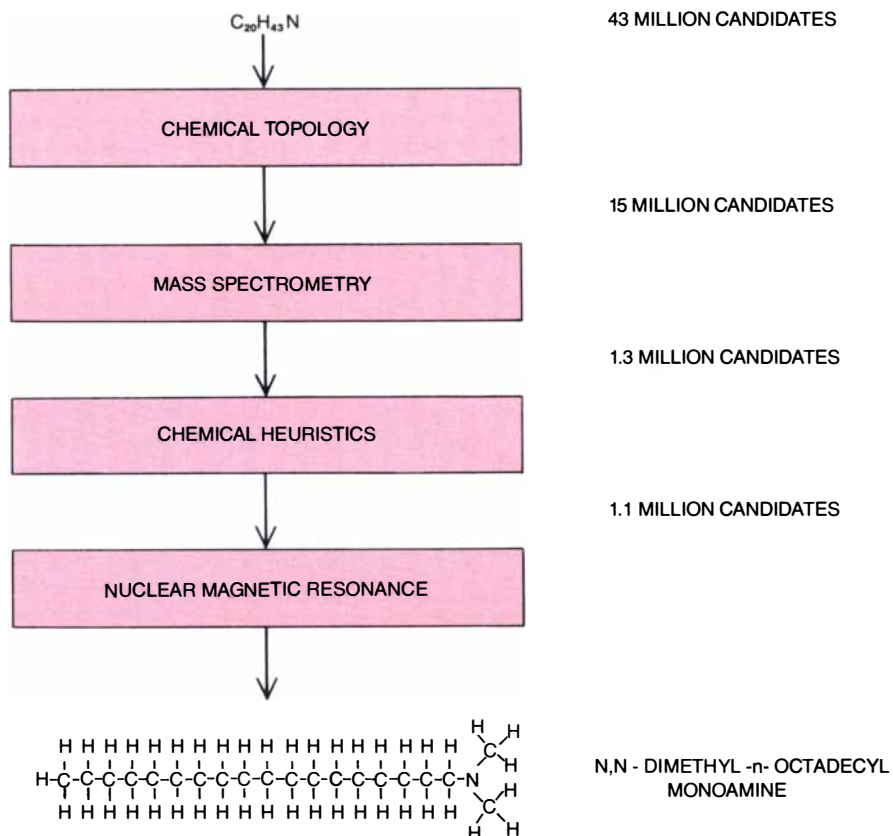
eventually, if the original hypothesis was valid, the process terminates in an unadorned contradiction. In the example shown the hypothesis is "Marcus hated Caesar." In the idealized case in which little is known about the world (*left*), only one axiom (5) includes a literal contradicting the negation of the hypothesis, and a computer program can quickly complete a proof. When more information is available (*colored axioms*), including a different cause of hatred (8), the program may choose the wrong axiom and proceed to a dead end at which no contradiction has been generated (*right*). In a real-world problem the number of possible choices is an exponential function of the number of axioms, which is very large, and finding a solution by blind search is infeasible. The example was provided by Elaine Rich.

guaranteed to end.) Robinson's work touched off a decade-long surge of activity in applying resolution and other closely related formal methods to automatic theorem proving. It turns out that computer programs are capable of proving statements of moderate difficulty and that the resolution method can also be adapted to programs whose purpose is to answer questions rather than to prove theorems. The great flaw of the resolution method is that it is subject to "combinatorial explosion": the number of resolutions the program must attempt increases exponentially with the complexity of the problem. Programs that successfully apply resolution to small test cases have consistently failed to "scale up" to more interesting real-world problems.

The same difficulty plagues software based on a different logical technique called structural induction. Such programs are given a large amount of data on the objects in a particular domain and told to construct a decision tree for discriminating between objects. The problem with structural induction algorithms, however, is that they incorporate no information enabling them to decide which variables are important or to deal with noisy data or exceptional cases. If the number of objects and associated features in a domain is large, the decision tree generated by the program becomes unwieldy.

For formal reasoning to work as the sole source of power in a program, the problem must be small. One application of a formal method that may prove fruitful in the near future is in the simulation of qualitative physical reasoning. John Seely Brown and Johan de Kleer of the Xerox Palo Alto Research Center have written a program that models changes in a pressure-regulating valve by means of qualitative equations. If the program is told, say, that the pressure on the left side of the valve has increased, the equations are changed accordingly and the program predicts the pressure change on the other side of the valve and the eventual equilibrium state of the system. This is a very simple example, but a similar approach is being used in the analysis and design of electrical circuits.

Most interesting problems, however, cannot be solved by relying on formal reasoning alone. The power of logical methods lies in their representation of the world in symbols that can be manipulated in well-understood ways (such as resolution) to produce inferences. That power is also their greatest weakness: many types of knowledge, including the uncertain and incomplete knowledge characteristic of most real-world problems, do not lend themselves to representation through precise logical formalisms. Programs that draw exclusively on logic are capturing only part of the understanding an intelligent person



EXPERT SYSTEMS exploit knowledge of a specialized domain to constrain the search for solutions. DENDRAL, the first expert system and one of the most successful, relies on the synergistic interaction of four types of knowledge to narrow the field of candidate structures for a particular organic molecule. If only the molecular formula $C_{20}H_{43}N$ is known, 43 million configurations of the atoms are possible mathematically. Knowledge of basic chemical topology, such as the fact that a carbon atom has four bonds, reduces the number of candidates to 15 million. The molecule's fragmentation pattern in a mass spectrometer, along with heuristic knowledge of what structures are most stable and thus most plausible, further limits the search. Finally, nuclear-magnetic-resonance data enable DENDRAL to identify the correct structure.

would bring to bear in attempting to solve a difficult problem.

Today there are dozens of large programs at work on difficult technical problems in fields as diverse as medical diagnosis, the planning of genetic experiments, geologic prospecting and automotive design. The primary source of power in these expert systems is informal reasoning based on extensive knowledge painstakingly culled from human experts. In most of the programs the knowledge is encoded in the form of hundreds of if-then rules of thumb, or heuristics. The rules constrain search by guiding the program's attention toward the most likely solutions. Moreover—and this distinguishes the heuristically guided programs from those relying on more formal methods—expert systems are able to explain all their inferences in terms a human being will accept. The explanation can be provided because decisions are based on rules taught by human experts rather than on the abstract rules of formal logic.

Consider MYCIN, a program developed by Edward H. Shortliffe of Stan-

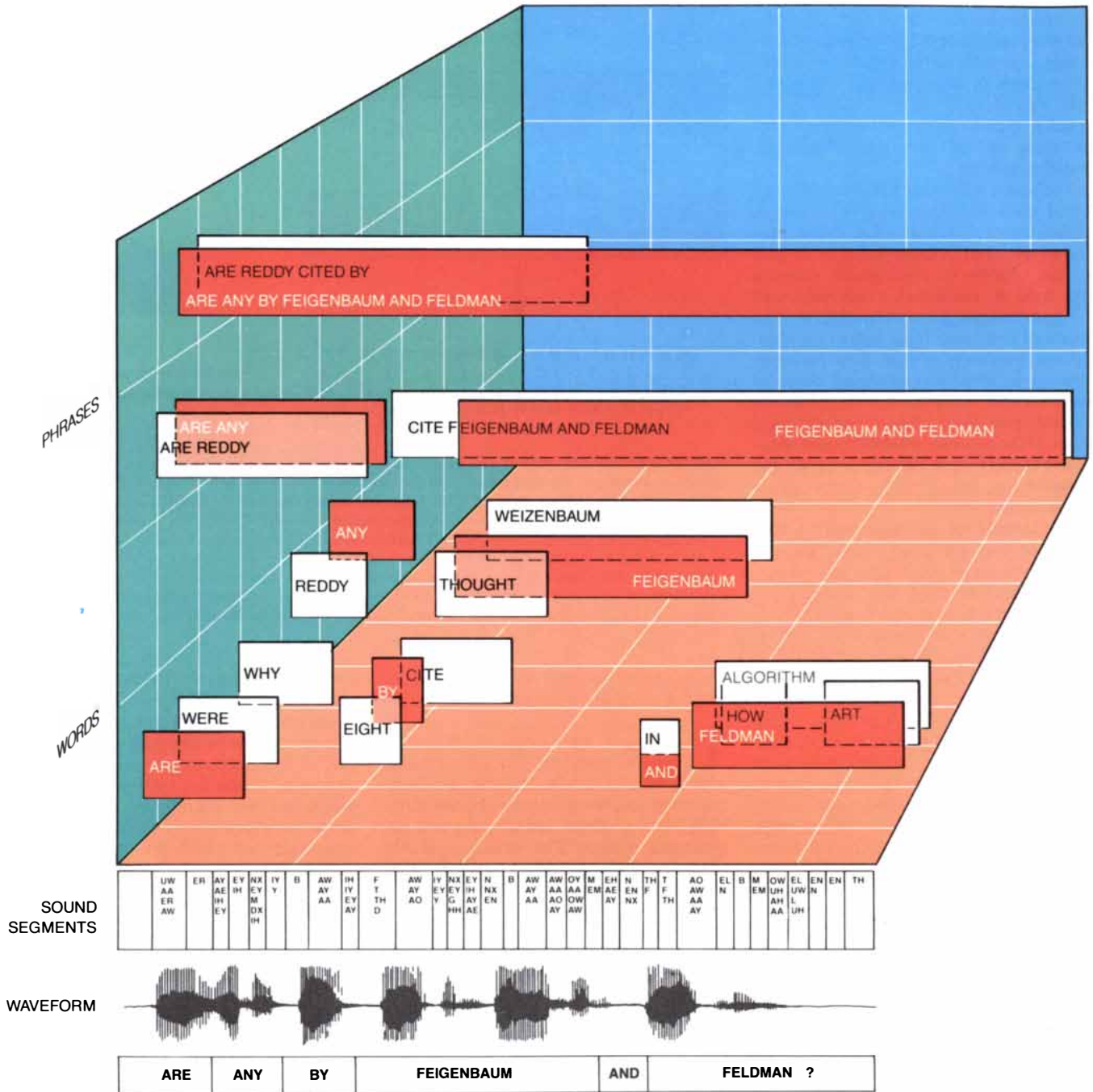
ford University to diagnose bacterial blood infections. The problem it faces is to determine which of many possible organisms might be responsible for a particular infection and to recommend a course of treatment on the basis of its diagnosis. To accomplish this MYCIN draws on a knowledge base of 500 heuristic rules, of which the following is a typical example: "If (1) the stain of the organism is gram-positive and (2) the morphology of the organism is coccus and (3) the growth conformation of the organism is clumps, then there is suggestive evidence (.7) that the identity of the organism is staphylococcus." As the program operates it converses with the user, asking for additional information on the patient that will allow it to apply different rules, and sometimes suggesting laboratory tests. At any time the user may ask MYCIN to justify a question or an inference by referring to the rule it is invoking. The program has shown itself capable of performing on a par with human practitioners.

In addition to heuristic reasoning, expert systems tap other sources of power, some of which are such staples of com-

mon sense that people rarely think of them consciously. Many programs, for instance, are able to focus their search by virtue of being oriented toward more or less specific goals. Again MYCIN is a good example. Starting with general information on the patient, MYCIN reasons backward from its goal of finding the identity of the disease-causing organ-

ism, asking questions to ferret out the specific symptoms that might substantiate a diagnosis. Having determined, say, that "the stain of the organism is gram-positive," MYCIN would without further search inquire about the morphology of the infecting organism, in the course of deciding whether the bacteria might be staphylococci. Such goal-directed-

ness does not mean a program has had its decision sequence "wired in," as is sometimes suggested by those who argue that expert systems do not really show intelligence; programs such as MYCIN are actually adaptable to situations unforeseen by the programmer, who does not strictly predetermine the use a program will make of its knowledge.



BLACKBOARD provides a way to organize a large amount of knowledge in an intelligent program. The information is stored in independent modules, each of which monitors only a small region of the blackboard and is activated only when entries are posted in that region by another module. The modular design helps to solve the problem of deciding which part of the knowledge base to apply at a given moment. In the example shown, adapted from a speech-understanding system developed by Raj Reddy, Lee D. Erman and their colleagues, the horizontal axis represents time, starting at the beginning of an ut-

terance, and the vertical axis represents the level of abstraction, starting with the sound waves and proceeding to the complete sentence. The third dimension indicates the level of certainty associated with each hypothesis posted on the blackboard; the most plausible of the many hypotheses possible at each time and at each level of abstraction are near the front of the cube. The blackboard allows knowledge modules at different levels to interact; for instance, once the program infers from pitch that the sentence is a question, that information guides the formation of hypotheses at the word or the phrase level.

Another powerful strategy exploited by intelligent human beings, including software designers, is to break up a complex problem into more tractable subproblems: it is the strategy of divide and conquer. A group at Carnegie-Mellon University has built four related programs, each of which is guided by heuristics to rediscover well-known physical and chemical laws. The programs are enabling the group to make progress toward understanding and mechanizing different aspects of scientific theory formation; eventually the workers will knit these solutions together into a single model of the entire process.

In a relatively narrow sense the divide-and-conquer approach is implicit in intelligent software itself. Goal-directed programs divide the search into more or less independent subgoals (nodes in the search tree). At a higher level, heuristically driven systems distinguish the problem itself from the meta-problem: the difficulty of deciding at any given moment which of hundreds of different rules should be "firing." The meta-problem is solved separately by an often complex process—sometimes requiring its own heuristics—of matching the search state with the preconditions, or "if" part, of an if-then rule.

Formal methods, although they are not the engine of inference, can also be helpful in managing an expert system. Some systems, for example, rely on logical or statistical procedures in deciding when it is no longer cost-effective to continue a search. Furthermore, since the if-then heuristics in an expert system generally do not express relations that are always true, each rule may have a confidence rating associated with it ("7" in the above example from MYCIN). The ratings linked to each step in a sequence of inferences are combined to produce a confidence measure for the final conclusion. This is done using Bayes' law or some other formal procedure of probability theory.

Each rule in an expert system may be simple, and sometimes there may be little or no organization among rules. Still the set as a whole is capable of performing difficult technical tasks with an expert's level of competence. This is a form of synergy, the whole being greater than the sum of its parts. Synergy is so pervasive that it is taken for granted, but almost all expert systems rely on it as a source of power.

One of the most successful intelligent programs was also the first expert system to be developed: DENDRAL, written by Edward A. Feigenbaum and his colleagues at Stanford in the late 1960's. Along with its successor, GENOA, it is now in use in organic-chemistry laboratories throughout the world. DENDRAL deduces the structure of organic mole-

cules from mass spectra, nuclear-magnetic-resonance data and other kinds of information.

Like MYCIN, DENDRAL is essentially diagnostic. A different type of expert system altogether is one that seeks to discover new information, or to rediscover from basic principles information already known. An example of such a program is EURISKO, which I developed with my students at Stanford. After giving it a relatively small amount of basic information on a subject, we have turned EURISKO loose in domains as diverse as set theory, a war game, computer programming and the cleanup of chemical spills.

EURISKO is guided in its search, which consists of synthesizing, analyzing and evaluating new concepts, by hundreds of fairly general heuristics. One of these is "look at extreme cases." This heuristic led EURISKO, while it was pondering the function "divisors of" in set theory, to consider numbers that have only a few divisors. In doing so it rediscovered prime numbers, which are numbers with only two divisors, as well as the fact that any number can be factored into a unique set of primes. The same simple heuristic also proved invaluable when EURISKO and I entered the national war game Traveller T.C.S., in which the object is to design a fleet that will defeat one's opponents in battles waged under a large set of rigid rules. After considering the rules EURISKO produced a fleet consisting almost entirely of small, swift attack vessels rather like PT boats and including one ship so fast and so tiny that it was virtually indestructible. Human Traveller players scoffed at this strategy and fielded more conventional fleets with a balance of ship sizes; EURISKO won.

Another widely applicable heuristic is "coalesce," which leads the program to consider what happens to a function of two variables x and y when the variables are assigned the same value. After EURISKO had already derived the functions of addition and multiplication from set theory, the coalesce rule prompted it to discover doubling (x plus x) and squaring (x times x). Applying "coalesce" to Traveller, EURISKO developed a novel strategy: it directed a disabled ship to fire on and sink itself. Because the game's stylized rules defined overall fleet agility in terms of the slowest vessel, this was a reasonable method of increasing the fleet's power. Finally, in studying computer programming EURISKO considered the function " x calls y ," where x is a program element that activates y , another element. The coalesce heuristic led EURISKO to define the important notion of recursive programming elements, or components of a piece of software that call themselves.

"Coalesce" and "look at extreme cases" are examples of heuristics that guide

a discovery program to define new concepts. If the program's mission is taken to be the search for interesting ideas, it must also have a second type of heuristic to help it decide which of the many concepts it generates are significant. Concept-synthesis rules steer search at the outset; evaluation heuristics channel the search along worthwhile paths once it has begun. EURISKO includes rules of the form "If all members of a set unexpectedly satisfy some rare property, then increase the 'interestingness' rating of that set and of the heuristics that led to its definition." Another rule directs the program, when it is deciding which of two very similar concepts to study, to pick the one that requires less computer time or questioning of the user.

From using heuristics to discover (or rediscover) new concepts or facts it is a short theoretical step to using them to discover new heuristics. The latter endeavor relates to what has long been a central goal of artificial-intelligence research: writing programs that learn from experience. In recent years a number of workers have in fact developed programs that draw general rules from their experience in solving individual problems. The generalization process is controlled by meta-heuristics.

The success of DENDRAL prompted its authors to write a new program, META-DENDRAL, which formulates general rules of mass spectrometry based on observations of how particular compounds are fragmented in the spectrometer. An example of a meta-heuristic in this case is the simple statement that the features of a molecule that are most important in determining its fragmentation pattern are those near the break points. Applying this heuristic, META-DENDRAL might formulate a rule to the effect that organic molecules tend to break where carbon and oxygen atoms are linked by single bonds. The new heuristic would then be helpful in deducing the structure of unknown molecules from their mass spectra. Similarly, Thomas M. Mitchell and Paul E. Utgoff of Rutgers University have written a program called LEX2 that derives problem-solving heuristics in integral calculus from its experience in computing particular integrals.

Designing more proficient learning programs depends in part on finding ways to tap a source of power at the heart of human intelligence: the ability to understand and reason by analogy. A little introspection and an attentive ear are all it takes to realize that people draw on analogy constantly in explaining and understanding concepts and in finding new ones. This source of power is only beginning to be exploited by intelligent software, but it will doubtless be the focus of future research.

I do not mean to suggest that no prog-

BMDPC STATISTICAL SOFTWARE

CONFIDENCE AND CONVENIENCE



BMDP[®] is Data Analysis with Confidence. Researchers and statisticians know quality research demands quality analysis. That's why they rely on BMDP Statistical Software for all their data analysis needs.

BMDP is a comprehensive library of dependable statistical programs. You can use BMDP to perform many types of analysis. Capabilities of BMDP range from simple data display and description to the most advanced statistical techniques. In fact, BMDP provides the widest range of statistical approaches required to effectively analyze your research data:

- Data Description
- Data in Groups
- Plots and Histograms
- Frequency Tables
- Missing Values
- Nonlinear Regression
- Analysis of Variance and Covariance
- Regression
- Multivariate Analysis
- Nonparametric Analysis
- Cluster Analysis
- Survival Analysis
- Time Series

Powerful statistical and computing procedures make BMDP especially

appealing to the experienced data analyst. However, you need not be a seasoned statistician or computer expert to perform data analysis with confidence. English-based instructions make BMDP easy to learn and easy to use.

As the acknowledged leader in statistical methodology, BMDP has become the standard by which others measure their performance. The BMDP programs are known for superior technology and state-of-the-art methodology. BMDP is committed to developing new statistical methods for data analysis, and continues to lead the way in providing superior statistical software.



BMDP is Data Analysis with Convenience. With our latest advance, you can even perform complex statistical analyses without a large computer. BMDPC offers you both the confidence of BMDP and the convenience of using your own PC.

The same programs that have been trusted for more than 20 years by researchers worldwide are now available for the IBM Personal Computer and compatibles.

Best of all, the BMDPC programs are available in low-cost subsets. So, you can pick and choose programs for your PC. Pick the types of analyses you use most, and choose from our broad selection of statistical programs.

Put the statistical software of choice on your IBM[®] PC^{*}, so you can perform data analysis with confidence and convenience.

Hardware Requirements

Minimum requirements for running BMDP on the PC include the following:

- a 5 megabyte hard disk
- the capacity for double-sided, double density floppy diskettes
- an 8087 floating point processor
- a PC/DOS 2.0 or later operating system
- 640 kilobytes of memory.

FOR MORE INFORMATION WRITE OR CALL

BMDP Statistical Software
1964 Westwood Blvd., Suite 202
Los Angeles, California 90025
Phone (213) 475-5700

ress has been made so far. Twenty years ago Thomas G. Evans of the Massachusetts Institute of Technology wrote a program capable of recognizing analogies between geometric figures, the kind of ability required by certain problems on I.Q. tests. Getting programs to find conceptual analogies is harder, and a number of investigators are working on this problem. Jaime G. Carbonell of Carnegie-Mellon has a program that recognizes the similarity between two algorithms that are written in different computer languages but have the same purpose. EURISKO, on the other hand, does not so much find analogies as use low-level analogical reasoning. In working on the design of integrated circuits, for example, EURISKO stumbled on the fact that symmetry is a desirable property for such chips, although it did not understand why; when it was later instructed to design fleets for the Traveller game, EURISKO decided to make them symmetrical and justified its decision by referring to its earlier experience in designing circuits.

Compared with human capabilities, however, this is extraordinarily meager. The poor performance of computer programs in finding and using analogies may be attributable more to the narrowness of their knowledge than to the inability of programmers to come up with suitable algorithms. People have an enormous store of concepts to draw on as possible analogues: perhaps a million distinct memories of objects, actions, emotions, situations and so on. This rich repertoire is not built into existing software, nor do programs have a chance to accumulate a large set of experiences from which to draw comparisons. Programs that run for a long time before they are stopped and restarted typically do not keep adequate records of their search experience, and when they are stopped, they lose all or most of the lessons they learned. Even EURISKO, which has run for weeks at a time and is restarted with most of its records intact, has had a short mental life, with experiences that are not nearly as varied as those of a human infant.

The prescription for improving a program's analogical reasoning is therefore the same as the one for raising the general performance of intelligent software: expand the knowledge base. Ideally an entire encyclopedia would somehow be stored in computer-accessible form, not as a text but as a collection of thousands of structured, multiply indexed units. Preliminary work toward this goal by a few investigators has revealed that it is even more elusive than it sounds: the understanding of encyclopedia articles itself requires a large body of common-sense knowledge not yet shared by computer software.

On the one hand, computer programs

"FRED IS LIKE A BEAR"

NAME	FRED		NAME	TYPICALBEAR
ISA	MAN		ISA	BEAR
HOME	15 MAIN ST.		HOME	CAVE
SIZE	LARGE	←	SIZE	LARGE
GAIT	LUMBERING	←	GAIT	LUMBERING
EATS	HONEY	←	EATS	HONEY
AGE			AGE	5
FINGERNAILSIZE	LONG	←	CLAWSIZE	LONG

FRAMES are a way of representing knowledge of a particular concept; among their other advantages, they can facilitate the drawing of analogies by an intelligent program. A frame consists of slots filled with attributes and associated values. If two objects have some of the same attribute names, an analogy can be drawn simply by filling empty slots in one frame with appropriate attribute values from the other frame. Heuristics guide the program in determining which values to transfer, leading it, for example, to consider extreme qualities of the source frame. When a slot in the source is absent in the target, the program may select a similar slot.

will have to become a lot more knowledgeable before they will be able to reason effectively by analogy. On the other hand, to acquire knowledge in such bulk it would seem that computers must at least be able to "understand" analogy when it is presented to them; certainly that is one of the most powerful learning techniques available to human beings. The problem is thus of the chicken-and-egg sort. Fortunately it is easier for a computer, as it is for a person, to understand an analogy put before it than to find one itself, and ongoing research gives some reason to hope that the dilemma will not prove completely intractable.

The key to getting a machine to understand an analogy is to represent information about the objects to be compared in a convenient way: for example, as frames consisting of sets of slots, where each slot contains a value for a particular attribute of an object. When the computer is told that two objects are analogous ("Fred is like a bear"), it can then simply fill in empty slots in one frame with values taken from the equivalent slots of the other. The hard part for an ignorant program, of course, is deciding which values it should transfer. (In which of his attributes is Fred like a bear?) Such decisions can be guided by heuristics. The "look at extremes" rule, for instance, is again useful; when an analogy is apt, it is often because certain unusual characteristics of the source apply to the target as well.

The use of frames in mechanizing the understanding of analogy illustrates a general fact, namely that the representation of knowledge can itself be a source of power in an intelligent system. A piece of knowledge can be represented in many ways in software, and I do not intend to go into them all here. The point is merely that each mode of representation makes it efficient to do certain operations and inefficient to do others.

Drawing analogies, for example, might entail a long and cumbersome search if each attribute of each object were represented in a program's knowledge base as a separate statement in formal logic. Choosing the right representation for a given problem reduces search.

Human beings, though, go well beyond a simple, one-time choice: we have the ability to switch back and forth between several forms of representation—words, symbols, pictures—and to look at a problem from different perspectives as we seek a solution. Such flexibility is difficult for software to emulate. In 1962 Herbert L. Gelernter designed a program that solved high school problems in plane geometry; each problem was represented both axiomatically and by a diagram. The logical representation enabled the program to construct formal proofs. The diagrams, on the other hand, suggested methods of proof and enabled the program to test conjectures; for instance, it could recognize when two line segments were parallel, when two angles were equal or complementary and so on. Although a coincidence of this kind could be an artifact of a particular diagram, the likelihood of such a coincidence was so small that it made the multiple-representation technique quite effective at eliminating search.

Unfortunately Gelernter's prescient thoughts about multiple representation have not yet been extended into other domains, although recently a few investigators have begun classifying forms of representation and working on techniques that would enable a program to convert from one form into another. The diagrams in Gelernter's program, however, were effective not only because they were a different form of representation but also because they were analogical: their pieces corresponded to real entities, and distances between pieces matched real distances, as on a road map. That is an advantage a logical representation cannot offer, and a number

of workers are looking for ways to exploit the potentially large power of analogical representation.

One line of this research deserves special mention. "Blackboards" are not a way of representing individual pieces of knowledge but of organizing the pieces into a large program; a blackboard represents the problem space itself. In speech understanding, to which this approach was first applied, the horizontal axis of the board represents time, with the beginning of a sentence at the left and the end at the right. The vertical axis measures the level of abstraction, which increases from sound wave to syllable to sentence as the program's understanding of an utterance progresses. Each if-then rule or set of rules in the program monitors a particular part of the blackboard and is triggered only when information is posted in that space; the blackboard thus helps to solve the meta-problem of deciding which rules should be firing at a given moment. Moreover, the knowledge modules, which operate

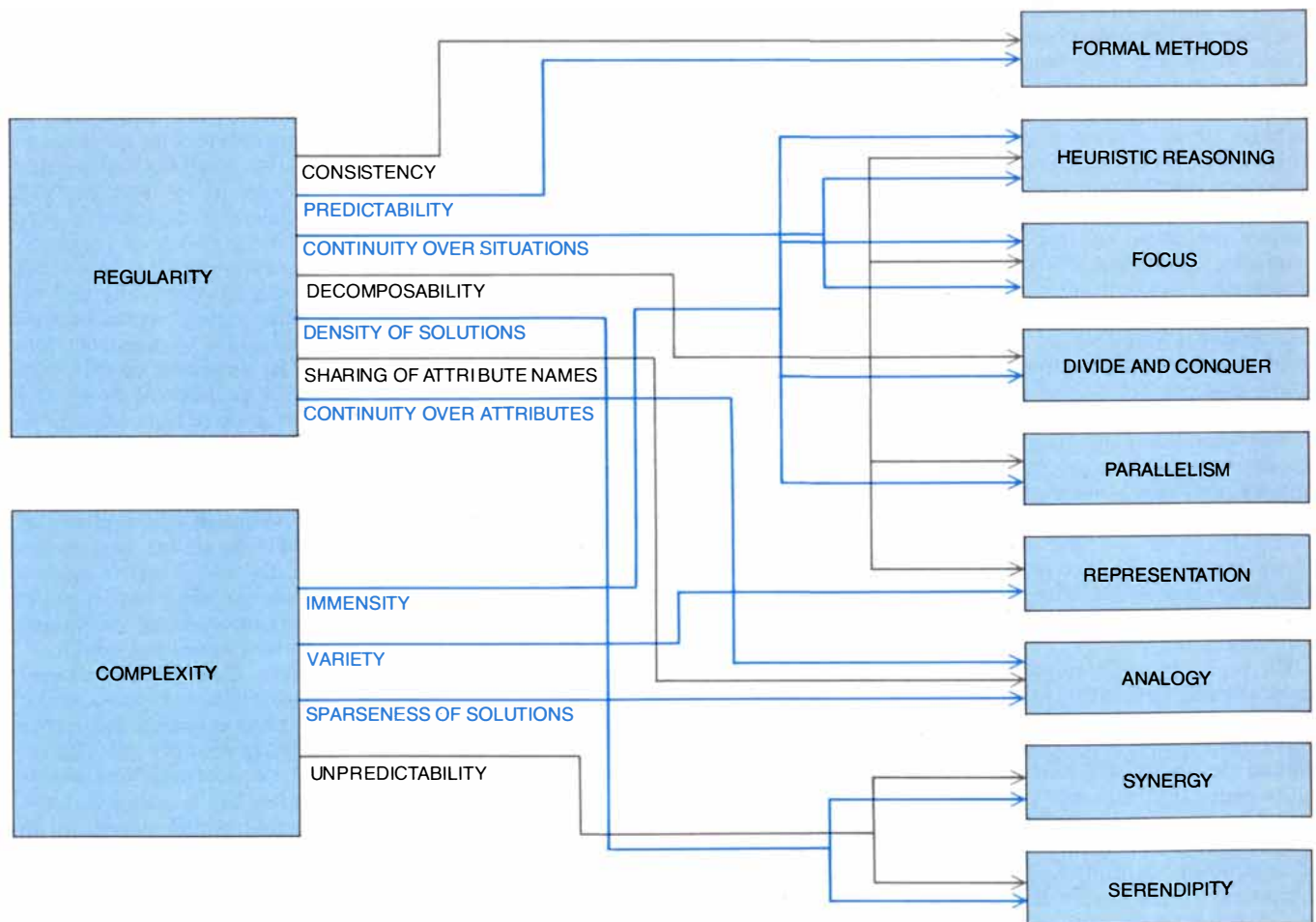
independently, need not all be if-then rules. A blackboard structure is therefore a natural way of exploiting the synergy among different types of knowledge in a single system.

Recently another source of power potentially available to intelligent systems has become something of a buzzword in artificial-intelligence circles: parallelism. At present most computers process information sequentially, one operation at a time. Several groups, however, including those working on the Japanese "fifth generation" and American "strategic computing objective" projects, are designing machines that will include on the order of a million processors operating in parallel. The possibility that processing speeds will increase by a factor of a million has prompted some workers to forecast revolutionary improvements in the performance of intelligent software.

The improvements will undoubtedly be significant. The rise in processing

speed may bring within reach the solutions to some interesting problems, such as getting a computer to understand speech as quickly as it is spoken; it should also be enough to enable a machine to beat the best human player at chess. Yet before predicting miracles from the fifth generation one should remember that most hard problems have search trees that grow exponentially. Even a millionfold increase in computing power will not change the fact that most problems cannot be solved by brute force but only through the judicious application of knowledge to limit the search.

A second reason for not treating parallel processing as a panacea is more subtle and is based on empirical evidence obtained by my colleagues and me. When we had EURISKO simulate the action of a progressively increasing number of parallel processors working simultaneously on tasks from its agenda, we found that once four processors had been simulated the rate at which



SOURCES OF POWER in problem solving are made meaningful (gray lines) and actually useful or cost-effective (colored lines) by certain properties of the problem domain. For example, it is possible to apply heuristic reasoning or the divide-and-conquer approach to a problem if the problem is regular in the sense that it can be decomposed into subproblems. It is cost-effective to do so, however, only if the domain is complex and immense; if the domain is more limited

and regular, it may be better to apply a formal logical approach. Similarly, analogies can be drawn most readily in a domain (such as medical diagnosis) in which objects (diseases) have many attribute names in common. Reasoning by analogy is cost-effective only when there is a continuity of attribute values (diseases with similar symptoms and causes often require similar treatment) and when a problem has few solutions (a set of symptoms is linked to only a few illnesses).

the program made significant discoveries did not increase further. The reason for this was that in completing its top-ranked task EURISKO usually discovered a new task it found more interesting than the rest of its original agenda. Where good heuristics allow such a "best first" search, parallel processing may have diminishing returns.

There is a final source of power in human problem solving that I should like to mention, at the risk of sounding tongue-in-cheek: serendipity. Although one cannot count on luck to solve specific problems, it is often reliable in the statistical sense. For example, Woodrow W. Bledsoe of the University of Texas at Austin has found it worthwhile to include the following "fortuitous accident" heuristic in his theorem-proving program: "Whenever a new proposition is deduced, regardless of whether it solves the current subproblem, check to see whether it solves any of the higher-level goals."

To a certain extent all empirical scientists rely on luck when they gather data in the hope of finding some pattern. Empirical learning programs such as EURISKO, whose mission it is to seek new concepts and regularities, depend on serendipity in the same way. This open-ended activity can be made less risky by confining the search to a problem space in which interesting findings are known to be densely packed. The full exploitation of serendipity, however, demands a willingness on the part of software designers and their sponsors to use programs whose performance is far from guaranteed. Although universities and corporations routinely do this with human scientists, it may be many years before they and program designers themselves lose their reluctance to take such chances with intelligent programs.

Each source of power I have described is made meaningful and applicable by certain properties of a problem domain and cost-effective by others. The two types of properties are rather like necessary and sufficient conditions for the application of a particular source of power. Consider analogy: it is meaningful between two concepts only if they share many of the same attribute names, and it is useful or cost-effective if in addition the concepts are actually similar in some of their qualities, that is, if certain of their attribute values are comparable. Most diseases, for example, have attribute names in common—"cause," "symptoms," "treatment" and so on—and it is therefore possible to draw analogies among them. It is useful to do so because illnesses that have similar causes frequently turn out to require similar treatments. In fact, medical students often learn about new diseases by analogy to ones they have already studied, and

medical-diagnosis programs may one day do the same.

One property common to many interesting problem domains is that of being immense. Immensity is usually regarded as a hurdle to be overcome, but it presents an opportunity to the problem solver as well. If the search space is large, pieces of it may be summarized in the form of statistics, theorems or heuristic rules. This opportunity does not arise in the case of problems that are not immense but are difficult in the sense of being time-consuming; the testing of drugs for long-term side effects is a good example.

When human beings are confronted with a complex problem, they intuitively draw on all appropriate sources of power. Early artificial-intelligence programs, in contrast, were seriously weakened by their reliance on a single approach, usually some formal method. Many software designers now recognize the importance of exploiting the gamut of human problem-solving techniques—as well as the synergy that arises when different sources of power are allowed to function together.

With the exception perhaps of synergy and serendipity, the sources of power I have discussed are all methods of organizing and applying knowledge to reduce search. If the future of artificial intelligence lies in making these human tools available to machines, it depends just as certainly on the ability of programmers to provide their systems with the right raw material: the huge knowledge base of facts and experience from which human beings reason. To a certain extent such knowledge can be incorporated in a system "by hand," with the programmer doing all the work. The duplication by machines of many of the most impressive human intellectual feats will remain impractical, however, until programs become more like human beings in two fundamental ways: in their ability to accumulate their own experiences over a long mental lifetime and in their ability to communicate with and learn from one another.

Designing software that fits this description is a tremendous challenge, but I believe it will be accomplished someday. Most existing programs were designed with a static environment in mind. In areas where the state of the art and therefore the problems are changing rapidly—computer architecture, integrated-circuit design and biotechnology are examples—this property is already revealing itself as a serious drawback: programs working in these problem areas quickly become obsolete. The ability to adapt to a changing environment demands intelligence. In my view intelligence will increasingly be perceived as a necessity rather than a luxury in computer software.

PROOFWRITER™

For the IBM, TI PC/XT's
and Compatibles

**Word Processor,
Program Editor,
and Spelling Checker**

For Scientific and
Multilingual Applications

Features:

- Full Screen Editor
- Foreign Language and Scientific Symbols easily entered and printed
- Simple "cut and paste" capabilities
- Easy to print expressions such as: \bar{X} \hat{A} A^i_j \underline{A}
- Equation Mode
- Equation Macro Storage: insert any stored equations with 3 keystrokes
- Mainly what is seen on screen is what is printed
- Files easily interfaced with mainframe/minicomputers
- Mail Merge
- Characters Generated on most matrix printers
- Extensive Footnote and Endnote capabilities
- View Scientific Symbols with Optional Character Proms. Proms available for Science, Math, Statistics, Western European, Greek, Russian, Hebrew (\$125)
- **PROOFWRITER** has most flexible printer interface of any word processor. Compatible with:
Matrix printers: Toshiba, TI, Epson, IBM, C. Itoh, Okidata, NEC, Gemini, Prowriter, IDS.
Impact printers: Diablo, XEROX, NEC, Brothers, Qume, and Daisywriter

PROOFWRITER \$250

**PROOFWRITER
INTERNATIONAL \$300**

with alternate keyboards
and 6 "dead" keys

Try it with

Demo Disk Tutorial \$5

2 disk drives/256 KB

VISA or MC accepted

IMAGE PROCESSING SYSTEMS

6409 Appalachian Way
P.O. Box 5016

Madison, Wisconsin 53705
U.S.A.

(608) 233-5033

verhältnismäßig $\int x^2 + y^2 dx dy$ $\frac{\partial X}{\partial Y} \times \left\{ \int_0^{\infty} e^{-wt} F(t-\bar{T}) dt \right\} = \sum_n A_n B_{ik}$

PERSONAL COMPUTING MAKES GOOD BUSINESS SENSE.

ONE WAY.

Read this... if you want to learn more about personal computers before you buy one.

Turn to the computing magazine for business people, not technical wizards. *Personal Computing*. Must reading if you want to know more about personal computers and what they can do for you... before you commit to buy.

Every month, *Personal Computing* will expand your understanding of today's most indispensable business tool. From the basics (How do you get started?). To the breakthroughs (How good is the Macintosh?). You'll hear if it's wise to plunge into computing or wait (our advice: get in now!). You'll see how people like you are putting personal computing power to work for them in their businesses.

You'll find answers to your most important questions... "How much of a system do I really need... how much should I spend... what will it do... is it easy to learn?" And more. You'll enjoy our monthly reviews of hardware and software. From PC to Lisa and everything in between. You'll get essential data on capabilities and prices—for all brands. So that when you're ready to buy, you can buy smart.

See for yourself. Please accept the next issue of *Personal Computing* with our compliments. Return the bound-in card (or coupon) and reserve your complimentary copy plus a subscription for 11 more issues.

OR THE OTHER.

Read this... if you want to get more out of the personal computer you already own.

Turn to the computing magazine for business people that's as much a user's guide as it is a source of price and product information. *Personal Computing*. Must reading if you want to fully capitalize on your computing investment.

Every month, you'll find out how you and your machine can become more productive. Through extensive articles on software. How to evaluate it, how to buy it, how to use it effectively. Insightful advice on programs for all kinds of business applications. DataBase Management. Spreadsheet. Word-processing. Communications. And much more.

You'll learn about all the new hardware on the market. Whether it's advisable to expand your system or trade-up to a new one. Get the lowdown on portables. See how peripherals can add punch to your presentations. Our monthly reviews make it easier to decide what's right for you.

You'll discover *unexpected* home uses. Like how to use your system for home security, paying bills, educating your children or even shopping by computer. *Personal Computing* will save you time and money.

See for yourself. Please accept the next issue of *Personal Computing* with our compliments. Return the bound-in card (or coupon) and reserve your complimentary copy plus a subscription for 11 more issues.



**PLEASE ACCEPT A
COMPLIMENTARY COPY.**

**PERSONAL
COMPUTING**

P. O. Box 2941
Boulder, CO 80322

5AXA4

Please send me my complimentary issue of *Personal Computing*. At the same time, enter my subscription for 11 more issues, for a total of 12. I understand that I will be billed at the Introductory Subscription rate of only \$11.97—a saving of \$18.03 off the single-copy cost of \$30.00 and \$6.03 off the regular subscription price. If I'm not completely satisfied with my complimentary issue, I will write "cancel" on the bill and return it, and that will end the matter. The complimentary issue is mine to keep in either case.

Please allow 4-6 weeks for delivery of your first issue. **Your subscription may be tax deductible.**

Name _____

Company Affiliation _____

Street _____

City _____

State _____ Zip _____

THE AMATEUR SCIENTIST

Success in racquetball is enhanced by knowing the physics of the collision of ball with wall

by Jearl Walker

A four-wall game such as racquetball, squash or handball demands of the player a great deal of skill in judging angles and bounces. The ball comes off the wall in a direction determined by the physics of the collision. An understanding of this physics enables a player to predict the ricochet of a ball approaching him and to calculate the ricochet he would like to achieve in order to put the ball out of the reach of his opponent. In discussing these phenomena I shall call to my aid some strange related tricks that can be demonstrated with a highly elastic solid ball sold in toy stores.

The toy ball is almost perfectly elastic: if you drop it, it bounces back nearly all the way to your hand. (A perfectly elastic ball would return to its initial height.) The ball also has a rough surface, so that when I throw it along the floor, it does not slip. Because of the ball's elasticity and roughness, it can be bounced in some surprising ways.

When I throw the ball downward at an angle, it bounces across the floor in a repeated pattern of high, short hops and low, long hops. If I put some spin on the ball as I throw it, it bounces to the left and right until it runs out of energy. The most startling demonstration involves throwing the ball under a table. A smooth ball would bounce between the table and the floor until it reached the far side of the table. A rough elastic ball bounces back to the thrower.

To study how a ball collides with a surface I first considered a uniformly solid ball bouncing on a floor. Suppose the ball approaches the floor moving to the right and downward. It helps to describe the velocity as being in two parts, one part parallel to the floor and the other part perpendicular. In addition the ball can be spinning about its center. A clockwise spin is a negative rotation and a counterclockwise spin is positive.

The ball's kinetic energy is in three parts, one part for each component of the velocity and one for the spin. If the ball is completely elastic, the collision does not change the total kinetic energy.

(The total kinetic energy is said to be conserved.) Only an ideal ball and collision follow this rule. In practice some kinetic energy is lost by being converted into other forms of energy. For example, some of it might end up in the vibrations of the ball. I shall ignore such losses and concentrate on the movements of a totally elastic ball.

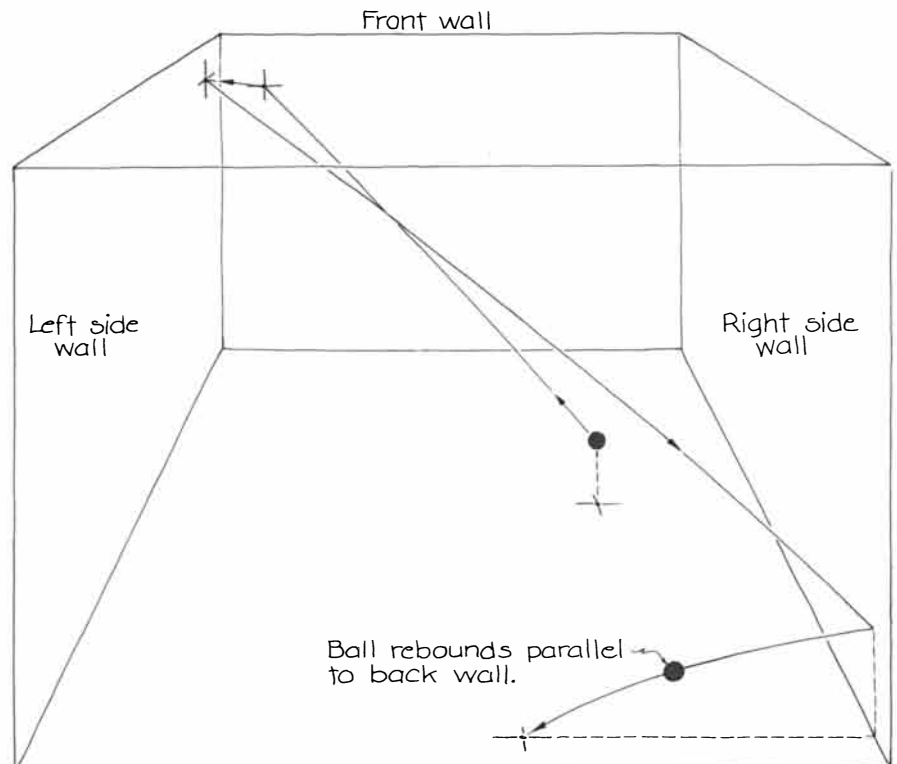
The collision of the ball with the floor changes the perpendicular velocity in a simple way: it reverses the direction but leaves the magnitude and the associated kinetic energy unaltered. The parallel velocity and the spin are altered in more complicated ways. Still, the total kinetic energy is unchanged. An elastic collision might decrease the spin, but the parallel velocity would then be increased just enough to keep the total ki-

netic energy constant. This requirement of conserving the total kinetic energy is a strong tool for predicting the rebound.

Another important point is that the total angular momentum is conserved. One contribution to the angular momentum comes from the spin. This contribution is equal to the rate of spin multiplied by the ball's moment of inertia. The spin angular momentum is considered to be negative if the spin is clockwise and positive if it is counterclockwise. The moment of inertia depends on the mass of the ball and the way the mass is distributed. For a solid ball of uniform density the moment of inertia is two-fifths of the product of the mass and the square of the radius.

The other part of the angular momentum depends on how fast the ball is moving parallel to the floor at the instant it touches the floor. This contribution to the angular momentum is equal to the product of the ball's mass, the parallel velocity and the radius. If the parallel velocity is toward the right, the contribution is negative; toward the left it is positive. The collision may change the two contributions to angular momentum in both magnitude and sign, but the total angular momentum remains. In sum, regardless of how the ball is thrown to the floor or how it spins, the total kinetic energy and the total angular momentum must remain constant in an ideally elastic collision.

The easiest demonstration is to drop the ball to the floor. If it has no spin initially, it must bounce back to your



The troublesome Z shot in racquetball

Introducing a sophisticated

Behind every Smart Desk is a very smart decision-maker. Especially when the desk is equipped with a new IBM 3270 Personal Computer/GX.

This 3270 PC lets you create charts, graphs, 3-D technical diagrams and other high resolution graphics right in your office or lab. You can even view engineering designs created on other IBM graphics systems.

The graphics are created by downloading data from your company's main-frame computer or by running stand-alone. So, in either case, the host is free to go its way while you go yours. The results: faster response time and more cost-effective performance.

The 3270 PC/GX's 19-inch screen displays up to 16 colors. An optional "mouse" makes for fast, easy creation of graphics. An optional writing tablet lets your sketches appear right on the screen. There are even features that let you enlarge important details.

And, because this new computer is an IBM 3270 PC, you get a lot more than just pretty pictures.

It can display up to four different host sessions concurrently and run a full range of IBM Personal Computer applications. You also get two electronic notepads on screen for keeping track of important information.

IBM also has available a 14-inch model of the 3270 PC with advanced color graphics: the 3270 PC/G. Put yourself behind a 3270 PC/G or GX, and you'll

get the picture of how smart a Smart Desk can be.

To receive literature, call 1 800 IBM-2468, Ext. 428, or send in this coupon.

IBM 9-84
DRM, Dept. 2Y3/428
400 Parson's Pond Dr.
Franklin Lakes, NJ 07417

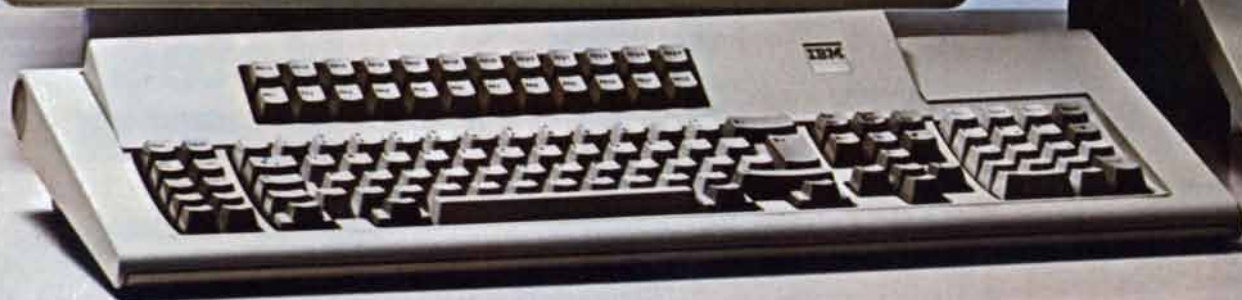
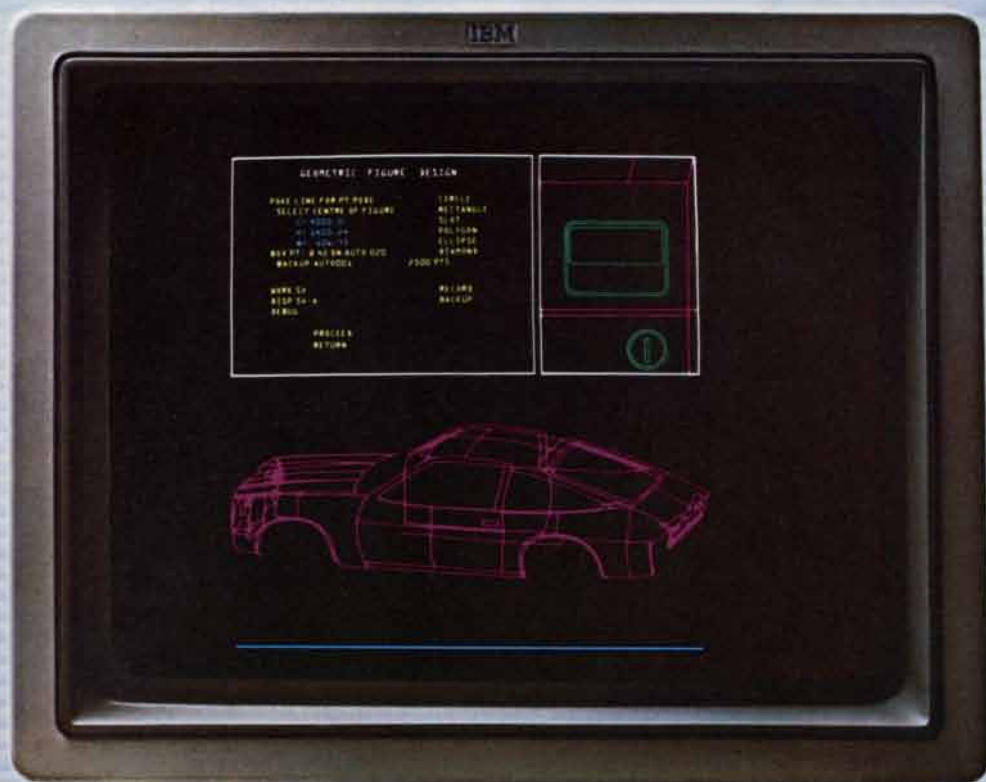
Please send me more information on the new IBM PC/G and GX.
 Please have an IBM representative contact me.

Name _____
Company _____
Title _____
Phone _____
Address _____
City _____ State _____ Zip _____



The Smart Desk from IBM

simple approach to graphics.





YELLOW IS A SIGN OF IMPORTANCE.

That's why people with important business messages use our Post-it[™] Notes adhesive note pads. Bright notes that stick virtually anywhere. To make sure your messages get noticed. Call 1-800-328-1684 for a free sample. Then get more Post-it Notes from a nearby stationer or retail store. And start getting the recognition you deserve.

Commercial Office Supply Division/3M



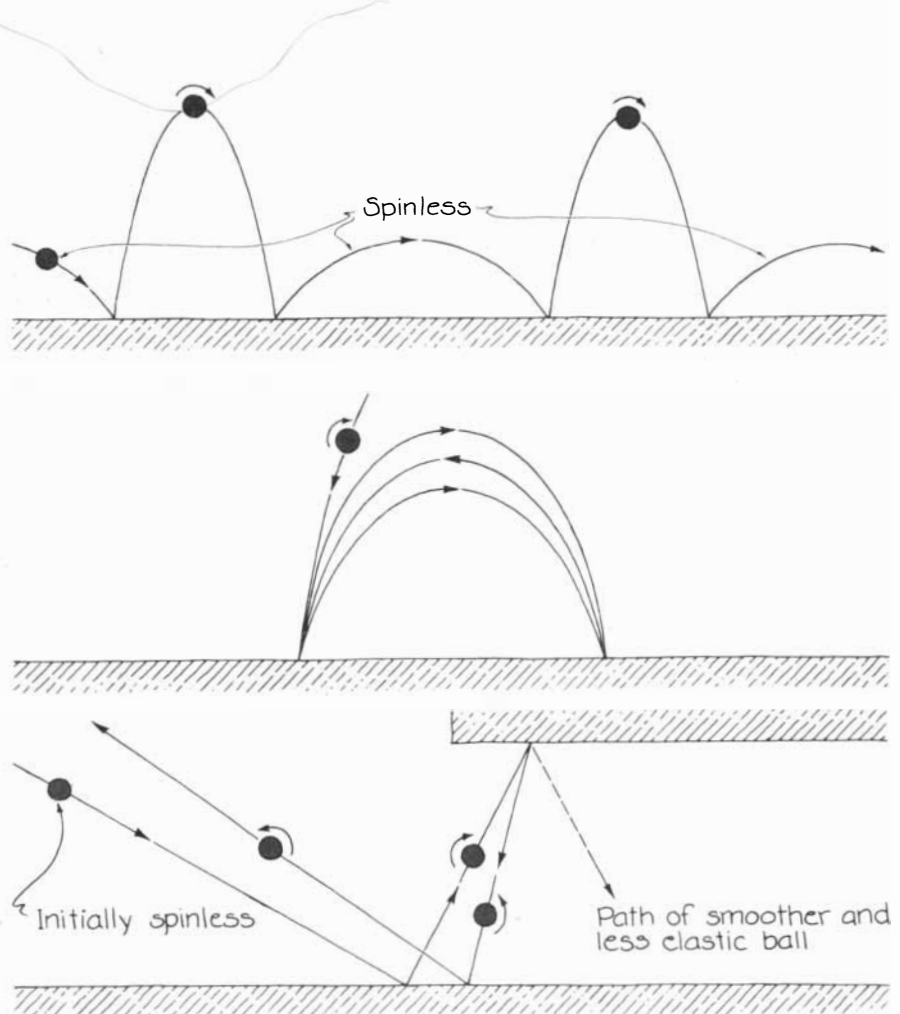
hand without spin because of the conservation rules. The only kinetic energy it has is associated with its perpendicular velocity. Since that velocity is only reversed by the collision, without any change in magnitude, the kinetic energy is unchanged. None of it can be transferred to the spin or to parallel velocity, and so the ball must travel straight upward. This result also satisfies the requirement that angular momentum be conserved. Before the collision and after it the ball's angular momentum is zero.

Suppose you put a clockwise spin on the ball. The collision directs the ball onto a new path. At the collision with the floor the spin creates a friction force toward the right, reversing the direction of spin. Because of the friction force, the ball also acquires a parallel velocity, so that it bounces to the right. The energy for the parallel velocity is taken from the energy of the initial spin.

Energy is also transferred when the ball is thrown to the floor at an angle and without spin. I had expected the path after such a bounce to be just as steep as the initial path, but it is steeper because the collision reduces the parallel velocity, converting some of its kinetic energy into spin energy. In terms of angular momentum the collision reduces the amount associated with the parallel velocity and increases (from zero) the amount associated with the spin. The total kinetic energy and the total angular momentum are conserved.

The steepness of the path after a collision depends on the initial steepness and the spin. When the initial spin is negative (clockwise), the final steepness is less than it is when the ball is thrown down without spin. A strong spin directs the ball along a low path over the floor. When the initial spin is positive (counterclockwise), the ball may bounce forward in a steep path, upward perpendicular to the floor or even backward, depending on the strength of the initial spin. The bounce is straight up if the ball initially has just the right amount of positive spin. (The product of the spin and the radius of the ball must be equal to three-fourths of the ball's initial parallel velocity.) With more counterclockwise spin the ball rebounds to the left. If the spin is less than the threshold amount, equal to zero or negative (clockwise), the rebound is to the right.

The steepness of the rebound can be understood in terms of the friction where the ball touches the floor. The friction force is opposite to the direction in which the surface of the ball is moving. At the moment of contact the surface motion has two sources: parallel velocity and spin. The friction opposes the sum of these two motions. For example, if the ball is thrown down at an angle and without spin, the surface touching the floor is moving to the right. The friction force acting on the surface



The odd bounces of a rough, elastic ball

is toward the left, which reduces the parallel velocity. The ball bounces toward the right with less rightward velocity than it had before the collision. Since the amount of perpendicular velocity is unaltered by the friction, the ball bounces in a path steeper than the one it followed in approaching the floor.

I also considered events in which the ball makes several bounces on the floor. Suppose the ball is thrown to the right without spin. The first bounce reverses the perpendicular velocity (so that the ball goes upward), decreases the parallel velocity and imparts a clockwise spin. The ball rises to its maximum height and falls back to the floor. The surprising feature is that this second bounce restores the initial spin (which was zero) and parallel velocity. The result is the same regardless of the initial values of spin and parallel velocity. If the ball continues to bounce along the floor, its initial values of spin and parallel velocity are restored after every even number of bounces.

The phenomenon was readily apparent in the action of the elastic toy ball. I painted the equator of the ball so that I could monitor the spin. When I threw the ball to the floor with no initial spin, the first bounce was high and short, so

that the ball did not move very far horizontally before the next bounce. The spin was clockwise. The second bounce was low and long. The ball had essentially no spin. Thereafter the ball repeated the pattern of a high, short bounce followed by a low, long one. Since the ball was not totally elastic, each bounce was less energetic than the preceding one. A perfectly elastic ball would periodically resume its initial spin of zero and its initial parallel velocity.

The interactions of spin and parallel velocity account for the strange actions of a ball thrown to the floor so that it strikes the underside of a table. If the ball is initially without spin, it bounces from the floor on a steep path with a rapid clockwise spin; when it hits the table, it rebounds to the left with a counterclockwise spin. The second bounce from the floor is also to the left with a counterclockwise spin. The perpendicular velocity has been reversed three times but is unchanged in amount. The parallel velocity is now toward the left and is almost unchanged in amount. Hence the ball almost returns to the launch site.

Suppose the ball were smoother and less elastic. The first bounce would result in a weak spin and the second (from

the underside of the table) would not be to the left. The ball would continue to travel to the right until it exhausted its kinetic energy.

I next turned my attention to an ideally elastic, hollow racquetball. Such a ball should perform all the tricks of a solid ball, although the spin values differ because the hollow ball has a different moment of inertia. If the ball is thrown at an angle to the floor (toward the right), it will hop straight up provided the spin is counterclockwise and the product of the spin and the ball's radius is equal to one-fourth of the parallel velocity rather than three-fourths.

In racquetball the serve comes off the front wall of the court. The ball rebounds to the opponent either directly or by bouncing from the side walls. The opponent must return the ball to the front wall before it bounces twice on the floor. Except on the serve, the ball can also be bounced from the back wall and the ceiling. I shall consider the shots that are allowed after the serve.

A player can impart spin to the ball with the racquet in only two ways: by stroking forward and over the top of the ball (achieving topspin) or forward and along the bottom of the ball (achieving backspin). The illustration at the left on page 222 depicts the spins from

a view on the right side of the court.

Consider a ball hit hard and low toward the front wall with topspin. The collision is similar to one I described for a solid ball. The topspin (clockwise in the illustration) creates an upward friction force that directs the ball upward and reverses the spin. When the ball returns to the floor, the counterclockwise spin forces a low bounce toward the rear of the court. The potential advantage of such a shot is that your opponent may not expect the high rebound from the front wall or the low hop from the floor.

If you hit the ball hard and low toward the front wall with backspin, which is counterclockwise, it bounces toward the floor with a clockwise spin. It hits the floor close to the front wall and rebounds steeply upward. The potential advantage of this shot is that your opponent may not be able to reach the ball before it bounces from the floor a second time.

Usually my stroke gives the ball little or no spin, but it ends up spinning as soon as it bounces from a wall or from the ceiling. Consider a ceiling shot, which I often make to change the pace of the game. My opponent must adjust not only to the new path but also to strange hops off the floor. Suppose I make the ball bounce from the front

wall to the ceiling. It leaves the ceiling with a clockwise spin. When it hits the floor, its parallel velocity is sharply reduced, making it bounce almost straight up. My opponent, who is expecting a rebound path resembling the path of the approach to the floor, waits too far back in the court.

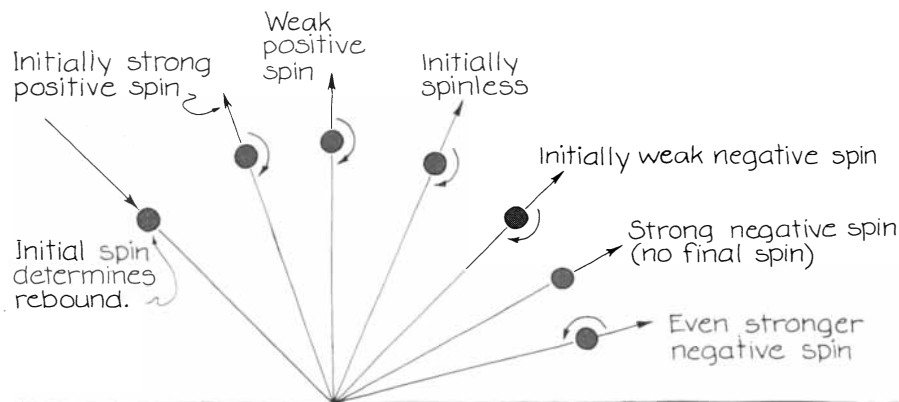
If I make the ball bounce from the ceiling to the front wall, it approaches the floor with a counterclockwise spin. The collision with the floor increases the parallel velocity, sending the ball into a low hop. Again my opponent misjudges the rebound path and misses the ball. Both ceiling shots are better if I start them from about midcourt. Then the spin as the ball approaches the floor is strong and the strange hop is enhanced.

Suppose the ball is bounced off the front wall so that it moves toward the left side of the court. If you take an overhead view and ignore the curvature of the path due to gravity, the arrangement is similar to the one in which a solid ball is thrown at an angle to the floor. The collision reverses the perpendicular velocity (in this case the velocity perpendicular to the front wall), decreases the parallel velocity (the velocity toward the left side wall) and imparts a clockwise spin. In the overhead view the final path is steeper with respect to the front wall than the initial path because of the reduction in the parallel velocity. An opponent can quickly learn how to deal with this type of rebound in racquetball.

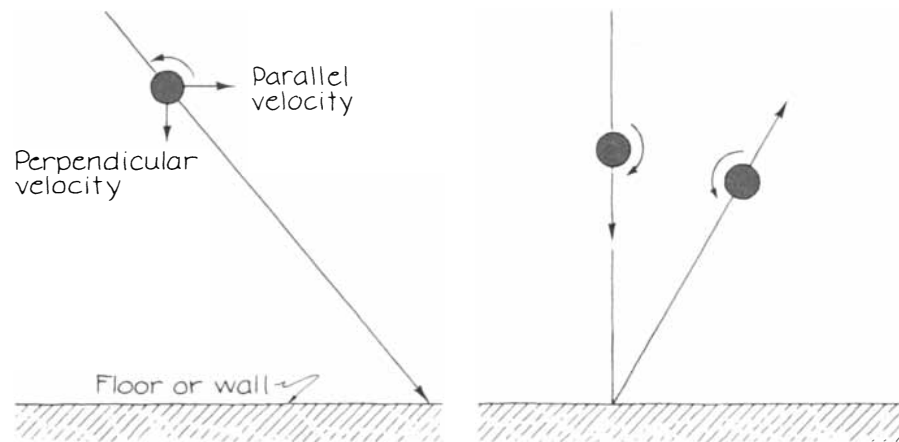
A more difficult shot to anticipate is one that bounces from two walls. Consider an overhead view of a shot in which the ball bounces from the front wall and then from the left side wall. The first bounce gives the ball a clockwise spin and a velocity directed toward the rear wall. Can you make the ball rebound from the side wall in any direction you choose or is the final angle of rebound fixed? Can the final spin be zero or any value of clockwise or counterclockwise rotation? To answer these questions I employed some mathematics published independently by Richard L. Garwin of Columbia University and George L. Strobel of the University of Georgia.

Assume the ball is launched toward the front wall with no spin and has a small initial perpendicular velocity. You can make such a shot if you are near the front of the right side wall. Then an ideally elastic racquetball rebounds from the left side wall at an angle of about 12 degrees. If you are closer to the center of the court, the initial perpendicular velocity is larger and the angle of the rebound from the left side wall is smaller; the ball travels along the wall to the rear of the court.

You can use this arrangement to advantage. Suppose your opponent is near the middle of the right wall. By bouncing the ball off the front wall and into the



How the bounce depends on the initial spin



The two components of a ball's velocity

How spin deflects a bounce

A comprehensive software approach to symbolic mathematical computations.

Computer algebra, whereby a computer manipulates symbolic mathematical expressions, has always been something people have thought about, but it was never something people thought would work for them.

Now there is a newly available software tool, one that can do not only rudimentary algebraic problems but also extremely complex analyses.

It has been under continual development since 1969 at M.I.T., the result

An engineer working for a major aerospace company needed to evaluate the following integral dealing with turbulence and boundary layers:

$$\int (k \log(x) - 2x^3 + 3x^2 + b)^4 dx$$

He had worked on this problem for more than three weeks with pencil and paper, always arriving at a different result. He was never sure which of the many results he had come upon was correct.

copter blade motion studies, solid state physics, maximum likelihood estimation, and atomic scattering cross section analysis. The range of use is expanding every day.

MACSYMA™ currently operates on Symbolics' 3600, DEC's** VAX** series, TOPS 20** and Honeywell Multics** computer systems.

So if you're still solving problems the old fashioned way with pencil and paper, or trying to approximate results

Can your computer software evaluate this integral in closed form? Can you even find it in a table of integrals?

$$\int \operatorname{erf}(ax) \operatorname{erf}(bx) dx = - \frac{\sqrt{a^2 + b^2} \operatorname{erf}(x \sqrt{a^2 + b^2})}{ab \sqrt{\pi}} + x \operatorname{erf}(ax) \operatorname{erf}(bx) + \frac{e^{-a^2 x^2} \operatorname{erf}(bx)}{a \sqrt{\pi}} + \frac{e^{-b^2 x^2} \operatorname{erf}(ax)}{b \sqrt{\pi}}$$

$$\text{where } \operatorname{erf}(x) \equiv \frac{2}{\sqrt{\pi}} \int_0^x e^{-u^2} du$$

MACSYMA solved this problem in seconds.

of research in the area of artificial intelligence.

The program is called MACSYMA.™

It's the most comprehensive approach to symbolic mathematics. The outgrowth of more than 100 man-years of development, it contains more than 300,000 lines of code and is supported by more than 500 pages of carefully maintained documentation.

More important than any of that, though, it's available today.

From Symbolics.

Symbolics presents MACSYMA™ for computer algebra.*

In only a few minutes, you can use MACSYMA™ to do meaningful work.

As an interactive tool, it can help you explore problems in basic or advanced mathematics, problems that you can't begin to approach using pencil and paper or any numerical software.

For all its sophistication, however, a novice with no prior programming experience can use MACSYMA.™ The user doesn't have to learn a new language either. As a matter of fact, users can interact with MACSYMA™ in an almost conversational manner.

An example: Three weeks long-hand vs. 10 seconds processing time.

In less than 10 seconds after entering the problem in the computer, MACSYMA™ gave him the correct answer, not in numerical terms, but in symbolic terms that gave him real insight into the physical nature of the problem.

Applications for the real world. Potential for new worlds.

MACSYMA™ has hundreds of practical, realworld, immediate applications. It can simplify, factor or expand expressions, solve equations analytically or numerically, differentiate, compute definite and indefinite integrals, expand functions in Taylor or Laurent series, compute Laplace transforms, manipulate matrices and tensors, plot functions in 2 and 3 dimensions, generate FORTRAN output from MACSYMA™ expressions, provide most of the standard numerical techniques, and much more.

Right now, more than 1,000 scientists, engineers and mathematicians throughout the world use MACSYMA™ in a range of applications as diverse as acoustics, plasma physics, antenna theory, VLSI circuit design, control theory, numerical analysis, fluid mechanics, genetic studies, ship hull design, ballistic missile defense systems design, underwater shock wave analysis, heli-

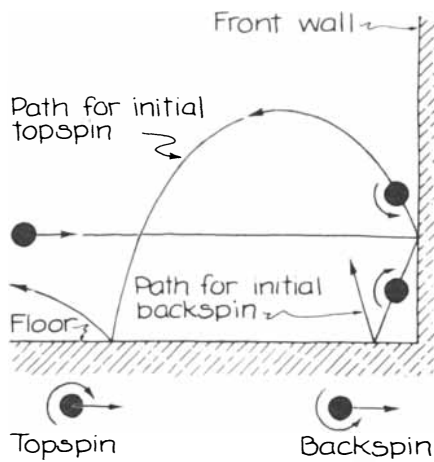
with numerical systems, make it your business to find out about MACSYMA.™

We'll send you a complete literature kit including an article published in the December 1981 issue of Scientific American, the solution to the integral dealing with turbulence and boundary layers, and a full capabilities brochure. Simply clip the coupon below.

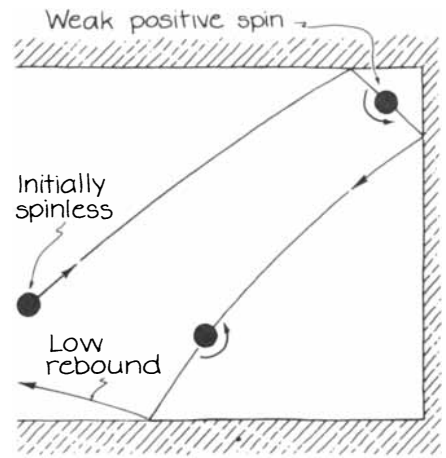
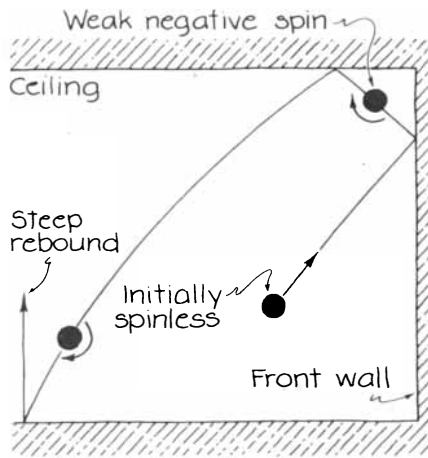
Symbolics Inc., 11 Cambridge Center, Cambridge, MA 02142		SA9
Name _____		
Title _____		
Company _____		
Address _____		
City _____	State _____	Zip _____
Telephone _____	Ext _____	
MACSYMA™ is available to colleges and universities at special rates.		

Macsyma™
from *symbolics*

*As titled in the December 1981 issue of Scientific American.
**DEC, VAX and Tops 20 are registered trademarks of Digital Equipment Corporation.
Multics is a registered trademark of Honeywell Inc.
MACSYMA is a trademark of Symbolics, Inc.



The energetics of topspin and backspin



Schemes for making use of the ceiling

left side wall so that it travels along the side wall to the back of the court, you can make it almost impossible for him to return the shot. Even if he is not far from the final path of the ball, the rebound off the side wall might at least prove confusing.

When I tested my calculations with a real racquetball, I found an approximate agreement. The steepest angle of rebound from the side wall was larger than the 12 degrees I had predicted. As I increased the initial perpendicular velocity by moving from the right wall toward center court, the angle of rebound decreased until the ball almost hugged the left wall on its way to the rear of the court.

The discrepancy between the actual and the predicted rebound off the side wall arises from the inelastic collisions of a real racquetball. If the ball hits a wall squarely, it compresses uniformly, storing its energy as elastic potential energy. Only part of the energy is reconverted into kinetic energy as the ball pushes off from the wall, again taking the shape of a sphere. A racquetball might bounce back with 60 percent of its energy in such a collision. The perpendicular velocity would then be about 80 percent of the initial value. (The change in velocity is proportional to the square root of the change in energy.)

A glancing collision is more difficult to interpret because the compression of the ball is not uniform and depends on the angle of the collision. The loss of kinetic energy and angular momentum reduces both the spin and the parallel velocity. (When the ball skims along the wall or the floor in an extreme glancing shot, you can hear the energy loss as a high-pitched squeal as the ball skips over the surface.) In my calculations I chose to reduce the spin and the parallel velocity after a collision by .4. With these reductions I found closer agreement between my predictions and the actual rebounds.

I was also able to explain why a real

racquetball does not return to me when I throw it under a table. The reductions in energy and angular momentum in the bounces from the floor and the underside of the table trap the ball into bouncing almost vertically until it exhausts its kinetic energy.

Is there a way to hit the ball to the front wall so that it rebounds from a side wall parallel to the front wall? With such a shot you could win every game because your opponent could not possibly get to the ball in time. As it turns out such a shot is impossible. A rebound from a side wall is always toward the rear of the court.

Can a rebounding ball have any direction of spin or even no spin? Yes, because its final spin depends on the initial ratio of perpendicular and parallel velocities. For a perfectly elastic racquetball a spin of zero results when the ratio is 1 to 5. A smaller ratio yields a clockwise spin (from an overhead view), a larger ratio a counterclockwise spin.

The Z shot is a three-wall rebound that is marvelous to watch. When it was first introduced in the early 1970's, it confounded even the most experienced players. The ball is hit to the top left side of the front wall, bounces to the left side wall, crosses the court to the rear of the right side wall and then rebounds parallel to the back wall. An opponent will need experience to anticipate the final rebound, but even then the ball will be difficult to return to the front wall. If I hit the Z shot less than perfectly, the ball might still be difficult to return if it hits the floor and then the back wall. My opponent must catch it near the back wall before the ball makes its second bounce on the floor.

Initially I thought a perfect Z shot was impossible. I doubted that the final rebound could be made to move parallel to the back wall. Armed with my mathematics I set out to follow the bounces.

I immediately met a problem. If the ball is assumed to be perfectly elastic, it rebounds from the left side wall at

such a small angle that it hits the back wall instead of the right side wall. I factored an extra-long court into my calculation. I also ignored the curve resulting from gravity and made the calculation as though the ball remained in a plane parallel to the floor.

To launch the Z shot a player stands near the right wall at about midcourt. The ball is hit to the top left side of the front wall about three feet from the corner and three feet from the ceiling. Since such a shot makes the ball leave the left side wall with a clockwise spin, its collision with the right side wall creates a friction force toward the front wall.

Consider the velocity and the spin of the ball just before and just after the collision with the right side wall. The perpendicular velocity is reversed, directing the ball toward the opposite side wall. What happens to the spin and the parallel velocity? The collision is similar to one I considered earlier. The friction during the collision opposes both the spin and the parallel velocity, reducing the parallel velocity and reversing the spin. Under the proper conditions the parallel velocity can be reduced to zero, so that the ball's path is perpendicular to the side wall. This is how a perfectly executed Z shot makes the ball travel parallel to the back wall.

When my calculations include the loss of energy with each collision, my predictions are closer to the actual path of a Z shot in a court of the proper dimensions. The possibility of a final rebound parallel to the back wall is still present. My calculations are flawed, however, since the actual path has three dimensions. My assumption of a flat trajectory simplifies the calculations because the axis about which the ball spins is always kept parallel to the wall. In the actual flight of the ball the spin axis is often at an angle with respect to the side walls.

The around-the-walls shot also hits three walls. The ball is bounced from the right side wall to the front wall and then off the left side wall. The shot is

BREAKTHROUGH: COMPUTER GRAPHICS THAT CREATE MODEL PATIENTS FOR SURGEONS.

We're using computers to give reconstructive surgeons a startling new perspective on their work—three-dimensional images that depict the patient's face and skull from any angle, inside or out.

These 3D images show key relationships between bone and soft tissue. They help surgeons predetermine precisely how bone and skin grafts should be placed to achieve the desired results. They can be used to create physical models that let a surgeon perform a trial procedure or envision the results of his work before surgery begins.

Our breakthrough in computer modeling helps surgeons correct birth defects and undo the damage of disfiguring accidents.

We're creating breakthroughs not only in health care and information handling but also in communications and equipment leasing.

We're McDonnell Douglas.



MCDONNELL DOUGLAS

How our \$2,000 CAD from burning up

Until now, high-quality, professional computer-aided design could only be done on expensive, large-scale mini or mainframe computers.

But now there's AutoCAD.[™] A full-function graphics program that turns your low-cost personal computer into a full-fledged CAD workstation. So that designing an AUTO-MATIC 50-350 GPM-class fire nozzle has never been easier. Or more cost-effective.

**90% of mainframe CAD
at 5% of the price.**

AutoCAD is designed especially to work on today's most popular microcomputers to fully automate the drawing process. For just about any

drawing job you can think of.

With its extensive, push-button editing facilities, AutoCAD lets you DRAW an object (or any part of one); ROTATE or SCALE it; DRAG it; ZOOM in and out on it; STORE it away in your own parts library and call it up whenever you need it; FILL an area; use an unlimited number of LAYERS; even automatically DIMENSION distances and angles in any part of your drawing.

And, at only \$2,000 for the complete AutoCAD program, you can have all this power in a fully-configured, top-of-the-line desktop workstation for less than \$15,000.

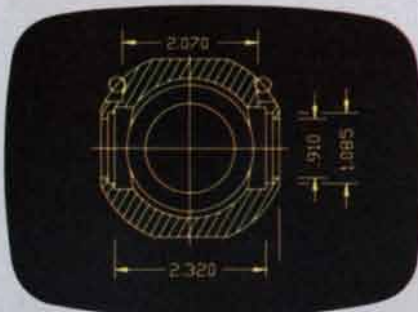
**You'll learn it
in two days.**

The beauty of AutoCAD is that you don't have to know a thing about computers. Even if you've never touched a keyboard, you'll pick it up in a matter of days, and feel comfortable inside a week.

Sound too good to be true? A simple one-touch command structure always points you

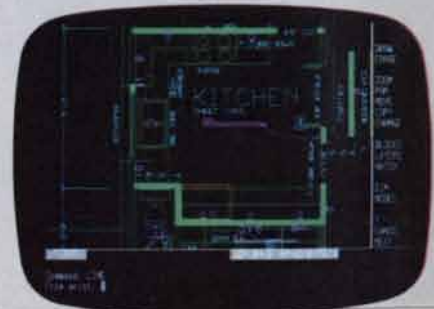
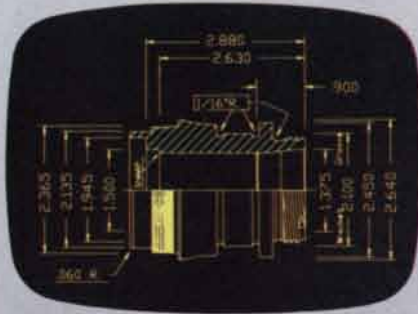


in the right direction. A HELP command is always available to keep you on track. And AutoCAD even lets you choose



*Zoom in on a part
to focus on the details*

*AutoCAD automatically
dimensions distances and angles*



*Would the Pantry be better where the
stove is? Use the MOVE command*

from among a wide variety of input devices to suit your particular needs: including pointing devices, mice and easy-to-use electronic drawing

software can keep you your design budget.

Fire nozzle and nozzle designs provided
courtesy of TASK FORCE TIPS, Inc.

PC boards, skyscrapers, and everything in between.

No matter
what type of
drafting your

work involves—from electro-
mechanical circuit layout to
structural schematics to archi-
tectural design—AutoCAD
can give you a whole new
perspective on the drawing
process.

And on saving money.

Because whether you're in a
two-person shop or a two-
billion dollar company, you'll
find that the AutoCAD system
pays for itself in just a few
short months.

Beyond that, your Auto-
CAD software will run on
newer, and more powerful com-
puters when they become
available—so that you can be
sure your investment in creat-
ing drawings and training staff

will never go up in smoke.

Fire off a letter.

There are currently over
7000 AutoCAD users world-
wide. If you'd like more infor-
mation, write or call your
nearest AutoCAD dealer. He or
she can provide you with all
the details, including literature,
specifications, and hands-on
demonstrations.

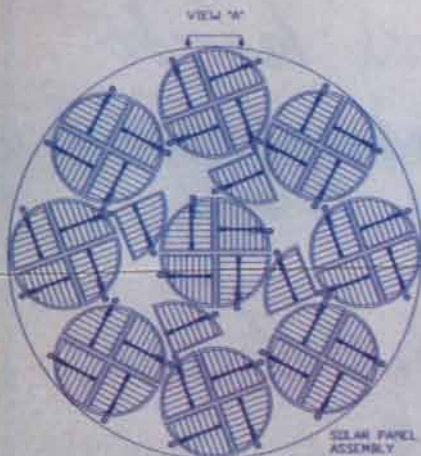
AutoCAD. For designers,
it's why the personal computer
was invented.



AUTOCAD™

AUTODESK, INC.
2658 BRIDGEWAY
SAUSALITO, CA 94965
(415) 331-0356
TELEX 756521 AUTOCAD UD

tablets. (As well
as printers, plotters,
and microcomputers
from a growing number
of manufacturers.)



*Circles, arrays and area
fills are easy with AutoCAD*

*"If you are wondering
why Berlin is the*

CENTER OF CAD/CAM

*in Germany,
let us enlighten you."*

Economic Development

BERLIN

*11% of all people engaged in R+D in the Federal Republic of Germany
work in the City of Berlin. As a center for the conversion
of scientific knowledge into economic practice Berlin has the leadership
position, e. g. within the field of CAD/CAM, laser and software.
By the way: The City of Berlin is the only location in the European Community,
where you get a bonus up to 14% for Berlin products and services.
Ask for the free CAD/CAM leaflet or mark the code number.*

Wolfe J. Frankl, Berlin Economic Development Corp., 767 3rd Ave., New York, N.Y.
10017-2079, Tel. (212) 980 1545.

Trivia Question of the Month

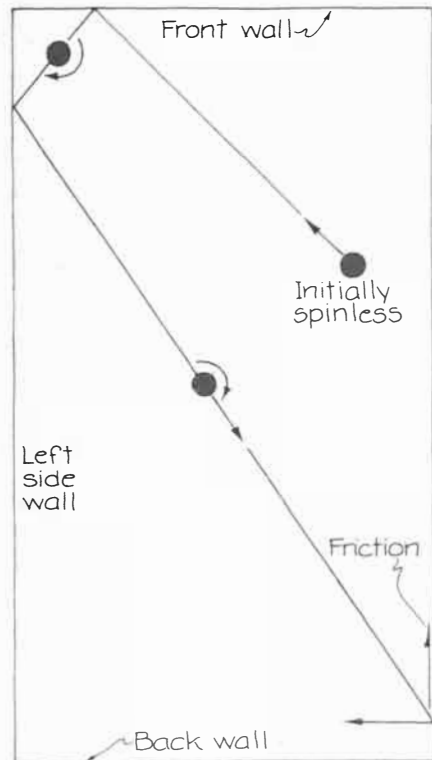
*Q: Which software package
in 1984 will be in as high
demand as Trivia Pursuit and
the Cabbage Patch doll?*

*For the answer call B.M.I at:
(619) 434-6165*

designed to confuse an opponent, but if the ball ends up at midcourt, he may have an easy chance of returning it to the front wall. I wondered if there was any way I could set up the around-the-walls shot to make the ball rebound from the left side wall parallel to the front wall. Expecting the ball to come to the rear of the court, my opponent would surely be caught off guard by this strange rebound.

I tried the shot in many ways without success. I wondered if the problem was my lack of playing skill, and so I turned again to mathematics. My calculations showed that such a rebound is possible if the ball begins with much energy and makes a small angle with the right side wall. If I had made the calculations earlier, I could have saved myself many futile swings of the racquet.

Many more shots can be studied with either a solid ball or a hollow racquetball. Perhaps there are some clever shots that even the professional racquetball players have yet to discover. You may be interested in studying how a ball loses energy in a glancing collision with a wall. You may also be interested in following the flight of a ball in three dimensions, so that the spin axis is no longer parallel to the walls. For this purpose a computer simulation of racquetball would be helpful. Be careful if you experiment with a solid, highly elastic ball in a racquetball court. I tried it just once. The ball moved and rebounded so fast that all I could do was get out of the way.



The Z shot as it would be seen from overhead

SAVE BIG ON COMPUTER PRODUCTS

MODEMS	
Smartmodem 300	200
Smartmodem 1200	489
Smartmodem 1200B	425
Micromodem II E	240
Smartcom II Software (for IBM PC)	90
Cables to Hayes Modems	CALL
NOVATION	
SmartCat Plus (NEW)	CALL
J. Cat 300B direct	99
103 SmartCat 300B Smart	159
103/212 Smart 300/1200 B	378
AppleCat II 300 baud for Apple	200
212 AppleCat 300/1200B for Apple	390
Access 1-2-3 1200B for IBM	378

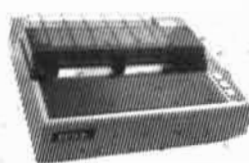
MONITORS	
SUPER SPECIAL!!	
COMREX	
12" Green H. Par.	75
AMDEK	
V300G 12" green	122
V300A 12" amber	145
V310A 12" amber (for IBM)	160
Color I+ 13" composite	280
Color II+ 13" RGB	425
Color IV 13" RGB analog	740
Monitor Cables	
CB 5990 Apple II to Monitor	7
CB 5991 IBM to RGB	19
CB 5992 for TI-99/4A or Commodore	15

QUADRAM	
MICROFAZER	
GRMP-8 Par/Par	135
GRMSP-8 Ser/Par	145
GRNMSB-8 Ser/Ser	145
GRMPS-8 Par/Ser	145
QUABOARD (for IBM PC)	
QR 5310 (no mem. installed)	210
QR 5364 84K (mem. installed)	270
QR 4064 84K (mem. installed)	270
QR 8201 Quicolor-1	200
QR 8202 Quicolor-2 (upgrade kit)	200
QUADLINK	
QR 3000 for IBM	475
QR 3010 for Compaq	475
QR 3020 for Columbia	475

AST	
Six Pak Plus II	270
Mega Plus II	270
1/0 Plus II	112

SAVE 31%-43%
OFF MFR. SUGG. RETAIL PRICES ON

PRINTERS
EPSON • OKIDATA • DIABLO
SCM • DELTA • GEMINI • TTX
RADIX • COMPUTE-MATE
MANNESMANN TALLY



CALL FOR PRICES

webdash
MAXELL
Dyan

DISKETTES	
10 - 5 1/4" Floppy Diskettes	
SS/SD	SS/DD DS/DD
Webdash \$13.00	\$16.00 \$17.50
Maxell NA	20.00 26.00
Dyan NA	23.00 30.00
10 - 8" floppy diskettes	
Dyan NA	32.00 37.00

CALL FOR QUANTITY PRICING ON
10 OR MORE BOXES OF DISKETTES

**HUGE DISCOUNTS all
RIBBONS • DUST COVERS
PAPER • POST CARDS • LABELS**
for almost every make and model

CALL TOLL FREE 800-621-1269 EXCEPT Illinois, Alaska, Hawaii

Corp. Accts. Invited. Min. Ord. \$15.00. Mastercard or Visa by mail or phone. Mail Cashier's Check. Min. Ord. Pers. Check (2 wks. to cr.) Add \$4.00 1st class. AK, HI, P.R., Canada add \$10.00 first item; \$1.00 ea. add. ship. & hand. Shipments to IL address add 7% tax. Prices sub. to change. WRITE for free catalog. RETURN POLICY: Defective Only: Most products replaced within 30 days of purchase with identical merchandise only. Computer and large peripherals replaced only when defective on arrival (within 3 work days of delivery). Other problems covered by mfr. warranty. ALL ELEK-TEK MERCHANDISE IS BRAND NEW, FIRST QUALITY AND COMPLETE.

ELEK-TEK, inc.

6557 N. Lincoln Ave., Chicago, IL 60645
(312) 631 7800 (312) 677 7660

HEWLETT-PACKARD

CALCULATORS	
HP-11C	56 HP-15C 90
HP-12C	90 HP-16C 90
HP-41C	145 HP-41CX 245
HP-41CV	168 HP-97 560

all software & accessories too

PORTABLE COMPUTERS

HP-71B	399 HP-75D 799
--------	----------------

series 70 software & peripherals discounted too

HP-2225B ThinkJet Printer (HPIL) 375



SHARP	
CALCULATORS	
EL 5100	43
EL 5500 T	70
EL 512 T	28

SHARP	
HAND HELD COMPUTERS	
pc 1250A	80
pc 1260	CALL
pc 1261	CALL
pc 1500A	160

PRINTERS DISCOUNTED TOO

CABLES — INTERFACES
accessories for Computer Printers

GRAPPLER PLUS	105
16K BUFFERED GRAPPLER	165
Apple Dumping GX	65
Cardco G	85
CB5609 10 ft. Par. Cable for IBM	25
CB5622 10 ft. 36x36 Parallel	32
CB5629 10 ft. 25x25 RS-232	25
CB5618 6 ft. TI-99/4A parallel cable	25
CB5620 6 ft. par. TRS80 mod. I-III-IV-22	CALL

Speak German like a Diplomat!

What sort of people need to learn a foreign language as quickly and effectively as possible? *Foreign service personnel*, that's who.

Now you can learn to speak German with the same materials used by the U.S. State Department—the Foreign Service Institute's *Programmed Introduction to German*.

The FSI spent thousands of dollars and many years developing these materials for use by members of the United States diplomatic corps. Today people in all walks of life who need to learn to speak a foreign language are turning to this outstanding audio program.

The FSI's German Course is by far the most effective way to learn German at your own convenience and pace. It consists of a series of cassettes, recorded by native German speakers, and accompanying textbook. You simply follow the spoken and written instructions, listening and learning. By the end of the course you'll find yourself learning and speaking entirely in German!

This course turns your cassette player into a "teaching machine." With its unique "programmed" learning method, you set your own pace—testing yourself, correcting errors, reinforcing accurate responses.

This FSI Programmed Course comes in two volumes, each shipped in a handsome library binder. Order either, or save 10% by ordering both:

□ **Volume I. Programmed Introduction to German**, 10 cassettes (13 hr.); and 647-p. text, \$125.

□ **Volume II. Basic Course Continued**, 8 cassettes, (8 hr.), and 179-p. text, \$110. (Conn. and N.Y. residents add sales tax.)

TO ORDER BY PHONE, PLEASE CALL
TOLL-FREE NUMBER: 1-800-243-1234.

To order by mail, clip this ad and send with your name and address, and a check or money order—or charge to your credit card (VISA, MasterCard, AmEx, Diners) by enclosing card number, expiration date, and your signature.

The Foreign Service Institute's German Course is unconditionally guaranteed. Try it for three weeks. If you're not convinced it's the fastest, easiest, most painless way to learn German, return it and we'll refund every penny you paid. Order today!

116 courses in 39 other languages also available. Write us for free catalog. Our 12th year.

Audio-Forum
Room 603
On-the-Green
Guilford, CT 06437
(203) 453-9794



AUDIO-FORUM®

Or visit our New York sales office. 145 E. 49th St., New York, N.Y. 10017 (212) 753-1783

THE DOMINO CHAIN REACTION SETS

Described in last issue's
"The Amateur Scientist"
by Jearl Walker

Developed by
Professor Lorne A. Whitehead
Department of Physics
University of British Columbia

This set is a startling and memorable demonstration of the chain reaction concept. It serves as an analogy for such diverse phenomena as nuclear explosions, electronic amplification, population growth, and even the political "domino theory." No classroom should be without one! The largest domino in the full set releases about 50 joules of energy when it topples—an amplification factor of 2 billion.

Complete 13 piece set — \$165.00*
Economy 10 piece set — \$ 39.00*

*Plus Postage

Please send all orders to:
Ginsberg Scientific Company
Route 1, Box 104 AB
Macks Creek, MO 65786
(314) 363-5260



On the leading edge
of computer
communication potentials

- Professional Partner in your Project Since 1972
- Research
- Training
- Project Coordination
- Courseware Development
- Consultation

People's Computer
A Non Profit Corp.

2682 Bishop Drive Suite 107
San Ramon, CA 94583
415 833-8604

BIBLIOGRAPHY

Readers interested in further explanation of the subjects covered by the articles in this issue may find the following lists of publications helpful.

COMPUTER RECREATIONS

PRINCIPLES OF NEURODYNAMICS: PERCEPTORS AND THE THEORY OF BRAIN MECHANISMS. Spartan Books, 1962
PERCEPTORS: AN INTRODUCTION TO COMPUTATIONAL GEOMETRY. Marvin Minsky and Seymour Papert. The MIT Press, 1969.

COMPUTER SOFTWARE

PSYCHOLOGY OF INVENTION IN THE MATHEMATICAL FIELD. Jacques S. Hadamard. Dover Publications, Inc., 1945.
UNDERSTANDING MEDIA: THE EXTENSIONS OF MAN. Marshall McLuhan. McGraw-Hill Book Company, 1964.
THE MYTHICAL MAN-MONTH: ESSAYS ON SOFTWARE ENGINEERING. Frederick P. Brooks, Jr. Addison-Wesley Publishing Company, Inc., 1974.
MATHEMATICS: THE LOSS OF CERTAINTY. Morris Kline. Oxford University Press, 1980.
THE FRACTAL GEOMETRY OF NATURE. Benoit B. Mandelbrot. W. H. Freeman and Company, 1982.
THE ELEMENTS OF FRIENDLY SOFTWARE DESIGN. Paul Heckel. Warner Books, 1984.

DATA STRUCTURES AND ALGORITHMS

AN AXIOMATIC BASIS FOR COMPUTER PROGRAMMING. C. A. R. Hoare in *Communications of the ACM*, Vol. 12, No. 10, pages 576-583; October, 1969.
THE HUMBLE PROGRAMMER. Edsger W. Dijkstra in *Communications of the ACM*, Vol. 15, No. 10, pages 859-866; October, 1972.
ALGORITHMS + DATA STRUCTURES = PROGRAMS. Niklaus Wirth. Prentice-Hall Book Company, 1976.

PROGRAMMING LANGUAGES

FORTH: AN INTRODUCTION TO THE FORTH LANGUAGE AND OPERATING SYSTEM FOR BEGINNERS AND PROFESSIONALS. Prentice-Hall, Inc., 1981.
HISTORY OF PROGRAMMING LANGUAGES. Edited by Richard L. Wexelblat. Academic Press, 1981.
THE IMPACT OF ABSTRACTION CONCERNS ON MODERN PROGRAMMING LANGUAGES. Mary Shaw in *Studies in Ada Style*, by Peter Shaw, Andy Hisgen, Jonathan Rosenberg, Mary Shaw and

Mark Sherman. Springer-Verlag, 1981.

PROGRAMMING LANGUAGES: DESIGN & IMPLEMENTATION. Terrence W. Pratt. Prentice-Hall, Inc., 1984.

OPERATING SYSTEMS

CONCURRENT EUCLID, THE UNIX SYSTEM AND TUNIS. R. C. Holt. Addison-Wesley Publishing Company, Inc., 1983.
OPERATING SYSTEM DESIGN: THE XINU APPROACH. Douglas Comer. Prentice-Hall, Inc., 1984.
THE UNIX PROGRAMMING ENVIRONMENT. Brian W. Kernighan and Rob Pike. Prentice-Hall, Inc., 1984.

COMPUTER SOFTWARE FOR WORKING WITH LANGUAGE

UNDERSTANDING NATURAL LANGUAGE. Terry Winograd. Academic Press, 1972.
METAPHORS WE LIVE BY. George Lakoff and Mark Johnson. University of Chicago Press, 1981.
NATURAL LANGUAGE PROCESSING. Harry Tennant. Petrocelli Books, Inc., 1981.
THE MENTAL REPRESENTATION OF GRAMMATICAL RELATIONS. Edited by Joan Bresnan. The MIT Press, 1982.
LANGUAGE AS A COGNITIVE PROCESS. Terry Winograd. Addison-Wesley Publishing Company, Inc., 1983.
TEX Book. Donald E. Knuth. Addison-Wesley Publishing Company, Inc., 1983.

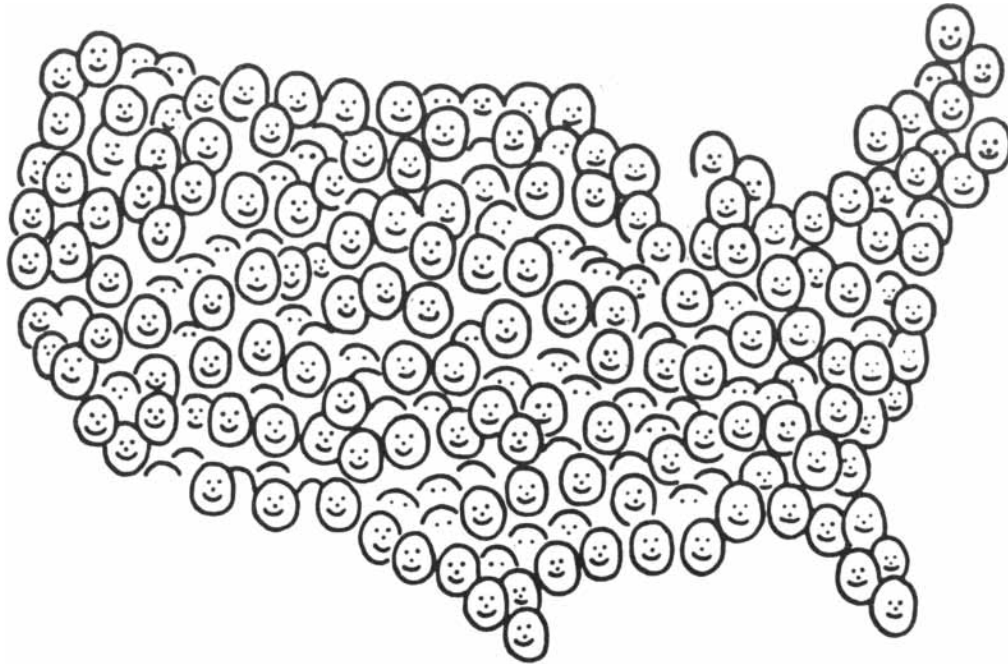
COMPUTER SOFTWARE FOR GRAPHICS

THE COMPUTER IMAGE: APPLICATIONS OF COMPUTER GRAPHICS. D. P. Greenberg and A. Marcus. Addison-Wesley Publishing Company, Inc., 1982.
FUNDAMENTALS OF INTERACTIVE COMPUTER GRAPHICS. James D. Foley and Andries van Dam. Addison-Wesley Publishing Company, Inc., 1982.
COMPUTER IMAGES: STATE OF THE ART. Joseph Deken. Stewart, Tabori & Chang Publishers, Inc., 1983.
PROCEEDINGS OF THE SIGGRAPH '84 CONFERENCE, JULY 23-27, 1984, MINNEAPOLIS, MINNESOTA. Edited by Hank Christiansen in *Computer Graphics*, Vol. 18, No. 3; July, 1984.

COMPUTER SOFTWARE FOR INFORMATION MANAGEMENT

AS WE MAY THINK. Vannevar Bush in *The Atlantic Monthly*, Vol. 176, No. 1,

COMPUTER END-USERS ARE ALL OVER THE UNITED STATES.



SO ARE COMPUTER SHOWCASE EXPOS.

Hundreds of thousands of business owners, corporate managers, educators, and professionals have attended COMPUTER SHOWCASE EXPO in cities across the United States. They're in the market to buy personal computers, small business systems,

packaged software, and related products and services for use in their offices and homes. Join the exhibitors who have already used COMPUTER SHOWCASE EXPO to reach this explosive U.S. market.



COMPUTER SHOWCASE EXPO WE'LL TAKE YOU TO YOUR MARKET.

Atlanta, GA	Detroit, MI	New York, NY
Chicago, IL	Los Angeles, CA	Philadelphia, PA
Cleveland, OH	Miami (S. Florida), FL	Tampa, FL



Conference & Exposition Producers

300 First Avenue, Needham, Massachusetts • (617) 449-6600 • TELEX - 951176 • TWX - 710-325-1888

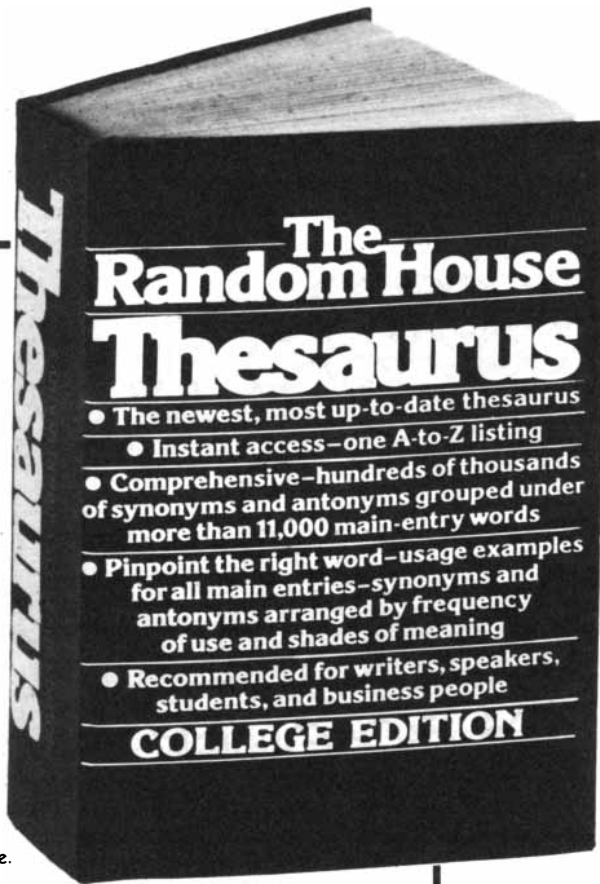
Regional Office: 4700 North State Road 7, Suite 121, Fort Lauderdale, Florida 33319 • (305) 484-6800

New!

The one-step thesaurus.

Open it *once*—there's the word you're after! No time-wasting cross-references. Sample sentences for each entry! Available with *The Random House College Dictionary* in a boxed gift set.

 Now at your bookstore.
RANDOM HOUSE



Explore NEW curves, relations, transformations & problems in the Euclidean plane through.....

STRUCTURAL EQUATION GEOMETRY

by J. Lee Kavanau, UCLA

The author has *"discovered innumerable new worlds within that innocuous-looking....Euclidean plane.....contains a wealth of geometrical reflection & insight."* Prof. Alex. Grothendieck

Nov., 1983, 512 pages, 61 pp. of figs. \$19.95 + \$3p/h
A UNIQUE SOURCEBOOK no geometry instructor can afford to be without

► Introduces Prof. Kavanau's companion treatises on the analysis of general algebraic curves that...◄

"open up fields of seemingly inexhaustible wealth"

Prof. Alexander Grothendieck

"represent tremendous amounts of new information"

Prof. Morris Newman

SYMMETRY, An Analytical Treatment

August, 1980, 656pp., illus., \$29.95 + \$4 p/h

"One of the most original treatments of plane curves to appear in modern times. The author's new and deeper studies...reveal a great number of beautiful & heretofore hidden properties of algebraic plane curves." Prof. Basil Gordon

"Provides sharp new tools for studying the properties of general algebraic curves." Prof. Richard Fowler

"Striking new results on symmetry & classification of curves...Read this book for more in symmetry than meets the eye." Amer. Math. Monthly, 1981

Send SASE for \$3,000 Geometry Competition details.

1983 Co-Awardees: Profs. J.F. Rigby, Cardiff; J.B. Wilker, Toronto; W. Wunderlich, Vienna

Science Software Systems, Inc., 11899 W. Pico Blvd., Los Angeles, Calif., 90064

CURVES & SYMMETRY, vol. 1

Jan., 1982, 448pp., illus., \$21.95 + \$3.50p/h, all 3 books, \$62 + \$7p/h

"Casts much new light on inversion & its generalization, the linear fractional (Moebius) transformation, with promise of increasing their utility by an order of magnitude." Prof. Richard Fowler

"Replete with fascinating, provocative new findings...accompanied by a wealth of beautiful & instructive illustrations." Prof. Basil Gordon

"Extends the idea of inversion into quite a new field." E. H. Lockwood

VISA, M/C, 213-477-8541, OUTSIDE U.S., PREPAID ONLY, + ADDIT'L \$1/BOOK p/h

pages 101-108; July, 1945.

THE ART OF COMPUTER PROGRAMMING, VOL. 3: SORTING AND SEARCHING. Donald E. Knuth. Addison-Wesley Publishing Company, Inc., 1973.

TOWARD PAPERLESS INFORMATION SYSTEMS. F. W. Lancaster. Academic Press, 1978.

COMPUTER SOFTWARE FOR PROCESS CONTROL

MODERN CONTROL ENGINEERING. Katsuhiko Ogata. Prentice-Hall Book Company, 1970.

DIGITAL CONTROL OF INDUSTRIAL PROCESSES. Cecil L. Smith in *Computing Surveys*, Vol. 2, No. 3, pages 211-241; September, 1970.

MINI- AND MICROCOMPUTER CONTROL IN INDUSTRIAL PROCESSES: HANDBOOK OF SYSTEMS AND APPLICATION STRATEGIES. Edited by M. Robert Skrokov. Van Nostrand Reinhold Company, 1980.

SOFTWARE FOR INDUSTRIAL PROCESS CONTROL. *Computer*, Vol. 17, No. 2; February, 1984.

COMPUTER SOFTWARE IN SCIENCE AND MATHEMATICS

ORDER IN CHAOS. Edited by David Campbell and Harvey Rose. North-Holland Physics Publishing, 1983.

SMP REFERENCE MANUAL. Stephen Wolfram. Inference Corporation, Los Angeles, 1983.

DOING PHYSICS WITH COMPUTERS. *Physics Today*, Vol. 36, No. 5; May, 1983.

CELLULAR AUTOMATA: PROCEEDINGS OF AN INTERDISCIPLINARY WORKSHOP, LOS ALAMOS, NEW MEXICO. Edited by Doyne Farmer, Tommaso Toffoli and Stephen Wolfram. North-Holland Physics Publishing, 1984.

COMPUTER SOFTWARE FOR INTELLIGENT SYSTEMS

ARTIFICIAL INTELLIGENCE. Patrick Henry Winston. The MIT Press, 1982.

KNOWLEDGE-BASED SYSTEMS IN ARTIFICIAL INTELLIGENCE. Randall Davis and Douglas Lenat. McGraw-Hill Book Company, 1982.

ARTIFICIAL INTELLIGENCE. David L. Waltz in *Scientific American*, Vol. 247, No. 4, pages 118-133; October, 1982.

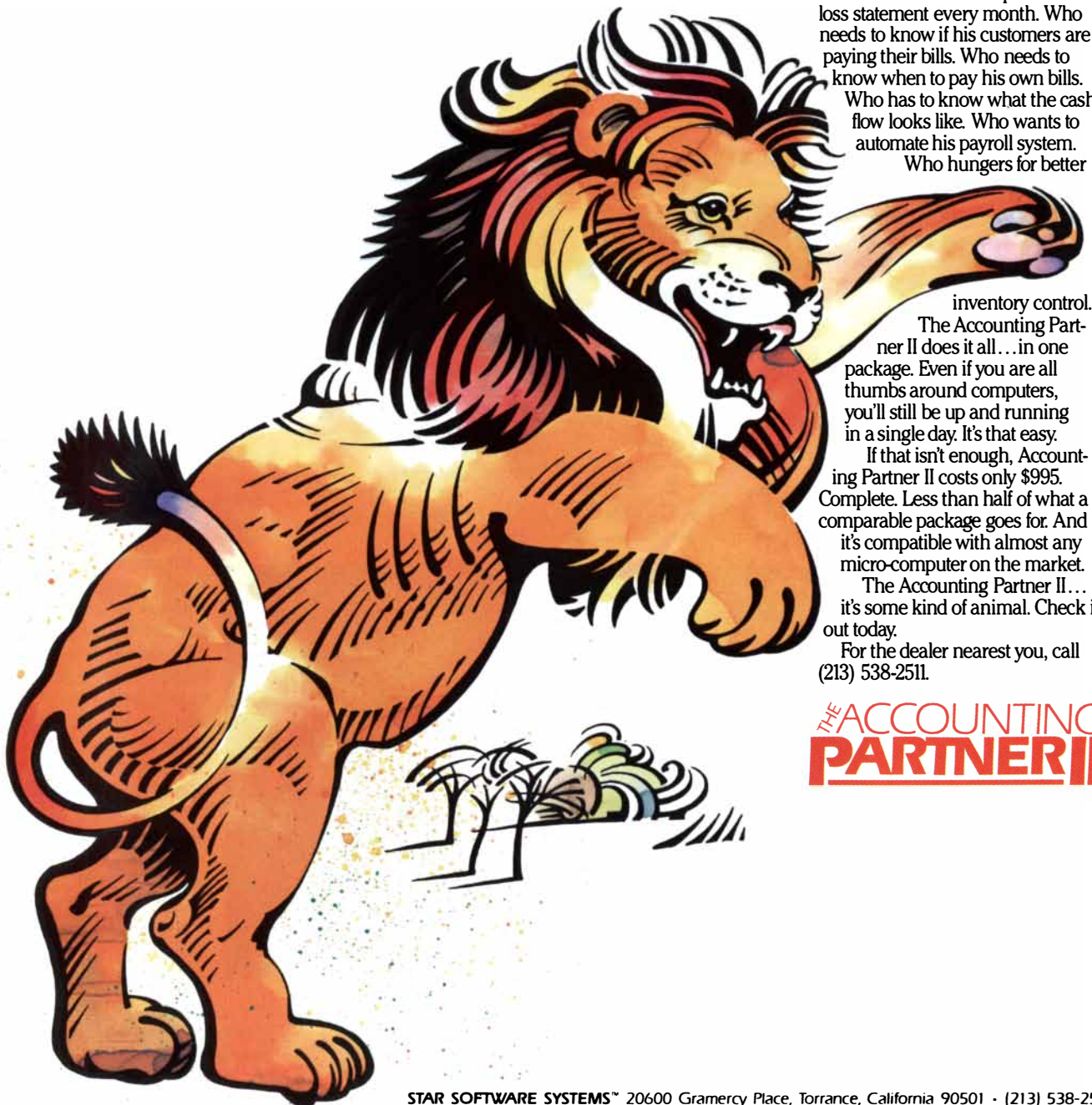
ARTIFICIAL INTELLIGENCE. Elaine Rich. McGraw-Hill Book Company, 1983.

THE AMATEUR SCIENTIST

KINEMATICS OF AN ULTRAELASTIC ROUGH BALL. Richard L. Garwin in *American Journal of Physics*, Vol. 37, pages 88-92; 1969.

THE DYNAMICS OF SPORTS. David F. Griffing. Mohican Publishing Company, Loudonville, Ohio; 1982.

ACCOUNTING PARTNER II™ THE NEW KING OF SMALL BUSINESS SOFTWARE.



What makes Accounting Partner II the best way for a small business to leap into computerizing?

Star Software Systems has probably studied the micro-computer software needs of the small business to a more in-depth degree than anyone else around. Maybe that's why nearly 50,000 small businesses turned to Star last year alone.

Now Star introduces Accounting Partner II for the small businessman who needs to keep on top of his business. Who needs a profit and loss statement every month. Who needs to know if his customers are paying their bills. Who needs to know when to pay his own bills.

Who has to know what the cash flow looks like. Who wants to automate his payroll system.

Who hungers for better

inventory control.

The Accounting Partner II does it all...in one package. Even if you are all thumbs around computers, you'll still be up and running in a single day. It's that easy.

If that isn't enough, Accounting Partner II costs only \$995. Complete. Less than half of what a comparable package goes for. And it's compatible with almost any micro-computer on the market.

The Accounting Partner II... it's some kind of animal. Check it out today.

For the dealer nearest you, call (213) 538-2511.

THE ACCOUNTING
PARTNER II

STAR SOFTWARE SYSTEMS™ 20600 Gramercy Place, Torrance, California 90501 • (213) 538-2511

Designed to do everything a modern car should. It just looks better doing it.

Technology never looked so good.

Tempo, the car that combines form and function.

Tempo's aerodynamic shape manages the flow of air over and around it to reduce overall lift and improve stability and directional control.

Tempo technology includes features like front-wheel drive for all-weather traction, four-wheel independent suspension for a smooth ride, and a High Swirl Combustion engine for quick power response.

Tempo's new tach.



You can now get a new tachometer in your Tempo as part of the optional Sports Appearance Group. This option includes new low-back bucket sport seats, a sports instrument cluster, 3-oval sport steering wheel, contoured rear seat and package tray.

This Sports Appearance Group offers a sporty new flair for those who like their Tempo a bit more upbeat.

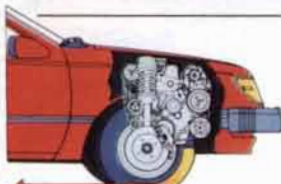
New diesel option.

Ford Tempo now has a new optional diesel engine.

It is a true diesel engine, not merely a modified gas engine. This new diesel has additional sound insulation. Cold weather starting problems usually associated with most diesels are eliminated. And, of course, it has strong diesel mileage:

41 EPA EST. MPG* **56** EST. HWY.

Front-wheel drive.



Tempo's front-wheel drive configuration is practical for all driving conditions. It gives you good traction in rain, snow and mud.

Tempo's front-wheel drive is powered by its own efficient High Swirl Combustion engine. And the whole oper-

ation is coordinated by the most advanced automotive computer in the world.

The EEC-IV. It monitors and controls engine operation precisely and instantly for optimum power output and fuel efficiency.

The inside story.



Tempo's five-passenger computer-refined interior has more room than a Mercedes 300D.

It provides an excellent combination of head, shoulder, hip and leg room.

Reduced insurance rates.

The Allstate Insurance Company offers reduced

rates on collision and comprehensive coverages to Tempo owners, because of Tempo's construction with features like 5 mph impact bumpers.

Reduced rates are realistic testimony to Tempo's structural integrity.

Best-built American cars.

When we say "Quality is Job 1," we are talking about more than a commitment. We are talking about results.

A recent survey concluded Ford makes the best-built American cars.

The survey measured owner-reported problems during the first three months of ownership of 1983 cars designed and built in the U.S., and the commitment continues in 1984.

Lifetime Service Guarantee.

As part of Ford Motor Company's commitment to your total satisfaction, participating Ford Dealers stand behind their work in writing with a free Lifetime Service Guarantee. No other vehicle company's dealers, foreign or domestic, offer this kind of security. Nobody.

See your participating Ford Dealer for details.

*For comparison. Your mileage may differ depending on speed, distance and weather. Actual highway ratings will probably be lower. Not available with A/C.



Ford Tempo

Have you driven a Ford... lately?





Get it together—Buckle up.

Are you laying out good money today for a video system that won't be good enough for tomorrow?

Panasonic gives you a portable VHS™ recorder with true Hi-Fi sound. An Auto Focus camera that records in extreme low light. Automatically. Outdoors. Indoors. Now. And years from now.

Introducing the Panasonic Hi-Fi video recorder PV-9600. And color/sound camera PK-958. So sophisticated they have everything you could want in a video system.

Connect the camera to the lightweight portable recorder. The camera focuses automatically. Even lets you record weddings, birthday parties and other special moments. Without special lights. Touch a button for instant replay. Right in the camera.

8-hour recording. No other system has more.

Slide the recorder onto its compact tuner-timer. Connects automatically. No wires. Now you can record up to

eight hours of TV on a single cassette. Even program it to record up to eight TV shows. Over a two-week period while you're away.

And whether you're recording a high-stepping pro halfback. Or your child's first steps. You'll enjoy watching them even more with jitter-free special effects. Like slow motion. Or stop motion. Thanks to Tech-4. Our four-head playback technology.

VHS Hi-Fi. Sound that goes beyond stereo.

Connect the video recorder to your stereo system. Play any prerecorded VHS Hi-Fi movie. Or musical performance. From classical to rock. You'll experience sound your conventional stereo alone could never give you.

So make sure your first video system is good enough to be your last. Panasonic. The video system that's here today. And here tomorrow.



The Panasonic Las Vegas Invitational golf's richest. Over \$1,000,000 in prize money. \$150,500 to the winner. September 19-23, 1984. Watch it on ESPN.

Panasonic
just slightly ahead of our time.