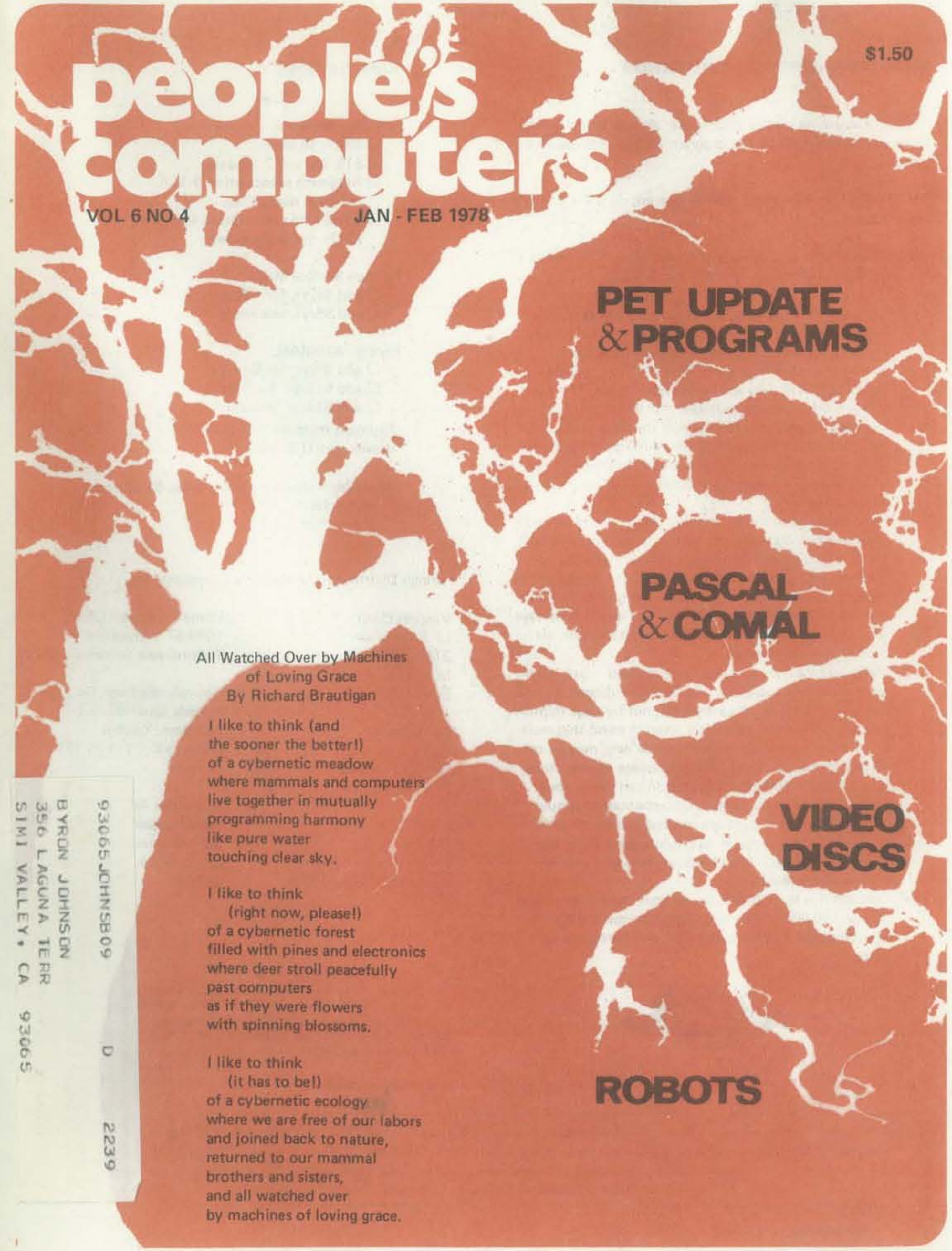


$1.50

people's computers

VOL 6 NO 4 JAN - FEB 1978

PET UPDATE & PROGRAMS

PASCAL & COMAL

VIDEO DISCS

ROBOTS

All Watched Over by Machines of Loving Grace
By Richard Brautigan

I like to think (and
the sooner the better!)
of a cybernetic meadow
where mammals and computers
live together in mutually
programming harmony
like pure water
touching clear sky.

I like to think
(right now, please!)
of a cybernetic forest
filled with pines and electronics
where deer stroll peacefully
past computers
as if they were flowers
with spinning blossoms.

I like to think
(it has to be!)
of a cybernetic ecology
where we are free of our labors
and joined back to nature,
returned to our mammal
brothers and sisters,
and all watched over
by machines of loving grace.

SUBMITTING ITEMS FOR PUBLICATION

LABEL everything please, your name, address and the *date;* tapes should also include the program name, language and system.

TYPE text if at all possible, double-spaced, on 8½ x 11 inch white paper.

DRAWINGS should be as clear and neat as possible in black ink on white paper.

LISTINGS are hard to reproduce clearly, so please note:

- Use a new ribbon on plain white paper when making a listing; we prefer roll paper or fan-fold paper.
- Send copies of one or more RUNS of your program, to verify that it runs and to provide a sense of how things work -- and to motivate more of us to read the code. RUNS should illustrate the main purpose and operation of your program as clearly as possible. Bells, whistles and special features should just be described in the documentation unless they're particularly relevant.
- Paper tapes of both the program and runs can provide us with a way to make our own listing if we need to. Then, if you give us permission, we can let CCC (Community Computer Center) sell your program cheaply via paper tape, to further the spread of inexpensive software. Finally, if we are so lucky as to have access to a system on which your program runs, we can try it out ourselves.
- Make sure your code is well documented – use a separate sheet of paper. Refer to portions of code by line number or label or address please, not by page number. When writing documentation, keep in mind that readers will include beginners and people who may be relatively inexperienced with the language you're using. Helpful documentation/annotation can make your code useful to more people. Documentation should discuss just which cases are covered and which aren't.
- If you send us a program to publish, we reserve the right to annotate it (don't worry, we won't publish it if we don't like it).
- Last but not least, please try to limit the width of your listings: 50-60 characters is ideal. Narrow widths mean less reduction, better readability, and better use of space.

SUBSCRIPTIONS

U.S. Subscriptions
- ☐ $8/yr. (6 issues)
- ☐ $15/2 yrs. (12 issues)
- ☐ Retaining subscription @ $25 ($17 tax deductible)
- ☐ Sustaining subscription @ $100+ ($92+ tax deductible)

Foreign Surface Mail
- ☐ add $4/yr. for Canada
- ☐ add $5/yr. elsewhere

Foreign AIRMAIL
- ☐ add $8/yr. for Canada
- ☐ add $11/yr. for Europe
- ☐ add $14/yr. elsewhere

Payment must be in U.S. dollars drawn on a U.S. bank.

These back issues are available at $1.50 each:
Vol 5, No 6
Vol 6, Nos 1,2,3

Foreign Distributors of *People's Computers*

Vincent Coen
LP Enterprises
313 Kingston Road
Ilford, IG1 1PJ
Essex, UK

Comicro AG
Badenstrasse 281
CH-8003 Zurich
SWITZERLAND

ASCII Publishing
305 HI TORIO
5-6-7 Minami Aoyama
Minato-Ku, Tokyo 107
JAPAN

Home Computer Club
1070-57 Yamaguchi
Tokorozawa, Saitama, JAPAN

Kougakusha Publ. Co., Ltd
Haneda Biru 403, 5-1
2-Chome, Yoyogi
Shibuya-Ku, Tokyo 151
JAPAN

Computer Age Company, Ltd
Kasumigaseki Building
3-2-5 Kasumigaseki
Chiyoda-Ku, Tokyo 100
JAPAN

"All Watched Over By Machines of Loving Grace" excerpted from the book THE PILL VERSUS THE SPRINGHILL MINE DISASTER by Richard Brautigan. Copyright (c) 1968 by Richard Brautigan. Reprinted by permission of DELACORTE PRESS/ SEYMOUR LAWRENCE.

Cover photo Drew McCalley.

People's Computers is published bimonthly by People's Computer Company, 1263 El Camino Real, Box E, Menlo Park, CA 94025. People's Computer Company is a tax-exempt, independent, non-profit corporation, and donations are tax-deductible. Second class postage paid at Menlo Park, California, and additional entry points.

people's computers

VOL 6 NO 4
JAN - FEB 1978

STAFF

EDITOR
Phyllis Cole
ASSISTANT EDITOR
Tom Williams
ART DIRECTOR
Meredith Ittner
PRODUCTION
Donna Lee Wood
ARTISTS
Maria Kent
Ann Miya
Judith Wasserman
TYPISTS
Maria Kent
Barbara Rymsza
Sara Werry
BULK SALES
Christine Botelho
BOOKSTORE
Dan Rosset
PROMOTION
Dwight McCabe
Andrea Nasher
CIRCULATION
Bill Bruneau
DRAGON EMERITUS
Bob Albrecht

RETAINING SUBSCRIBERS

David R. Dick
John B. Fried
Scott Guthery, Computer Recreations
W. A. Kelley
John C. Lilly
Frank Otsuka
Bernice Pantell
Larry Press
Shelter Institute

SUSTAINING SUBSCRIBERS

Algorithmics Inc, Bruce Cichowlas
Don L. Behm
BYTE Publications, Carl Helmers, Virginia Peschke, Manfred Peschke
Paul, Lori and Tom Calhoun
Bill Godbout Electronics
Dick Heiser, The Computer Store

CONTENTS

EDITOR'S NOTES

Since I became editor a year ago, this publication has undergone many changes. Some stemmed from readership survey results. Others reflect the personalities of the staff, the advent of home (versus hobby) microcomputer systems, and a growing dissatisfaction with BASIC. We've had letters both supporting and decrying almost each and every change—a not unexpected response given our widely diverse readers. Or at least you were a very diverse group a year ago—are you still? It's time to find out: whether or not you're a subscriber, please fill out the questionnaire in the center of the magazine and mail it immediately so we can report on preliminary results in our March-April issue—thanks.

Readers have asked about the 'Bally Library Computer' which since July has been advertized by JS&A National Sales Group. While Bally is marketing a Z-80 based video game system that can be used as a calculator, the system as it stands cannot be programmed by the user. The ads sound as if you can very cheaply buy a keyboard plus tape drive plus memory that will enable you to program, but we hear that specifications for any such add-on have yet to be written. JS&A has not fulfilled its repeated promise to supply technical information. Here's hoping Bally can live up to the crazy promises in the garbled ad copy—but I'm not going to hold my breath waiting.

Are you familiar with Dungeons and Dragons, the epic fantasy game? As I hear it, D&D inspired 'Adventures', a computer fantasy game of exploration. One of Adventure's unique and attractive features is that to play the game you communicate with the computer in short English sentences. Its swelling audience is not limited to game enthusiasts and science fiction fans—its appeal is far broader than that. Our March-April issue will feature an article on how to program Adventure-like epic games. So take heed: such games may be habit forming—avoid the article unless you're prepared to have Adventure enter your life!

Phyllis Cole

LETTERS

I have a home-made 8080A-based computer system at home with a SWTPC-AC-30 Cassette Interface and a terminal connected, along with 60K of RAM and 2K of ROM (which contains a monitor I wrote).

I would like to know if anyone has a super-Basic interpreter (i.e. 'with all the trimmings') so that I may utilize a lot of my RAM storage space. I would still have to modify it, though, as my I/O ports are numbered in a helter-skelter manner and my RAM starts at location 1000H and goes through FFFFH.

I would not mind even typing the object code into my machine for the first time, but wonder if it is possible to get a tape in Kansas City Format 'ORG'ed at 1000H.

As a college student, I do not have unlimited resources but could scrape up a few bucks to pay the person who would let my computer live in the glory of BASIC (or anything else if possible). Thank you very much.

Philip Hunt
100 E. Norwich, Apt 3
Columbus, OH 43201

Dear People,

I picked up a copy of *People's Computers* at the Personal Computing Expo in New York City and have decided to subscribe. I must be mad! I have too much to read already!

For Paul Holbrook (the high school senior who asked about colleges for computer science in your September/October issue): not all IBM monsters are batch machines. I used to work with an IBM 370/158 with 5 megabytes of memory (is that monster enough?) that had a super time-sharing system: interactive computing for 100+ users simultaneously! The name on the article doesn't always indicate the system on the inside.

I enjoyed the articles on computer networks but I'm still not sure what a 'protocol' is.

Leigh James
29B Robbins Lane
Rocky Hill, CT 06067

A 'protocol' is an agreed-upon standard.

I was both amazed and delighted to see the article 'Human or Machine? Aesthetic Preferences for Pseudo-Random Computer-Generated Patterns' by A. Michael Noll in the November-December issue of *Creative Computing*. Mr. Noll describes a series of experiments virtually identical to those proposed in my article 'Computer as Art Critic' (*People's Computers*, Nov.-Dec. '77). Those doubtful about the artistic value of computer-generated art should note that one of the computer-generated items shown in Noll's article was preferred over a similar work by Piet Mondrian, the famous Dutch artist.

Jim Day
17042 Gunther Street
Granada Hills, California 91344

Jim Day's 'Computer as Art Critic' struck a responsive chord in me, and propelled me to my personal computer-driven text-editing system to write a reply. (My pet computer is Poly 88 with a Qume printer.) The fundamental program Jim calls for was carefully explored about 60 years ago in a book, *Aesthetic Measure* by G. D. Birkhoff. Birkhoff attempted to find formulae which could compute which of two works of art was the better. Computers have been applied to the problem in a book by James Gips called *Shape Grammars* that was published recently. The subtitle of his book is: 'Artificial Perception, Shape Generation, and Computer Aesthetics'. Gips' work goes a long way, in my opinion, toward making computer evaluation of critical values meaningful. This book is by no means easy reading, but Mr. Day and others for whom the topic is interesting might find it of value. It is published by Basel and Stuttgart: Birkhauser Verlag, 1975. In any case, the field is not as unexplored as one might think.

I'd like to offer a suggestion or two that might make discussing computer evaluation of art a bit easier. First of all, distinguish between *criticism* and *esthetics*. The basic idea is that esthetics is a branch of philosophy, and criticism is not. An Esthetic is a framework on which critical judgements can be hung. Esthetics is to criticism as physics is to engineering. Books have been written about this, so I ask forgiveness for not laying it all out here. Don't feel bad, Jim, since Birkhoff also got his criticism and esthetics mixed. One of the nice things about Gips' book is that he keeps them straight.

Another specific thing I'd like to mention is Jim's statement that pi is a 'perfectly "random" number' is not at all true. First of all, there is no such animal in mathematics as a 'random number'. There is the concept of a random *sequence* of numbers. Put a bit loosely, to be random, a sequence must be unending in length, and there must not be any computer program that can produce the sequence. Knuth's *The Art of Programming* has a complete discussion of the definition of a random sequence. Since there are programs that can produce the digits of pi (of course), its digits most definitely are *not* random.

Jim's insight into the interplay of randomness and regularity in creating art is right in line with much contemporary thought on the subject, and much traditional thought as well. He will be happy to know that experiments like those he describes, where hundreds of people are asked to judge among a set of patterns for their 'esthetic' quality, have been done many times. They abound in literature of psychology.

There have been cases where science *has* discovered physical bases to our supposedly esthetic preferences. H. Helmholtz, in his book (written in the 1800's) *The Sensation of Tone* (reprinted in paperback by Dover), showed that the consonant musical intervals most cultures prefer—the octave, the fifth, and so on—are actually generated by the human ear! He explained many of the features of musical harmony by further physical and physiological experiments. There are *reasons* we like the sounds we hear.

I am glad Mr. Day has brought the subject to *People's Computers*' readers. It is a fascinating subject, which involves computer science, philosophy, mathematics, psychology, physiology and art. Thanks.

Jef Raskin
Box 511
Brisbane, CA 94005

The following letter refers to Don Quixote Starship (DQS), an incredible game originally proposed several years ago by Dragon Emeritus Bob Albrecht, founder and for 5 years editor of this publication. DQS has been fading in and out of our pages (see Volume 5, Numbers 4–6)—it's currently out.

Dear Dragons,

I'm not sure if this idea is relevant to such a venture as DQS, but it might be useful to near-earth 'operations'. Operations is in quotes because I mean it in more ways than one. This idea deals with the (medical) operating rooms on board long-duration space missions.

I'm sure it's great to dream of drones programmed to perform any operation on any person and available anywhere on your ship, but let's get back to a more near-term feasible plan. In terms of space saving and economy it would be desirable not to have to support a complete medical crew on-board. But in terms of technology it would be quite an extensive and expensive project to develop surgeon drones. What is needed is a way to provide medical care, including the ability to do major operations, from earth within the limits of today's technology. Enter the 'TELE-SURGEONS'. I believe that by adapting radioactive materials handling devices to operating room situations and by making them remote controllable (100,000 miles?) by digital space communications, an operating team on earth could perform any operation in space. All that would be required is a room on-board the spacecraft with an operating table and equipment and mechanical arms that are remote-controlled by the arms of the doctors and nurses on earth. It would also have audio-visual communications to instruct the patient and to watch what they're doing.

I'm sure it would take a lot of practice to perform an operation on a patient that you could only see on a T.V. screen, but it would be a novel service. Consider a TELE-SURGEON network spread across the planets, able to come to the aid of any spaceman, anywhere. There would never be the problem of awaking the ship surgeon for an emergency operation that he is probably in no condition to perform. There would always be several fresh medical teams, ready to perform operations 24 hours a day. (For anyone, regardless of race, color, religion, citizenship, etc.) We would have to rethink our thoughts on just what a hospital is. (Think of it, a large complex of operating surgeons and assistants, but also a place where no blood is shed! 'Place' would have lost its meaning.) It might even require a few modifications to the Hippocratic Oath. (It's about time some changes are made; it's been a long while since Hippocrates. The space age is a fitting time to do a little reworking.)

But regardless of its social and psychological implications, I think that by keeping some facets of this mission planet bound, such as the medical things, it will make more room for the fanciful ideas that we all would like to see be made possible on flights of fancy such as DQS.

Andy Riemer
Rt 6 Box 295
Hayward, WI 54843

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

Being the proud owner of a PET 2001, serial number 23, I was extremely pleased to see the sections devoted to the PET in your magazine. I strongly encourage you to continue to offer programs and techniques on using this handy little micro, since Micro Soft BASIC (the PET's language) is pretty compatible with other micros running M.S. BASIC.

I would like to comment about the Pet Drawing Program presented in the Nov-Dec issue. Although extremely ingenious, I did detect three errors in the listing given.

The first and easiest to spot was in line 30, where my issue says:

```
30 WH=50: X–20: GR=ASC(" ")
AND 127
```

The error is obviously a misprint, where the statement X–20 should read X=20.

The other two errors relate to lines 30 and 40 where nothing is shown for graphic characters.

The standardization I have used when submitting PET programs using special symbols follows. I urge adoption of these or some form of set standards so that omission errors such as these will not occur again.

My system for cursor and screen control characters uses an underscore with each symbol.

≤ cursor left
≥ cursor right
o cursor down
Q cursor up
S home cursor
* home cursor, clear CRT
R RVS on
/ RVS off

When any particular symbol is then required to be printed, such as a heart, or cross-hatch or some other graphic symbol, I use the following method.

For a solid circle (Shift-Q):
GR=ASC ("shift-Q") . . . etc.

Note that each of my examples is underscored; usually I have used lower case letters as well. The key is that lower case letters are not instantly available upon power up of the PET, nor can a user underscore any particular character while using the next line below. Thus, when seen in a program listing, the user will (or should) see that what is called for is not possible to key in as shown, and that some special function or character is being called for.

Thus, using my conventions for program listing on the PET, the lines numbered 30 and 40, in the drawing program would be:

```
30 Y=12: X=20 GR=ASC("shift-Q")
AND 127
40 PRINT "*"
```

Also, just in case you didn't know, a call to SYS(64824) (a system switch command to device number 64824) is a call to the PET system itself, thus, the current program in the PET will be cleared and a power up will be initiated again, producing the same starting heading the PET displays when first switched on.

Finally, before I end this letter, for any PET owner who tried getting the lower case letters by a POKE 59468, 14 then didn't know how to get back to the regular character set, every 16 counts from 12 and 13 will return the character set shown on the keys. Thus POKE 59468, 12 (or 13, 28, or 29, up to the limit of 255) will return the graphics rather than the lower case letters. It was interesting to note that shift-), shift-left arrow and the symbol 'pi' were also changed to different graphics as well as the 26 letters, even though 'pi' retained its value when used in the program.

Again, thank you for some fine articles on the PET. If this is any indication for the future, you have a life membership in this reader.

Mr. Craig A. Pearce
2529 S. Home Avenue
Berwyn, IL 60402

As you'll see from this issue's PET program listings, we too have tried to come to terms with the problem of listing the PET's graphic characters. We chose a method which involved using key-cap identifiers (CLR, HOME, RVS, etc.) whenever possible, to minimize the amount of 'memorizing' needed. By using square brackets and upper case letters we can prepare a program for listing by typing a version of the program on the PET, whether we're in graphics mode or lower case mode. Is your use of the underscore for typing listings on a typewriter? As you noted, the underscore can't easily be used on the PET unless you leave room to do so between each line of the program. Also, many printers can't cope with the underscore. Even the composer used to prepare most of the text for this magazine has no capability for underscoring.

Thanks very much for your comments, ideas and useful information. We look forward to more input (including programs) from readers for SPOT, the Society of Pet Owners and Trainers (see the article in this issue).

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

How exciting to read of your 'PET's First Steps' in the Nov-Dec. issue of *P'sCs*! Your article pointed out several important aspects about PET BASIC programming that just weren't available anywhere in current personal computing literature. Your educational contribution to PET users should prove enormous, especially considering that there just isn't any other software documentation available from Commodore or any other source.

Please feel assured that PET users everywhere will sincerely appreciate seeing more PET programs in print. The PET BASIC seems to be a highly refined version written by Microsoft; and although similar to most typical dialects of other microcomputer BASICs, there are still some very powerful refinements of this (PET) software dialect that need publication and more opportunity to be fully analyzed by users.

Although I can not yet say that I fully understand all the statements of your 'Drawing Program', I can say that it works well!! (I used the PETs in both the Berkeley and Palo Alto stores of Mr. Calculator to type in your program; and to the delight of the stores' staff and patrons, we had a fine time drawing graphic designs on the PET.)

So far the only bug in the PET BASIC I have heard of is that you are not able to dimension an array to over something like 255 elements. Did you find this to be a problem with your PET too?

Again, many thanks for printing programs to be used with the PET. I hope all PET users will respect your generosity as much as I do to print these special kinds of programs for our benefit. Keep on PETting–

George R. Julin
15 Poncetta Drive #322
Daly City, CA 94015

Glad you're enjoying the PET programs. Edna Wells has contributed a PET program and we hope others will join her.

PET BASIC has a number of bugs in addition to the one that limits the number of elements in any array to 255, but Commodore is currently preparing a new version of BASIC which will overcome most of them. Those of us with the early version will be able to buy ROM chips with the new version when it becomes available–just when that will be and how much the chips will cost hasn't been announced. Fortunately, most of the bugs are obscure. We'll report on them as time and space allows.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

I was delighted to read your article 'Our PET's First Steps'. On October 28 I received PET 78. While it would save small programs, it would not save large ones. Or at least would refuse to verify or load them. After about two weeks the 7167 bytes suddenly became 4854 bytes. Apparently a memory chip went bad. On November 15 I shipped it back for repair.

The bad news is that *People's Computers* came today meaning I could not try the drawing program. Two features you mentioned that I did not know were getting lower case and accessing screen memory directly. The Intro leaflet does not give much information for sure.

How can I get a copy of your PILOT interpreter for the PET? Looking forward to more PET articles and software. (My Altair runs but the PET is fun.)

H L Stuck
P O Box 2207
Chapel Hill, NC 27514

See pages 16-19 for more on the PET.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

PROGRAM ABSTRACT

Title: Graphics-to-ASCII Utility – ASCIIGRAPH
Copyright: Edna H. Wells – 20 November 1977
Permission to use, not to sell
Computer: Commodore PET 2001 (8K)
Language: 8K BASIC

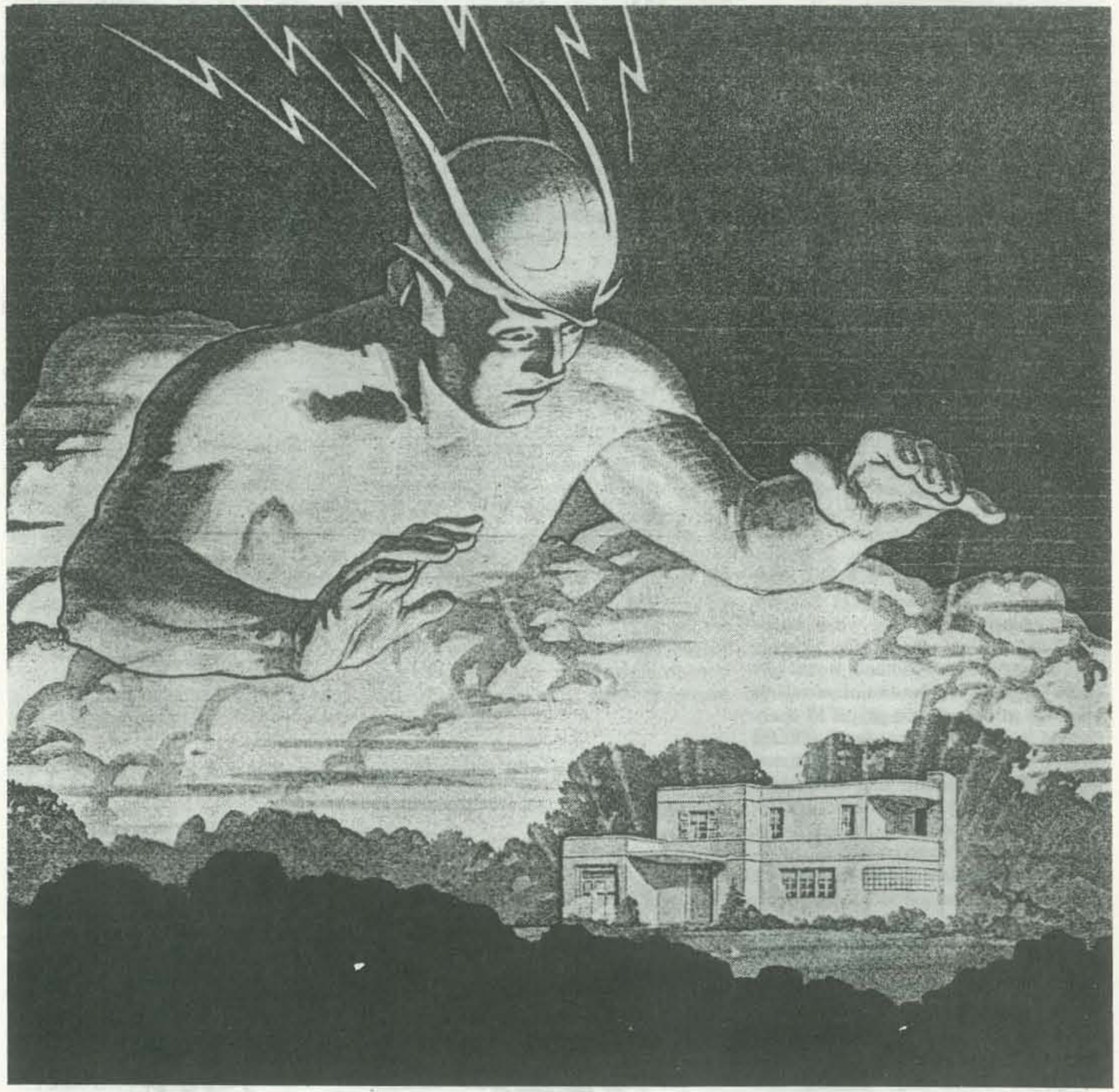
This program prints the upper-case graphic characters available on the PET and their numeric equivalents. The arithmetic expression 'CHR$(N)' is used to create the display. Under program control, the numeric equivalents shown in the display, when placed in the 'CHR$(N)' subscript, will return the graphic character. This can be placed in a string variable using a 'LET' assign statement.

The PET graphics have ASCII equivalents in the range 161 through 255. This program can be modified to return the characters in any range by changing line '25 I = J + 160'. The quantity added to variable 'J' determines the starting point. The eighty following characters will print, forty (a full screen) at a time. Pressing 'RUN/STOP' and 'RETURN' will hold the first screen display if desired.

```
1 REM ASCIIGRAPH
2 REM COPYRIGHT 1977 EDNA H. WELLS
3 REM PERMISSION TO USE, NOT TO SELL
10 PRINT "DISPLAY ASCII FOR GRAPHICS"
11 PRINT
12 PRINT "PRESS 'RUN/STOP' TO HOLD – ": PRINT
20 FOR J=1 TO 20
25 I=J+160
30 W$=CHR$(I): X$=CHR$(I+25)
35 Y$ = CHR$(I+50): Z$=CHR$(I+75)
40 PRINT W$; "  ="; I; TAB(10); X$; " ="; I+25;
45 PRINT TAB(20); Y$; " ="; I+50; TAB(30); Z$; " ="; I+75
50 PRINT
55 IF J<>10 THEN 50
60 GOSUB 100
65 NEXT J
70 GOTO 999
100 REM TIME DELAY
110 FOR K=1 TO 5000
120 NEXT K
130 RETURN
999 END
```

¶ ¶

A COMPUTER REVOLUTION?



BY PHYLLIS COLE, Editor

'Support the Revolution – buy a computer!' urges the slogan on a T-shirt I recently received. A revolution? Surely that's just advertising hype or the usual jargon that associates itself with a growing fad. Or can it be that the time has finally come for the chicken to move over and make room for a computer in the proverbial pot? Well, some say yes, and some say no.

The nay-sayers believe that over the next few years home computers will be so successfully mass-marketed that they may well compete with TV for the way in which at-home hours are spent. Just how will these numerous home computers be used? That depends on many things, but most particularly on what programs are available to excite, entertain, and last but by no means least, educate the buyer of the home computer.

COMPUTERS AS APPLIANCES

Many agree that if the home computer succeeds it will be at least in part because it has been accepted as an 'appliance', rather than as a general purpose computing machine. By appliance I mean a smallish machine which requires no special skill or training to operate, and which has only one or a few functions. The home computer as it now exists is not yet an appliance, but many such systems are heading in that direction.

What will a computer appliance be like? Let's assume it's a year or so in the future. Pretend you know nothing about computers, but wish to purchase a (mythical) Helpful Hannah system to regulate heating, lighting, smoke alarms, and burglar alarms in your home.

So off you go to a department store (already stores such as Sears have announced the sale of home computers). You are shown the sample, and told that the price includes a 'personalization' fee and an installation fee. A service contract is available, or in case of problems you can opt to bring the unit to a local appliance service center on a pay-as-you-go basis.

You buy the whole package. A few days later installer number 1 arrives to make a scale drawing of your house, showing heat vents, smoke alarms, outlet locations, and so on. You get to choose the labels for the various rooms, so in addition to the usual living room, kitchen, etc. the labels read 'Ginny's room', 'orchid room', and 'computer room'.

★ ★ ★ ★ ★ ★ ★ ★ ★

This article postulates a future in which you can buy a small computer that can perform a wide variety of functions. Such a computer system may be on the market sooner than we think – already most of the applications mentioned in this article are to be found on one small computer or another.

★ ★ ★ ★ ★ ★ ★ ★ ★

A week later installer number 2 arrives and spends some time installing the devices you want your computer to control and doing other electrician-like activities which terminate with the installation of a box with an odd-looking outlet on the wall near which Helpful Hannah will reside. The installer hands you the Helpful Hannah unit, a box about 10 x 20 x 30 cm which has approximately 20 buttons on it. You plug your TV monitor and Helpful Hannah into the wall and the Helpful Hannah into the newly installed wall box.

Now you're on your own, ready to tell Helpful Hannah just how to run your house. You press the button that says 'on/off', as the instruction manual said to do. This appears on the TV screen:

> Greetings from Helpful Hannah! In order to run your house I must be sure I have a correct picture of your house.

At this point a diagram of your living room appears on the TV screen, with dimensions, outlets, etc. all shown in their correct locations. Then this appears beneath the diagram:

> Press the 'yes' and 'no' buttons to answer questions.
>
> Is the picture of your living room correct?

You press 'yes' and get the message:

> Good. Press 'done' when you're ready to see the kitchen.

You continue to approve the floor plan until you get to 'Ginny's room' where you realize that one of the outlets is shown in the wrong place. So when asked

> Is the picture of Ginny's room correct?

You press 'no'. Then this message appears:

> Let's correct things one at a time. Type a number (then press 'done') that tells me one thing that is wrong.
> 1 room size
> 2 window(s)
> 3 door(s)
> 4 outlet(s)
> 5 heat vent(s)
> 6 smoke alarm
> 7 other

In this case, you press 4, then 'done'. You are asked 'Is an outlet missing?' and reply 'no'. Now an arrow appears on the screen; it points to each outlet in turn, asking 'is this outlet OK?' You press 'no' when the incorrectly located outlet is pointed out. Next the numbers 1 to 12 are displayed along the wall where the incorrect outlet is shown. You are told

> Type the number closest to where the outlet should be.
> Type '0' if it shouldn't be on the wall.

You type '8' then press 'done'; the picture is redrawn, and the question appears

> Now is the picture of Ginny's room correct?

You press 'yes' and continue.

Once you have verified the correctness of the floor plan, Helpful Hannah asks the necessary questions to determine how you want your house run. You specify temperatures for various parts of the house, both for when the house is occupied and when it's not. The amount of variation you can specify is limited not by Helpful Hannah, but by the type of furnace(s) and the location of heat vents. You are relieved to discover that as the salesperson promised, there is no difficulty in specifying that the room where you raise orchids be kept at a relatively high, humid constant temperature (you have already supplied the necessary thermostats, vents and valves for your computer to regulate).

In fairly short order you and Helpful Hannah fill in the details of your working relationship, then she's off and running. So you unplug the TV monitor and return it to the 8-year old, who's been impatiently waiting to finish the animated cartoon she's been making with the help of the Jumpin' Jack cartoon computer she's borrowed from the school library.

COMPUTERS AS MEDIA

Some computer systems will be far more versatile than those we class as appliances. For example, one system could carry out Helpful Hannah's functions and animate cartoons or play video games or perform some other function at the same time. Just how many people buy general purpose computers as opposed to appliance computers will depend on how easy to use the general purpose systems are and also how versatile they are.

Until recently, computers have been shrouded in mystery and hovered over by experts. To this day, esoteric incantations are necessary to do even rudimentary things on many computers – you may have to type strange lists of letters and numbers before you can begin to use the system. But times are changing – many people are now convinced that the potential of computers will remain untapped until we find a way to make computers accessible to everyone. That is, we must find a way to make computers easy to use and useful, not only to scientists but to language teachers and to retired go-go dancers writing their memoirs and to 6-year olds who want to invent new musical instruments and hear what they sound like right away. And computers should be for the contractor and the architect and the car designer and the lawyer and the Indian chief and the artist who's never before had the excitement of working with 2 million electric colors. And let's not forget the handicapped: computer-generated speech and control of all sorts of appliances and computer-based books, pictures, and art supplies are being developed to help the handicapped person broaden his horizon.

Another scenario of a mythical system will illustrate some of the potential for the computer as a new medium – as a tool for the mind. Let's assume that the system is so easy to put together that a kid, say, 10-years old can do it. It's likely that some 'putting together' will be necessary since most people will probably buy a basic system that plugs into a TV and then add to it. Suppose our current system has a keyboard, a color TV, a computer, and a 'disc drive' to permit computer

Saturday, September, 1980

The main players in the cast are yourself, your spouse, 12-year old Leslie, 6-year old Jamey, and last but by no means least, your computer.

7:00 AM	The computer turns up the heat in the living room, dining room, and kitchen (this happens earlier on work days).
7:30 AM	The computer turns on the radio to a classical music station and starts coffee; if needed, alarms will go off at times prescribed by individual family members.
8:30 AM	12-year old Leslie uses a computer program she wrote to help rehearse her part in a play in Spanish; the program 'speaks' in Spanish as it cues her and prompts as needed.
10:00 AM	Leslie is off to rehearsal; 6-year old Jamey continues work on his animated color cartoon.
10:02 AM	The computer notes that a window and door were left open; since the heat has already been turned off and people are at home, this is of no immediate concern.
11:30 AM	Jamey bows to pressure from neighborhood kids to play just one game of Wumpus; then just one game of Dinosaur; then just one game of Star Grazer; then you kick the kids off the machine so you can finish a report.
12:45 PM	Using a penlike device attached to the computer you 'draw' on the TV screen the various charts and diagrams needed for your report. When you're done straight lines are straight, curved lines are appropriately curvy, and you've used shading to effectively enhance several illustrations.
3:35 PM	Using your computer room telephone, you connect your computer to another computer which is programmed to do searches for legal references. In less than 15 minutes you have verified one reference for your report and completed two others. The bill for the service will appear on your next VISA bill.
5:00 PM	Jamey and friends excitedly use a plant identification program to identify the latest specimens they'll add to a growing collection of wild plants. The program, a birthday present from Jamey's grandmother, has also been used to identify the new plants that keep appearing in the yard and garden.
5:12 PM	The computer briefly interrupts Jamey with the information that the temperature in the house will soon drop below the desired minimum unless the side door and the window in Leslie's room are shut. Jamey shuts them.
6:55 PM	Heat goes on in the occupied rooms.
7:30 PM	Leslie and her friends get together their Saturday night band; one band member, the computer, plays a different instrument (usually one invented by the kids) each time the group meets.
9:00 PM	Leslie and crew repair to the kitchen and your spouse resumes a computer-assisted course for ground school. Upon discovering that much of the learning required for a pilot's license could be done at home on your own computer, both you and your spouse decided to study for your pilot's licenses. You both decide to complete the ground school course by the end of the month and to rent the flight simulation program from your flying club as soon as possible.

And so it goes. . .

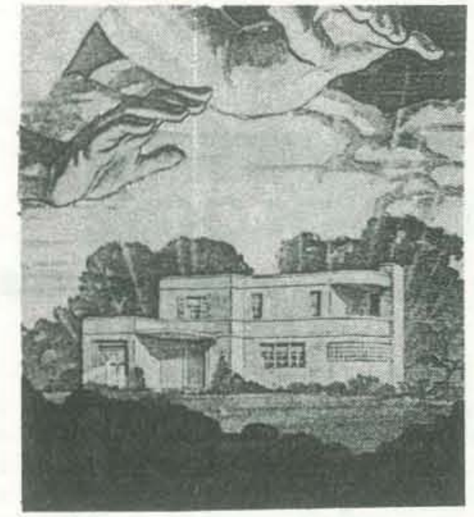
programs to be permanently stored and quickly retrieved. In addition, our system has some more specialized components: we have a printer that can print words or pictures shown on the computer onto paper; we have a way to generate high quality sound – and speech – using the computer; we also have assorted small devices used for pointing to the TV screen in different ways.

In the box on the opposite page is a brief glimpse into 1980. Join in the fantasy–you, your spouse, two children, and a computer have starring roles.

In the scenario, 'computer program' refers to a set of instructions written in a special programming language the computer can understand. At the time of the scenario there exists at least one programming language suitable for doing all the types of activities described in the scenario. And we assume that even young kids can learn to instruct a computer using this language.

WHAT NEXT?

Over the next 6 to 12 months, you'll likely see many more firms using mass advertising to sell computers to the 'average American' consumer. These computer systems will come in various shapes and sizes, with various capabilities. Some will have color; some will have optional 'attachments' to let you play video games, control appliances, and do automated telephone answering; others will also be useful to small busnessses. Some special purpose systems, such as



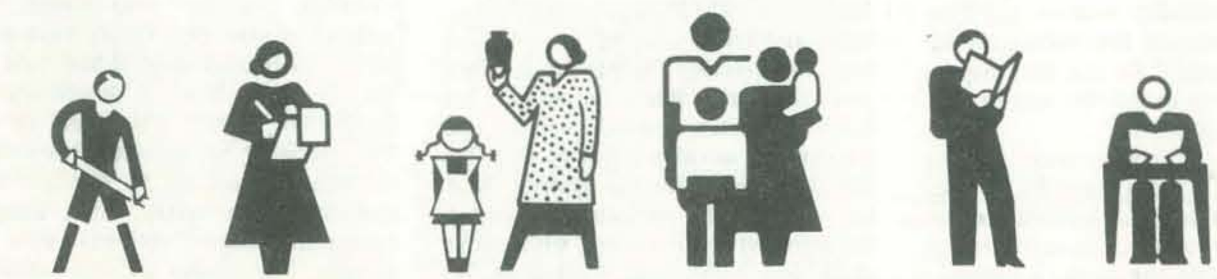
those that support only one or two video games, will fall into the 'appliance' category. Others will be designed as more general purpose machines. Prices will vary from several hundred dollars (for a computer without a TV) on up.

The availability of numerous high quality, low cost, useful computer programs will make general purpose computers attractive to many people. Computer manufacturers hope such programs will be developed by free-lance programmers and then sold to the manufacturers on a royalty basis. The manufacturer will then mass produce the programs, probably at first on inexpensive audio cassette tapes, and then sell the programs for $5-20 for most programs for home use. (Expect equipment and programs to be used in small businesses to cost a good bit more than items for the home user.) The dream of low cost quality programs appeals to many – consumers dream of the many wonderful things they can do so inexpensively; programmers dream of being their own bosses while having a reliable market for their wares; manufacturers dream of higher demand for their products because so many interesting programs are available.

But in what computer language or languages shall such computer programs be written? We need a language designed to deal with all the activities described in both scenarios and more. And such a language must not only be powerful, it must be easy to learn, so that all people, kids included, can quickly and easily learn to use it to do interesting things. Are we likely to see such a computer language in the next few years? I'm optimistic, in part because so many ideas are converging from so many sources – more and more people are agreeing on the elements needed for an adequate computer language.

HOME COMPUTERISTS OF THE WORLD . . .

Has the urge to join the home computer brigade hit you yet? If so, expect to find yourself swept up in the enthusiastic fervor marking the early days of any revolution. The confusion, camaraderie, and the 'this-could-be-the-start-of-something-big' feeling helps make bearable the delays that have come to be characteristic of the computer industry. If you've not yet united with the home computerists of the world, you have exciting times ahead. Tomorrow's choices will likely be far more exciting than today's dreams. . . □



AN EDUCATOR'S GUIDE TO VIDEODISC TECHNOLOGY

BY R KENT WOOD and KENT G STEPHENS

R. Kent Wood is associate professor of instructional media, Utah State University, Logan. Kent G. Stephens is associate professor of educational administration, Brigham Young University, Provo, Utah. *This article originally appeared in the February, 1977 issue of* The Phi Delta Kappan. *Reprinted with permission.*

Over the past several years it has become fashionable to categorize media as "print" or "nonprint." The recent development of videodisc technology makes such a dichotomy anachronistic. The videodisc is truly a mixed medium.

Consider that an average book has roughly 250 pages. Three hundred such books can be stored on a single side of a silver luster 12-inch videodisc. It looks very much like the 12-inch stereophonic records you now play on your phonograph. The videodisc itself is a pressed "floppy" product of clear mylar-type plastic overlaying an imbedded metallic center. Your 300 books can be retrieved and projected on a video screen from one of the discs. Or you can store and retrieve up to 50 hours of high-fidelity music. Or you can do the same with several educational films.

In one complete revolution of the videodisc, a single page or picture is recorded as "analog data." This makes it possible, for reasons we can't go into here, to play back a single page with greater fidelity than we get from current educational film techniques for "freezing frames." Or the data may be played back as sound. Or you can get both image and sound.

Consider: A classic book can be recorded on a single videodisc in both the original form and in a movie version. The first one minute of storage space on the disc can be the book; a one-hour and 59-minute film of the same book can be recorded on the remaining storage surface. Students can read the book page by page in a single-frame sequence. Then, at their command, the motion picture sequences unfold. Videodisc can move from print to nonprint at the flip of a switch.

One of the great advantages of videodisc technology is the low cost of materials. The projected materials cost of a single videodisc is approximately 50 cents; the total cost for a 30- to 45-minute videodisc program is roughly $10. Thus the cost of materials is only 5% of the total disc price. By contrast, more than 90% of the cost of a videotape program is of necessity in materials. Even the new low-cost Sony Betamax videotape sells for $16 for a one-hour blank.

There are two major systems of videodisc technology. The partnership of Telefunken of Berlin and Decca Records of London has been operative in Western Europe for several years now. The Telefunken-Decca system uses a stylus electrical pick-up comparable to the Radio Corporation of America system in the U.S. The Music Corporation of America represents the laser noncontact optical system. The two systems are compared in Figure 1 diagrams.

In the last several years more than $200 million has been spent on videodisc research in the U.S. and Western Europe. In 1972 Phillips of the Netherlands and MCA unveiled projects that had been secret. During the last few years the Zenith Radio Corporation has had a 90-person research team in two laboratories working furiously to refine the laser-type videodisc system. RCA has expended its research resources on the diamond stylus system, with a large plant and technicians located in St. Louis. MCA is planning to place 11,000 major motion pictures on their videodisc system, along with more than 400 educational films. In all, some 17 major companies have been expending research and development funds on videodisc technology.

It has been estimated that more than a half billion dollars has now been spent on general TV hardware systems development, including videodisc, and that less than 1% of that amount has been devoted to development of software. Eventually, however, videodiscs may have the effect of bringing better quality programming to the TV medium, because more of the funds committed can be devoted to development of worthy programs. As noted, the great advantage of videodisc lies in low material costs.

What can we expect of videodisc technology in the immediate future? MCA planned to put the first commercial videodisc player units on the market as early as Christmas, 1976, but continuing market surveys appear to have delayed that event for several months.

What will it cost to buy a player unit and equipment to convert your present color TV set to videodisc? The player units will sell for between $500 and $700. They look very much like the current audio record turntable units you presently use. The units have a wire lead that is connected to the VHF antenna terminals, just like your present TV antenna connection. Simply tune your set to an unused channel and turn on the videodisc player. A beautifully defined color picture will appear on the TV screen. It has greater fidelity than the images projected by standard film systems in use today. Also, imagine tying giant "lean-to" screens to your TV receiver. The screens are now available, showing life-size images. The new system can play motion sequences of film, move to slow motion or fast, or freeze a single frame, all at the user's command.

Some educators have also suggested tying computer technology systems to videodisc in order to provide programmed instruction. Others have suggested that in the future, journals – even such as the one you are now reading – can be produced in videodisc formats. The mass production costs would be far less than distributing the printed page in vogue today. Imagine this article with live demonstrations of videodisc technology shown in living color with motion. Imagine what it would be like in the future to hear and see Jerome Bruner, B. F. Skinner, or Carl Rogers speaking to you about their current research findings, with film clips of the actual research settings. One videodisc expert has suggested that videodisc technology is like history repeating itself; that is, they see shades of Gutenberg and another landmark invention comparable to the printing press.

One authority has listed these advantages for the videodisc as a new medium:

– There is widespread agreement on the projection of the base cost of an hour's information on the videodisc at about one cent a minute – not 40 cents or $4.

– The videodisc player itself could be only one-third the retail price of similarly functioning players in the film and tape technologies.

– From a single paper-thin roll-it-up-and-send-it-through-the-mail videodisc you could selectively call up any one of the more than 100,000 single picture frames stored on one side and display it indefinitely at the press of a button.

– You could mix stills and motion randomly, manually, and on a pre-programmed basis.

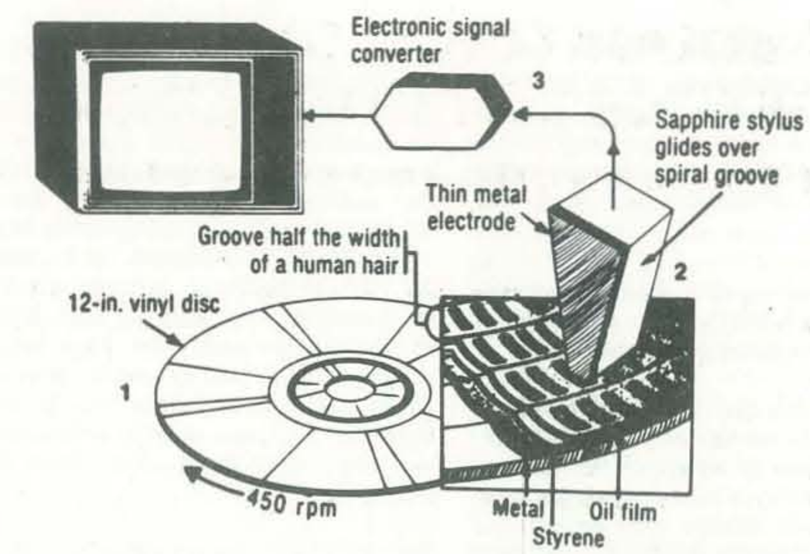
– You could go forward and backward at will, jumping from the first to the last part of a program in several seconds.

– Audio fidelity would be better than that currently provided by good quality LP audiodisc or tape, and there could be four channels available.

– In certain of the approaches being discussed, there is no mechanical contact made with the disc itself. This means that all of these advantages would never degrade through use; the disc would never wear out.*

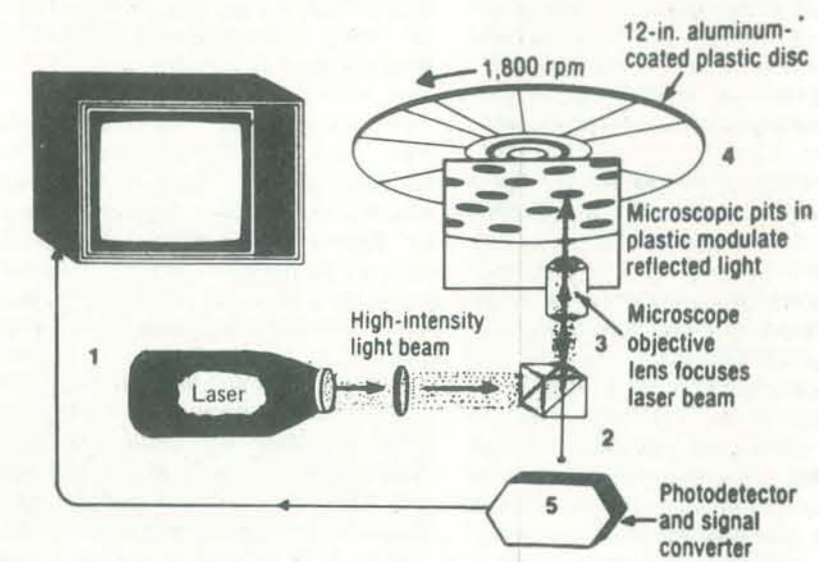
The videodisc player units seem a sure bet for huge commercial success. If they do make it big in the commercial market, chances are that they could become a prime tool used by educators in the teaching/learning process. □

Figure 1. Two Major Videodisc Systems



Telefunken-Decca System

Electrical Pickup. The picture and sound code on a spinning disc (1) is picked up electrically by a stylus (2) that transfers signals to a converter (3). The converter processes the signals into electronic form accepted by a TV set.



MCA System

Optical Pickup. A laser (1) generates a light beam aimed by a prism (2) and focused by a lens (3) on a disc (4) coded for picture and sound. Reflected light strikes a photodetector (5) that converts it to signals that are processed and fed to a TV set.

*Ken Winslow, "A Videodisc in Your Future," *Educational and Instructional Television*, May, 1975, pp. 21, 22.

Video Discs: Magic Lamps for Educators?

BY LUDWIG BRAUN

As the microelectronic revolution continues, it becomes increasingly apparent that human culture is standing on the verge of the first really new *era in the continuity of thought since the written word. Carl Sagan points out in his book* The Dragons of Eden, *that humans are the only animals capable of storing information outside the body, thus making it available for future generations.*

To date, ideas and information have been generated by the human mind and deposited for use by others in the form of pictures and the written or printed word. This process, whether done by medieval monks and scribes or the modern offset press, has remained one of information transmission. *With the advent of the electronic computer we are moving from an era of information transmission to one of information* processing *and transmission. We have created machines capable of taking available information, manipulating it according to human-devised schemata to generate new information,* recording *that and transmitting it–thus making it in turn available for processing.*

The invention of writing represented a fundamental change in the way human thought was transferred and recorded. Gutenberg's development of movable type represented a quantum jump in the volume, speed and cost effectiveness of written communication. The invention of electronic data processing represents a unique shift in the handling of information–the automated generation of new information. But only with the development of microminiaturization has an analogous quantum leap in volume, speed and cost effectiveness become possible. The full impact this revolution will have on human culture is something we can only dimly imagine.

The following article has been excerpted from a report prepared by Professor Ludwig Braun of the State University of New York for The National Institute of Education. The possibilities of interfacing the immense storage capacity of video discs to computers opens exciting possibilities in education and in other fields as well. *Tom Williams*

Remarkably, the history of video-disc recording goes back to 1927, when John L. Baird engraved video signals on a gramophone disc, in much the same way that audio signals were recorded. Baird was limited by the technology of his time to a 5,000 Hertz bandwidth. As a result, his pictures had only 30 lines and 15 black-and-white elements per line. Even though the resolution was poor, Baird deserves a great deal of credit. It took most of the ensuing half century and legions of engineers to bring to practical implementation the concept which he pioneered.

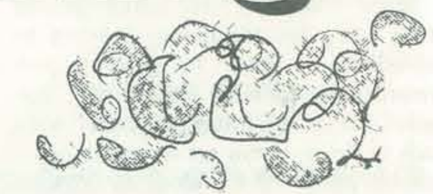
The next step in recording of video images occurred in 1956 when Ampex announced magnetic-tape recording of video images. The major technological breakthrough here was the frequency modulation of the video signal prior to its recording. Because frequency modulation is amplitude independent, the video signal essentially is recorded in two-level, or binary, form.

There are essentially two kinds of video-disc systems: The electromechanical systems in which there is mechanical transduction (hence physical contact); and electro-optical systems. The principal differences between these two systems are described in the article by R. Kent Wood and Kent G. Stevens on Page 14.

In this report, we shall focus on electro-optical systems because of their educational advantages and specifically on the Philips/MCA system because it will enter the American market place first and is likely to dominate this market for that reason, and because it appears to be superior functionally to other systems.

In the Philips/MCA system, during normal play, the disc rotates at 1,800 rpm. It has 54,000 tracks each containing a single frame of video information encoded in NTSC (National Television Systems Committee) format. This provides thirty minutes of video program time per side.

Because each frame occupies an entire track, there is one frame per revolution of the disc. As a consequence, it is possible



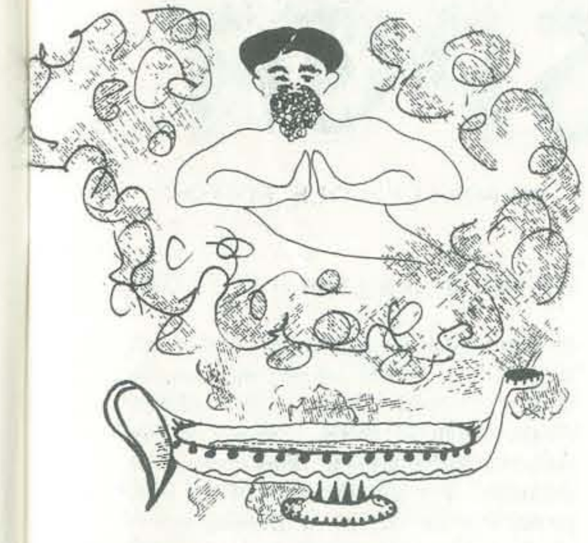
to use the system as a slide projector in the stop-frame mode by jumping back one track at the end of each revolution; hence, the image stays still. It is possible also to generate slow motion either forward or reverse by controlling the motion of the read head.

This system records the video signal in binary form by frequency modulation (as in the Ampex system). It is worth looking briefly at the information storage capacity of this disc from several viewpoints. These are:

- It is possible to store 54,000 individual slides on one disc side using the stop-frame mode.
- By compressing the audio signal by an effective factor of 300 before recording and then 'uncompressing' by the same factor on play back it is possible to store 150 hours of music or other audio information on one disc side.[1]
- A talking encyclopedia could contain 27,000 slides and 75 hours of commentary.
- There is a total capacity of 185,625 bits per track, or 1.25 *billion* bytes per disc side! This enormous capacity is available as a read-only-memory for storage of data or computer programs. It also is possible to store information contained in books in binary form (one byte per character) – the *Encyclopedia Britannica* could be stored in 50M bytes of memory, only four percent of the capacity of a single disc.

Because the video-disc systems are aimed at the consumer market, the signal encoding has conformed to the NTSC standard waveform. If educators are to be able to take advantage of the cost benefits of large consumer demand, they must conform to this same NTSC standard. This means that special interfaces are required between the information source and the disc, and between the disc and the computer or other information medium.

In the Philips/MCA system, if the focusing arm is stationary, the tracking mirror can



address any one of 100 tracks, and can move from any track to an adjoining one in 60 microseconds. Since 186K bits may be stored in one track, 100 tracks can contain 18.6M bits (or about 2.3M bytes) of information. This is 20 times the capacity of a mini floppy disc. The table compares the capacities of video discs with full track utilization, video discs using only 100 tracks, and the North Star mini floppy disc (one of the most popular discs in the microcomputer field).

A brief look at the table reveals that the 100-track video-disc system is clearly superior to the mini-floppy disc system by at least an order of magnitude in capacity, transfer rate, and access time, with system cost, disc cost, and error rate being comparable in both systems. The only significant disadvantage of the video disc as a mass memory device is that it is a read-only memory, whereas the floppy disc is read/write memory.

One potential drawback of the video disc is production of copies. The replication equipment is sufficiently expensive that few schools will be able to justify the cost; however, both Philips and MCA are planning to establish replication centers at several locations. A customer must submit a video-tape master with the information (pictures, data, etc.) encoded in NTSC format. The replication center will make a disc master for about $1,000 and will make disc copies for $1 each; thus 200 copies will cost $6 each, and 1,000 copies will cost $2 each. It should be noted that these costs represent only the replication cost, not the cost of generating the information to be stored on the disc.

Unfortunately, partly because video-disc systems are new, and partly because few people have had any access to such systems, there has been essentially no development of educational applications – indeed, there has been little detailed thought devoted to such applications. Mr. James Baker, of the U.S. Army Research Center, who is responsible within the Army for video-disc applications, feels that the only people in the United States who are doing significant work in this area are Dr. Robert Brantsen of Florida State University and the WICAT group in Utah under the direction of Dr. Dustin Heustin and Dr. Victor Bunderson.

In the author's search of the literature and in discussions with people around the country, the following areas of educational application of video-disc systems have emerged:

Linear video. Here, the video disc is used in place of movie film. The cost of a 20-minute educational film is well over $200; even rental of the film for a single showing costs $15-25. The estimated purchase cost of a video disc film is $10-15. Even if this were the only educational application, it would be exciting, because a school can build a 'film' library for less than the present rental budget. In addition, the teacher has the capability to achieve fast forward, and easy reversing, which are not available with film projectors.

Archival storage. Because of the large capacity of video discs to store information, and their low cost, they are very attractive as a medium for storage of computer programs and data files, and for storage of books, journals, and other printed materials.

Programmed instruction. The possibility of combining motion and still frames in full color with text and audio, is a very exciting alternative to more conventional programmed texts. Clearly, the courseware developer has a much more flexible and powerful tool than he ever has had with paper and printing press. Drs. Heustin and Bunderson and their WICAT group in Provo, Utah are developing an experimental video-disc programmed 'text' in high-school biology for the McGraw-Hill Book Company.

Interactive use with the computer. Dr. Alfred Bork of the University of California at Irvine suggests[2] that a video disc will contain a complete multi-media teaching package. Such a package might include a color video sequence presenting relevant historical background and pertinent information to set the stage for a complex computer simulation of some phenomenon in biology, chemistry, or physics. During interactive execution of the simulation program, and depending upon the student's actions, the computer will call upon the video disc to help it to generate appropriate supporting graphics, or an audio sequence, a background still frame, or even a new computer program. After the simulation is complete, the computer will call a testing program from the disc and check the depth of the student's understanding. Based on the results of

Continued on page 47.

Characteristic	Video disc full cap.	Video disc, 100 track	Mini floppy disc
Total capacity	1.25×10^{3}M bytes	2.3M bytes	100K bytes
Transfer rate	7M bytes/sec.	7M bytes/sec.	16K bytes/sec.
Cost/bit	6×10^{-8}cents	4×10^{-5}cents	5×10^{-3}cents
System cost	$500+interface	$500+interface	$700
Disc cost	$2(lots of 1,000)	$2(lots of 1,000)	$5
Avg. access time	5 sec.	3 millisec.	463 millisec.
Error rate	10^{-9}	10^{-9}	10^{-8}-10^{-11}

Comparison of video disc with a popular mini-floppy disc.

SPOT

BY PHYLLIS COLE, EDITOR

PET photo courtesy of Utter Chaos

Commodore's PET is a factory assembled personal computer based on a 6502 microprocessor. The unit includes a key-board, cassette tape unit, CRT, some graphics, upper and lower case, and an extended 8K BASIC. The system with 4K of user memory costs $595; the 8K model costs $795. For details, see the last 2 issues of People's Computers.

Until there's some sort of formal PET owners' organization, *People's Computers* will provide space each issue as a forum for PET people. The name of the forum will be SPOT (Society of Pet Owners and Trainers) unless something better comes along. Possible uses for the forum include swapping software and ideas (PET projects? and maybe Teachers' PET?) and complaints (obviously PET Peeves). Perhaps projects that involve hooking the PET to other devices should come under the heading ComPETible Stuff?

TEACHERS' PET

The 'we' of this article refers to a group of computer professionals – including your editor – who have purchased a PET as part of a project aimed at integrating computers into the daily routines at a local school. To date we've concentrated on preparing a wide variety of sample programs that we've shown to both kids and teachers. As of January we'll begin workshops with teachers and junior high age students to teach them how to program in a version of PILOT. The workshops will focus on the development of a body of programs to support the topics that will be studied in the latter half of the school year in the junior high. In addition, some programs for younger kids will be developed; the junior high school will serve as a primary resource for introducing the PET and PILOT to others in the school We plan to include even nursery-school age children in the project – already we have a demonstration program for youngsters that age.

TAPE TIPS

We've found that the most reliable tapes to use on the PET are Maxell, TDK, and Memorex. We've had some problems with the cheapest Radio Shack tapes. We definitely do not recommend Scotch tapes: we confirmed reports that when used as computer tapes they snag– although the problems of snagging may be reduced if very short tapes are used.

In its manual, Commodore recommends using 'Nortronics' Brand tape head cleaner. They recommend 'Nortronics', 'Hand-de-mag' and 'Robins' brands of head demagnetizers. Tape deck head cleaning and demagnetizing needs to be done every 50-100 hours of tape running time or when you have trouble reading tapes reliably.

SOFTWARE AVAILABILITY

PET programs are beginning to be advertised–let us know which ones you've tried, and whether you recommend that others purchase them. Some of the programs we've been working on are now available. For a description of the programs, a price list, and a licensing agreement send a stamped, self-addressed envelope to Computer Project, Peninsula School, Peninsula Way, Menlo Park, CA 94025.

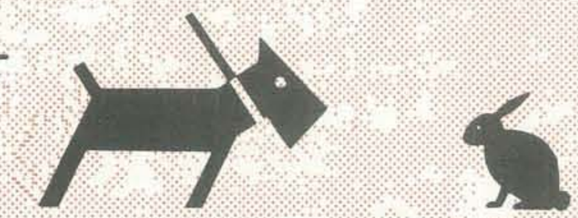
DIAL-A-PET

We've used a board from The Net Works (see announcement section) to connect our PET over the phone to other computers–and so we got the listings that accompany this article.

The PCNET (Personal Computer Network) Committee is involved in setting up protocols (standards) to enable different kinds of home computers to communicate with one another. We're implementing the protocols on the PET in BASIC. Using Commodore BASIC and The Net Works' board we can barely keep up a 30 character per second speed. However, we expect to be able to implement all the PCNET protocols in BASIC. So far, our PET can automatically dial the phone, and can receive and acknowledge packets of information. (See 'Computer Networks' by Larry Tesler, Volume 6, Number 2). Other folk are implementing them on other home computers, often in assembly language. You'll be hearing more of these efforts in later articles.

To receive the latest draft of the PCNET protocols, send $3.00 to cover replication and mailing costs to Dave Caulkins, 437 Mundel Way, Los Altos, CA 94022.

The Society of PET Owners and Trainers



NOTATION

Most printers can't print PET graphics. Also, Commodore BASIC lets you put cursor control characters into PRINT statements, and inside the quote marks displays special *reverse* characters when you do so. And most printers can't cope with printing in reverse (i.e. white on black).

To help make PET listings more readable, we've decided to indicate special stuff inside square brackets, using commas to separate items. Sometimes a number precedes an item to tell how many times it should be repeated. We use HOME, CLR, RVS, OFF and INST as shown on PET keys. Sometimes SPACE is used to show where a space should be typed. A single character indicates that the *graphic* shown as the *shifted* character should be used. Examples are provided in Figure 1.

Finally, you may have noticed colons at the beginning of some program lines. Colons indent lines inside FOR loops to help show the limits of the loop.

Listing says	You type	The screen shows	What happens when program is run
PRINT "[CLR, Q]"	PRINT " CLR key shift-Q "	PRINT " black heart on white white dot on black "	Cursor goes home (upper left) and screen clears, white dot printed in home position.
PRINT "[HOME,3 DOWN,S]"	PRINT " HOME key cursor down 3 times shift-S "	PRINT " black S on white 3 black Qs on white white heart on black "	Cursor goes home, then down 3 times (i.e. to row 4), next prints a white heart.

Examples of Notation for Special Characters
Figure 1

STARS

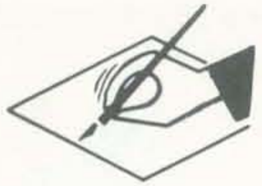
Here's a familiar number-guessing game that we've adapted for the PET. The program randomly selects a number from 1 to 100 for you to guess. You try to guess the number, then the program prints out from 1 to 7 stars (asterisks) depending on how close your guess is. When you guess the number, the screen goes blank then 40 stars are randomly displayed in the top 20 lines of the screen.

100 sets lower case mode.
110 clears the screen; moves the cursor to second row.
120-210 prints out the instructions.
220 selects X, the random number to be guessed.
230-240 gets the guess, G.
250 branches to line 40 if the correct number was guessed.
260 calculates D, the 'distance' that the guess G is from the correct answer, X.
270-290 loops to print from 1 to 7 stars, depending on the value of D.

D:	1	2	4	8	16	32	64
Q:	0	1	2	3	4	5	6
No.*:	7	6	5	4	3	2	1

300 gets next guess.
405 clears screen.
410-430 loops to print 40 stars in randomly chosen locations in the first 20 rows of the screen. In these 20 rows there are 40*20 or 800 locations to choose from. We randomly select Y, an integer from 1 to 800. The command POKE 32768+Y, 42 prints an asterisk (ascii code 42) in the Yth location on the screen.
440-490 moves the cursor home, then to the 22nd row (the second blank line), prints a message about how to restart the game.
500 waits until something is typed from the keyboard.
510-530 if RETURN was pressed, starts the game again; if anything else was typed, ends the game.

```
10 REM STARS
100 POKE 59468,14
110 PRINT "[CLR,DOWN]"
120 PRINT "WELCOME TO MY GALAXY. I'M IN CHARGE"
130 PRINT "OF THE STARS HERE. PLAY MY GAME OF"
140 PRINT "***STARS*** AND GET SOME STARS FOR"
150 PRINT "YOURSELF.": PRINT
160 PRINT "I WILL THINK OF A WHOLE NUMBER FROM"
170 PRINT "1 TO 100. YOU TRY TO GUESS IT. THE"
180 PRINT "MORE STARS I PRINT THE CLOSER YOU ARE."
190 PRINT "IF I PRINT 7 STARS ******* YOU ARE"
200 PRINT "VERY VERY CLOSE!!!"
205 PRINT: PRINT
210 PRINT "OK STARSEEKER, I'M THINKING OF A NUMBER."
220 X=INT(100*RND(1))+1: N=1
230 PRINT
240 INPUT "WHAT'S YOUR GUESS";G
250 IF G=X THEN 400
260 D=ABS(G-X)
270 FOR Q=LOG(D)/LOG(2) TO 6
280 :PRINT "*";
290 NEXT Q
300 N=N+1: GOTO 230
400 REM GOT IT
405 PRINT "[CLR]"
410 FOR Q=1 TO 40
420 :Y=INT(800*RND(1))+1: POKE 32768+Y,42
430 NEXT Q
440 PRINT "[HOME, 21 DOWN]";
450 PRINT "YOU GUESSED MY NUMBER IN";N;"TRIES!!!"
460 PRINT: PRINT "PRESS RETURN TO PLAY AGAIN."
470 PRINT "PRESS ANY OTHER KEY TO STOP."
480 GET A$: IF A$="" GOTO 510
490 IF ASC(A$)=13 GOTO 205
500 PRINT: PRINT "BYE FOR NOW!": PRINT
510 END
```



DRAW UPDATE

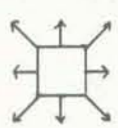
In our last issue we published a program, DRAW, that allows you to draw pictures on the PET. Even young children can quickly and easily learn to use the PET's graphic characters using this program.

Now we've added to the program. You can save on tape and retrieve the pictures that you draw. We've also modified the program to PRINT the characters on the screen, instead of using POKE. The program will run on either a 4K or 8K PET.

The Target. The program treats the screen as a grid of cells, 38 across and 24 down. When it starts, it clears (blanks) the screen and displays a large round dot (the 'drawing symbol') in a cell near the center. That cell is the initial 'target cell'. A white square blinks at you occasionally to let you know where the target cell is.

To draw a picture made of dots, use the program's target-motion keys. They are not the same as the PET's cursor keys. Instead, the digit keys 1 through 9 are used to make the target move one cell in any of eight directions.

7	8	9
4	5	6
1	2	3



Pretend the target cell is on the '5' key. To move it left, press '4'; to move it up and right, press '9'; and so on. Whenever the target moves, it will inscribe the drawing symbol in its new cell.

The Drawing Symbol. When you are tired of dots, press any graphic key. Shifting is only needed for the graphics on the 1-9 key. The graphic character on the key will become the new drawing symbol. It will be inscribed in the target cell.

Press RVS, and the color of the drawing symbol will be reversed. Subsequently chosen drawing symbols are not affected.

To erase, make SPACE be the drawing symbol. The reverse SPACE draws white stripes. DEL erases the target cell without changing the drawing symbol. You can 'un-DEL' using the '5' key.

Other Features. To move the target without changing the picture, get rid of the drawing symbol by pressing either of the CRSR keys, then use the digit keys to move the target. When you are ready to draw again, press a graphics key or RVS.

When you want to admire your drawings without the target cell blinking at you periodically, press RETURN. Then, to make it blink again, type any other key; '5' is a good choice.

To move the target to the center of the screen, press HOME. To start a new picture, press CLR (note that you'll have to shift). To stop drawing so you can do something else with your PET, first press RETURN and then press STOP.

Saving and Retrieving Pictures. When you're ready to save a picture, type the left-arrow cursor control key—i.e. the one that points to the tape recorder. First the cursor stops blinking while your picture is measured; during the 30 seconds or so that this takes, little streaks of light may dance about the screen. Next the first 3 lines of your picture are erased and you're asked 'SAVE FILE NAME?' Type in a name for your picture and press RETURN and then you'll be told to press RECORD and PLAY on the recorder. It'll take about a minute (more or less, depending on the size of your picture) to save your picture on tape. When saving is complete the program clears the screen and displays the initial drawing symbol, a dot, in the center of the screen.

Now rewind the tape. To retrieve your picture from tape and print it on the screen, type the up-arrow cursor control key—the one that points to the screen.

Notice the 'save' (cursor left) and 'retrieve' (cursor up) commands involve shifting—hopefully this will help avoid accidental typing of these commands.

Annotations. Here are brief descriptions of major elements of this program.

variables

LX, MX, LY, MY: least and maximum X and Y used.
X, Y, L: target X, Y, screen location.
GR$, R: drawing symbol graphic and reversal.
PG$, PR: previous graphic and reversal from location L.
SC=screen cursor address.
WH, BL, WT, FL: blink timers (white, black, wait, flash).
Character codes: SP=space; SS=shift space; CV=screen/ascii conversion factor; SH=shift bit; US=unshift mask; BY=byte mask; QT$=quote; CR=carriage return; RE=reverse; DE=delete; IR=initial reverse (OFF); XR=exchange reverse (XR–OFF=RVS).

line(s)

5 puts system into graphics mode (as opposed to lower case mode).
6-20 sets up constants, initializes variables; note that the first and last characters in strings H$ and V$ are cursor controls.
30 sets initial drawing symbol to a dot (shift-Q) and clears the screen.
40-60 prints dot in the center of the screen.
100 looks for keystroke.
150 converts keystroke to unshifted ascii.
200 checks that the target is blinked off.
250 RETURN causes a long blink.
300 resets short blink.
400-800 handles number key, graphic key, DEL, RVS and the cursor control keys.
850-875 the LEFT cursor causes a 'save' on tape; UP retrieves from tape.
1000 encodes symbol for display.
1200 resets drawing symbol.
1250 reverses drawing symbol.
1300-1500 displays symbol in target cell.
1700-2275 moves target in direction indicated by number key.
2300 locates new target.
2325 stores previous graphic.
2350 stores previous reverse information.
2400 checks for drawing symbol.
2500-2700 blinks the target if there's no symbol to draw.
3000-3020 blinks the target while you're not doing anything.
4000-4020 reverses the color of the target cell.
4500-4510 handles delete (DEL) key.
5000 initializes variables used in storing the picture on tape.
5010-5130 determines the size of the picture.
5500-5615 saves 3 lines of the picture in an array.
5618-5627 clears first 3 lines of the screen.
5630-5635 requests file name for the picture.
5640-5700 stores picture onto tape. Lines 5675-5677 turn the tape recorder motor on for 3 jiffies (3/60 second) and then off. This code is needed *only* for early PETs, which tend to have trouble reading data files. This technique should be used every 191 characters (1 record) or, as here, more often.
6000-6015 opens file, reads X1 and Y1 coordinates, checks status bit to see if end of file reached.
6020-6040 positions picture in center of screen.
6050-6100 reads and prints picture, closes file.


```
1 REM PET DRAWING PROGRAM
2 REM (c) 1977 PENINSULA SCHOOL
3 REM PERMISSION TO USE, BUT NOT SELL

5 POKE 59468.12
6 SC=32768: SP=32: SS=160: CV=191
7 BY=255: SH=128: US=127: QT$=CHR$(34)
8 LX=1: MX=33: LY=0: MY=24: DIM E$(2)
9 H$="[LEFT]"+CHR$(0)+"[RIGHT]"
10 V$="[UP]"+CHR$(0)+"[DOWN]"
15 N1$="1": N9$="9": CR=13: RE=18: DE=20
20 WH=50: BL=53: WT=5: XR=164: IR=146
30 GR$="[Q]": PRINT "[CLR]":
40 Y=INT((MY+LY)/2): X=INT((MX+LX)/2)
45 R=IR
50 PRINT "[HOME]":LEFT$("[12 DOWN]",Y):SPC(X):
60 GOTO 2300

100 GET C$: IF C$="" GOTO 3000
150 C=ASC(C$) AND US
200 IF FL>=WH THEN GOSUB 4000
250 IF C=CR THEN FL=-1E8: GOTO 100
300 FL=WH-WT
400 IF C$>=N1$ AND C$<=N9$ THEN 1700
450 IF C>=SP GOTO 1000
500 IF C=DE GOTO 4500
600 IF C=RE GOTO 1200
700 IF C$="[DOWN]" OR C$="[RIGHT]" THEN R=IR: GR$=""
750 IF C$="[HOME]" THEN GR$="": GOTO 40
800 IF C$="[CLR]" GOTO 30
850 IF C$="[LEFT]" GOTO 5000
875 IF C$="[UP]" GOTO 6000
900 GOTO 100

1000 GR$=CHR$(C+SH): R=IR
1100 GOTO 1300

1200 IF GR$="" THEN GR$=PG$: R=PR
1250 R=XR-R: PRINT CHR$(R);
1300 PRINT GR$:"[LEFT]":
1500 PG$=GR$: PR=R
1600 GOTO 100

1700 PX=X: PY=Y
1750 X=X+C+1-3*INT((C+2)/3)
1800 IF X<LX THEN X=LX
1900 IF X>MX THEN X=MX
2000 Y=Y+1-INT((C-49)/3)
2100 IF Y<LY THEN Y=LY
2200 IF Y>MY MY
2250 PRINT MID$(H$.X-PX+2.1):
2275 PRINT MID$(V$.Y-PY+2.1):
2300 L=SC+40*Y+X
2325 PG$=CHR$(PEEK(L) OR SH)
2350 PR=IR-(PEEK(L) AND SH)
2400 IF GR$<>"" GOTO 1300
2500 GOSUB 4000
2600 FOR DL=1 TO WT: NEXT DL
2700 GOSUB 4000
2800 GOTO 100

3000 FL=FL+1
3010 IF FL=WH THEN GOSUB 4000
3020 IF FL=BL THEN FL=0: GOSUB 4000
3030 GOTO 100

4000 PR=XR-PR: PRINT CHR$(PR):
4010 PRINT PG$: "[LEFT]":
4020 RETURN

4500 PG$=" ": PR=IR
4510 PRINT "[OFF, SPACE, LEFT]":CHR$(R):
4520 GOTO 100

5000 X0=MX: X1=LX: Y0=MY: Y1=LY: K=SC
5010 FOR Y=LY TO MY
5020 :FOR X=0 TO 39
5030 ::C=PEEK(K): K=K+1
5050 ::IF C=SP GOTO 5100
5060 ::IF X<X0 THEN X0=X
5070 ::IF X>X1 THEN X1=X
5080 ::IF Y<Y0 THEN Y0=Y
5090 ::IF Y>Y1 THEN Y1=Y
5100 :NEXT X
5130 NEXT Y

5500 SY=0
5510 FOR Y=Y0 TO Y1
5512 :K=SC+40*Y+X0
5515 :RV=0: L$=""
5520 :FOR X=X0 TO X1
5530 ::C=(PEEK(K)-SP AND CV)+SS
5535 ::V=C>BY: IF V=RV GOTO 5545
5540 ::RV=V: L$=L$+MID$("[RVS. OFF]".V+2.1)
5545 ::K=K+1: L$=L$+CHR$(SH*RV+C)
5600 :NEXT X
5615 :E$(SY)=L$: SY=SY+1
5618 :IF SY<3 AND Y<Y1 GOTO 5685
5620 :IF Y>Y0+2 GOTO 5660
5625 :PRINT "[HOME. OFF]":
5626 :FOR I=0 TO 2: FOR J=0 TO MX
5627 ::PRINT " ":: NEXT J: PRINT: NEXT I
5630 :INPUT "[HOME]SAVE FILE NAME": NM$
5635 :PRINT "[HOME]":
5640 :OPEN 1.1.1.NM$
5650 :PRINT#1. Y1+1-Y0
5655 :PRINT#1. X1+1-X0
5660 :FOR I=0 TO SY-1
5665 ::PRINT#1. QT$; E$(I); QT$
5670 ::E$(I)=""
5675 ::POKE 59411.53: T=TI
5676 ::IF TI-T<3 GOTO 5676
5677 ::POKE 59411.61
5678 :NEXT I
5680 :SY=0
5685 NEXT Y
5690 CLOSE 1
5700 GOTO 30

6000 OPEN 1
6005 INPUT#1. Y1
6010 INPUT#1. X1
6015 IF ST GOTO 6100
6020 Y0=INT((MY+LY+1-Y1)/2)+1
6030 X0=INT((MX+LX+1-X1)/2)+1
6040 PRINT MID$("[CLR. 12 DOWN]".1.Y0);
6050 FOR Y=1 TO Y1
6060 :INPUT#1.L$
6070 :PRINT SPC(X0):"[LEFT]";L$;
6080 :IF Y<=MY THEN PRINT
6090 NEXT Y
6100 CLOSE 1
6110 GR$=""
6120 GOTO 40
```

OTHER STUFF

We've made our first use of the 8-bit user port: we can flash a 40-watt lamp on and off under program control. The circuit we used is from *The First Book of Kim*, which is reviewed in this issue. The 115 volt AC circuit must be isolated from the 5 volt digital circuit. We used a $1.50 opto-isolator to do this, and a $1.40 Triac (electronic switch) to operate the lamp.

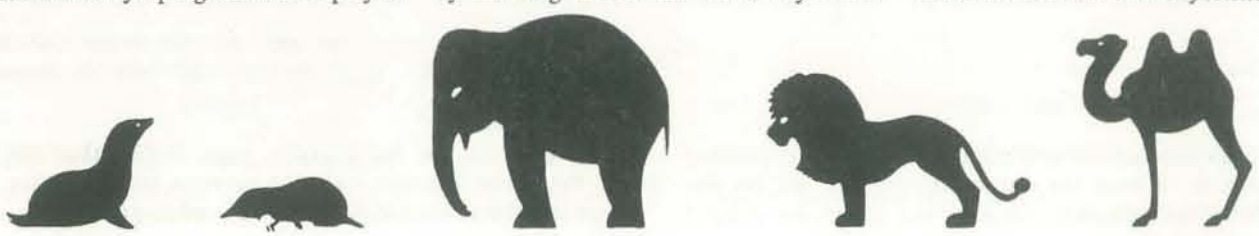
We've also used the PET to assist in analyzing data from a questionnaire. The program is designed so that inexperienced non-programmers can easily enter data, using one DATA statement per questionnaire. For easy reference, the DATA statement line number is the same as the sequential identification number assigned to each questionnaire. Checking the data after it was entered into the computer was facilitated by a program that displayed data for any specified questionnaire in an easy-to-read form.

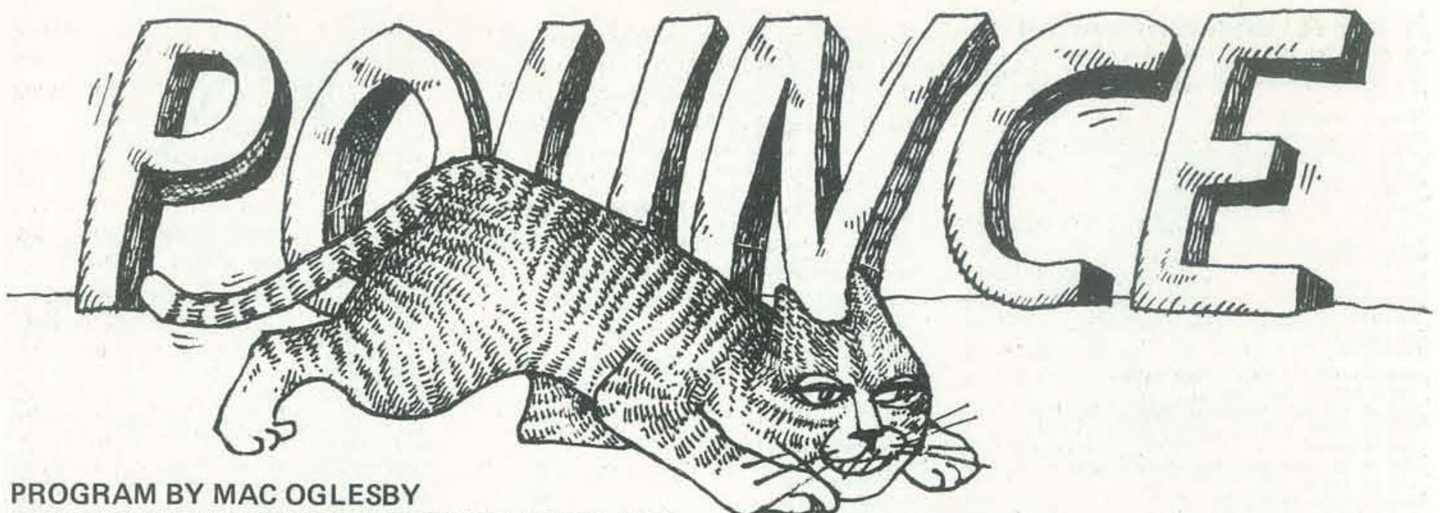
The PET and its associated questionnaire analysis programs have attended various meetings where survey results were discussed. To the delight of all concerned, questions such as 'What happens if we change factor X by amount Y?' and 'What's the median of Z?' were readily answered, and, when appropriate, the results were displayed as a graph. When the questions involved complex calculations whose results were not obvious, the computerized approach saved many hours of analysis and most likely forestalled many hours of discussion, since in a brief time period it was possible to explore many possible alternatives in depth.

A nice feature we've only recently discovered: you can list programs at a readable speed (about 2 lines per second) by holding down the RVS key while doing a listing. For more tips and another PET program, see the letters section of this issue.

We've seen draft versions of an introductory PET manual and a tutorial tape which introduces BASIC. Both are written for the novice. The tape's 10 lessons go into more detail than does the manual. I learned useful new information even with a brief glimpse at the tape—you don't need semi-colons to separate items in a PRINT statement—PRINT "HI, "N$ works fine. The PET User's Manual now being written promises to be the more complete work that many of us await. Commodore isn't even guessing when documentation will be available.

Last but not least, we hear that as of the first week of December Commodore was producing 100 4K PETs a week, and an unknown number of 8K systems.





PROGRAM BY MAC OGLESBY
DISCUSSION BY JOANNE KOLTNOW VERPLANK,
Community Computer Center

In this article we bring together two long-time supporters of People's Computers. *Mac Oglesby is a teacher in Vermont; from his time-sharing terminal in a log cabin he creates games for classroom use. We've published many of Mac's kid-tested games over the last few years.*

Joanne Verplank, the Director of Community Computer Center, has introduced thousands of kids to computers over the last few years. Here she shares with our readers her experiences with one of her favorite games, Mac's POUNCE.

We have to estimate and compare, visually, all the time – while driving, walking, or giving directions, in order to buy raw materials or decide which supermarket line is the shortest. However, visual skill building, which includes estimation and comparison, is often ignored in schools. POUNCE, an amusing chase game by Mac Oglesby, offers a chance for us to practice estimating and comparing short distances. POUNCE can be used by players at several levels of experience. Beginners easily learn to play, yet the game has enough variety to challenge the more advanced.

WHAT HAPPENS?

Players are shown representations of the cat and the mouse.

```
## = CAT
:: = MOUSE
```

When the game begins, the mouse is at the left margin and the cat is some distance away. The computer prompts POUNCE!!, waiting for a number to be typed.

```
::                    ##

POUNCE!!
```

The cat pounces toward the mouse, moving a distance corresponding to that number, in this case 4.

```
                    4

::                    ##

POUNCE!!
```

Several things can happen as a result of a pounce, depending on the relation between the size of the pounce and the distance between the animals.

1. If the size of the pounce matches the distance: The cat lands on the mouse and catches it, and the game ends.

```
:##

POUNCE!!
        1
*** YOU'VE CAUGHT THE MOUSE WITH 5 POUNCES!!
WANT TO CHASE ANOTHER MOUSE? YES
```

2. If the size of the pounce is smaller than the distance: The cat approaches the mouse, but doesn't get it.

```
:                                   #

POUNCE!!
          15

:                   #
```

3. If the size of the pounce is greater than the distance: The cat jumps over the mouse, landing on the other side. (In order for the play to remain on the paper, the whole frame of reference is shifted to the right.)

```
::::                                ####

POUNCE!!
          17

####                                ::::
```

(Since the cat always pounces toward the mouse, the player doesn't have to worry about direction when making pounce decisions.)

4. Sometimes, when the size of the pounce is *almost* the distance to the mouse, the mouse runs away.

```
POUNCE!!
         4

::::####

OH! OH! THE MOUSE SEES THE CAT!
  RUN, MOUSE, RUN!

####                                ::::
```

A small percentage of the time that the mouse runs, it will run into its hole. Then the game ends with the mouse the winner.

Two runs are on the opposite page. Notice that the sizes of the cat and mouse can vary between games. Notice, too, how the size of the cat affects the size of its pounce.

The game is easy, so almost anyone can play. Except for the few cases where a mouse runs into its hole (and this can happen to anyone) play continues until the cat catches the mouse. Also, the same noncommittal remark occurs at the end, no matter how many turns a player takes.

```
*** YOU'VE CAUGHT THE MOUSE WITH 1 POUNCE!!
WANT TO CHASE ANOTHER MOUSE? YES

     *** YOU'VE CAUGHT THE MOUSE WITH 9 POUNCES!!
     WANT TO CHASE ANOTHER MOUSE? YES
```

Since there is no turn limit, and players are not penalized for taking a long time, the game offers an encouraging situation in which people improve their estimation skills.

The few sentences used in the game are a parody of a children's reader. They're simple to read, so reading doesn't become a stumbling block; they're funny, so the simplicity doesn't insult anyone's intelligence.

While the game is easy to play, it's not trivial. Players have to understand the relationship between the number they type and the distance the cat moves. Since the distance depends on the size of the cat, which varies from game to game, the relationship is not immediately obvious. As players begin to relate the number they typed to the size of the cat and the distance it moved, they learn that a big cat takes big pounces and a small cat, small pounces. More practice enables them to estimate the size of the pounce needed to cover a particular distance. To do this, they have to compare the distance covered in earlier pounces with the remaining distance, and interpolate. They do this all visually, and often without saying anything.

I have seen players argue about their estimations, but never use measuring tools. Sometimes I point out relative distances, leading beginners into an understanding of the game. My conversation goes like this: 'If a pounce of 6 got us this far –, and a pounce of 2 got us this far –, how big a pounce do you think we need to get this far –?' We never do more than roughly indicate the distances, and that just for a moment.

The variety provided by the different sizes of the animals and the slightly different starting points offers a challenge to experienced players. As they become more expert in visual estimation, they need fewer samples of their cat's pounces and can judge accurately relatively greater distances. By the time they can catch the mouse in one or two pounces, they are quite skilled at estimating and comparing short distances, and they've had lots of fun.

A listing of POUNCE is on the next page. What next, Mac?

RUNS

```
### = CAT
::: = MOUSE

:::                                   ###

POUNCE!!
            24
TO POUNCE, JUST TYPE A WHOLE NUMBER FROM 1 TO 23.

POUNCE!!
         10

###:::

OH! OH! THE MOUSE SEES THE CAT!
  RUN, MOUSE, RUN!

:::                                               ###

POUNCE!!
         13

:::###

OH! OH! THE MOUSE SEES THE CAT!
  RUN, MOUSE, RUN!

###                    :::

POUNCE!!
         5

                    ###:::

POUNCE!!
         1
*** YOU'VE CAUGHT THE MOUSE WITH 4 POUNCES!!
WANT TO CHASE ANOTHER MOUSE? YES
```


```
# = CAT
: = MOUSE

:                                 #

POUNCE!!
         20

:              #

POUNCE!!
         8

:#

OH! OH! THE MOUSE SEES THE CAT!
  RUN, MOUSE, RUN!
LOOK! LOOK! THE MOUSE RAN INTO ITS HOLE!

WELL, THAT ONE GOT AWAY...
WANT TO CHASE ANOTHER MOUSE? YES
```

TRS-80

We've received our TRS-80, and written a review of the system. Unfortunately, our evaluation is quite negative. Our policy is to give manufacturers a chance to respond to negative comments before we publish them. Given holiday scheduling, our press deadlines are such that we must hold our review until our March-April issue. Meanwhile, we've sent a copy of the review to the people at Tandy Computers, offering to publish any timely and pertinent responses to the points raised in the review.

In addition, we encourage you readers to send in reports of any experiences you've had with the TRS-80.

Tom Williams
Assistant Editor

"It's a good poem, but try to put more feeling into it next time."

POUNCE

```
100 ' NAME:  ELEMLIB***:POUNCE
110 '
120 ' BY:  MAC OGLESBY ON 01/28/76.
130 '
140 ' DESCRIPTION:  LOOK! LOOK! SEE THE CAT JUMP OVER THE MOUSE?
150 '   POUNCE IS A GAME INVOLVING THE CONCEPTS OF SCALE AND
160 '   ESTIMATION BUT REQUIRING ONLY SIMPLE READING SKILLS.
170 '
180 ' INSTRUCTIONS:  TYPE "RUN" TO PLAY.
190 '
200 '
1000 LIBRARY "BASICLIB***:QUESTION"
1010 'DELETE QUESTION MARK
1020 CALL "QUESTION":0
1030
1040 RANDOMIZE
1050 LET A1=1                              'REPLAY RETURNS TO HERE
1060 LET C1=T1=0
1070 LET C$(1)=C$(2)=""
1080 LET S1=1+INT(RND*5)                   'SET THE SCALE (LENGTH OF CAT)
1090 LET N1=INT(70/S1)                     'MAX. ALLOWABLE INPUT FOR POUNCE
1100 LET L1=S1*(INT(20/S1)+INT(RND*(50/S1)))  'DISTANCE, CAT TO MOUSE
1110 FOR J1=1 TO S1                        'GENERATE CAT, MOUSE SYMBOLS
1120    LET C$(1)=C$(1)&":"
1130    LET C$(2)=C$(2)&"#"
1140 NEXT J1
1150
1160 'PRINT THE BOARD
1170 PRINT
1180 PRINT C$(2);" = CAT"
1190 PRINT C$(1);" = MOUSE"
1200 PRINT
1210 PRINT TAB(C1);C$(A1);TAB(C1+L1);C$(3-A1)
1220 PRINT
1230
1240 'HAS MOUSE SPOTTED CAT?
1250 IF L1>S1*(-3+INT(RND*8)) THEN 1380
1260    PRINT "OH! OH! THE MOUSE SEES THE CAT!"
1270    PRINT "  RUN, MOUSE, RUN!"
1280    IF RND>.2 THEN 1330
1290       PRINT "LOOK! LOOK! THE MOUSE RAN INTO ITS HOLE!"
1300       PRINT
1310       PRINT "WELL, THAT ONE GOT AWAY..."
1320       GOTO 1680
1330    LET L1=S1*(9-S1+INT(RND*(18-2*S1)))
1340    LET C1=0
1350    LET A1=3-A1
1360    GOTO 1200
1370
1380 PRINT "POUNCE!! ";CHR$(10);
1390 LINPUT A$
1400 IF (LEN(A$)-2)*(LEN(A$)-1)<>0 THEN 1820
1410    CHANGE A$ TO A
1420    FOR J1=1 TO A(0)
1430       'CHECK FOR DIGITS
1440       IF (57-A(J1))*(A(J1)-48)=>0 THEN 1460
1450          GOTO 1820
1460    NEXT J1
1470    IF A(0)=2 THEN 1500
1480       LET P1=A(1)-48
1490       GOTO 1520
1500    LET P1=10*(A(1)-48)+(A(2)-48)
1510    IF P1>N1 THEN 1820
1520       LET P1=P1*S1                    'POUNCE = INPUT X CAT'S LENGTH
1530       LET T1=T1+1                     'COUNT POUNCES
1540
1550       ON SGN(L1-P1)+2 GOTO 1580,1640,1770
1560
1570       'JUMPED OVER MOUSE
1580       LET L1=P1-L1
1590       LET C1=0
1600       LET A1=3-A1
1610       GOTO 1200
1620
1630       'CAUGHT MOUSE
1640       PRINT "*** YOU'VE CAUGHT THE MOUSE WITH";T1;"POUNCE";
1650       IF T1=1 THEN 1670
1660          PRINT "S";
1670       PRINT "!!"
1680       PRINT "WANT TO CHASE ANOTHER MOUSE? ";
1690       LINPUT A$
1700       LET A$=SEG$(A$,1,1)
1710       CHANGE A$ TO A
1720       IF (121-A(1))*(89-A(1))<>0 THEN 1740
1730          GOTO 1050
1740       STOP
1750
1760       'POUNCE FELL SHORT
1770       IF A1=1 THEN 1790
1780          LET C1=C1+P1
1790       LET L1=L1-P1
1800       GOTO 1200
1810
1820 PRINT "TO POUNCE, JUST TYPE A WHOLE NUMBER FROM 1 TO ";STR$(N1);"."
1830 PRINT
1840 GOTO 1380
1850
1860 END
```


CHECK OUT

A Reader's Report on COMPUTER STORES in the MINNEAPOLIS ··· AREA···

by Charles A. McCarthy

I checked out all the computer stores in the Minneapolis-St. Paul area listed in PCC's Reference Book. Here is what I found:

Microprogramming Inc
12033 Riverwood Dr, Burnsville MN
Has IMSAI, Poly, assorted other stuff, some books, a few mags. Primarily small business software development, and not hobby oriented. A very pleasant guy owns this (looks like one man and secretary) and I felt really bad about the 10 minutes he spent with me, for he clearly had a living to make and I suspect that his only capital was his time and smarts.

Byte Shop MN,
7547 Irish Ave, Cottage Grove MN 55016
This is a private home in an outer suburb, moderately affluent new development for people who want to appear on their way up. No sign of any store, and clearly strictly zoned against any commercial activity. I didn't feel it was worth stopping to investigate.

Byte Shop of Eagan,
1434 Yankee Doodle Rd, Eagan MN
A store in the next suburb over from Cottage Grove, in a small nest of a dozen of so small shops. Three visits made at various times of day. Always closed. Handwritten sign saying 'closed' covering the commercial 'closed' sign and the hours when open. Manufacturers' literature, back journals to about 2 months ago, a few books visible through store window.

Computer Depot Inc
3515 W 70th St, Edina MN 55435
A hobbyist store. A cut above Radio Shack, both in atmosphere and prices. Parts in bubble pack at full list price. Lots of books and mags. Half a dozen systems up, including Compucolor (purity mediocre, convergence poor), lots of memory board kits for sale, three guys in lab coats standing around talking to each other and knocking the competition (a good thing not to do when customers are around), and a goggle-eyed 12 year old kid who pushed me away from a terminal I was looking at so he could play Star Trek.

Computer Room St. Paul
3938 Beau D'Rue Dr, St. Paul MN 55122
Open for business, seems to handle only Altair, a few mags and books, no individual components for hardware hackers. No clerks. My contact was a very pleasant gal who was doing consumer software evaluation.

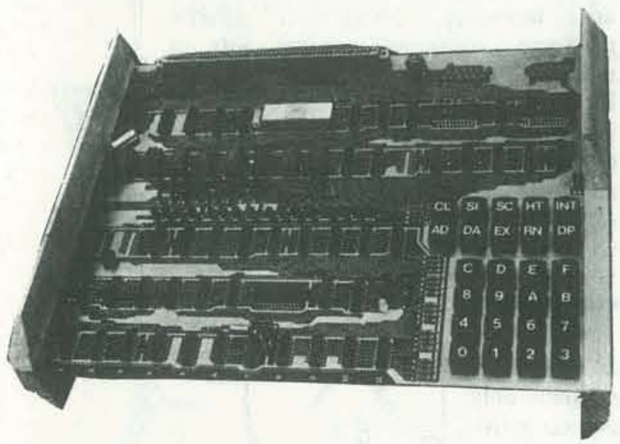
Computer Depot Inc.
1716 Midwest Plaza Bldg, Mpls MN 55402
A posh suite in one of the better downtown office buildings. Not for the hobbyist, but if you're a successful MD or lawyer or stockbroker with $50K who wants a fancy accounts receivable system to write off your tax bill, you'll be comfortable here. Tied in with Computer Depot in Edina. No equipment or literature. Looks like lawyer's office.

I also have two Radio Shacks near where I live. One is run by a reasonably nice guy (a ham) who doesn't know about computing machinery, nor anything about their competition, but is honest enough to say that he has no idea when anything will be available from them. The other store wanted $100 down against an unknown delivery date, even though he had *no* literature and admitted that he would not have a demo when they were available. Caveat emptor.

THE DATA HANDLER USERS MANUAL:

CONCLUSION

BY DON INMAN



The DATA HANDLER is a complete microcomputer system on a single PC board based on the MOS technology 6502 microprocessor. The DATA HANDLER can operate at very high speeds as a stand alone microcomputer or dedicated controller for even such high speed devices as disk peripherals. External TTYs or terminals are not needed since the DATA HANDLER contains 26 keyboard switches for full function hardware front control; personal expandability of the system is achieved by using the Altair/IMSAI peripherals on the DATA HANDLER PC board. The DATA HANDLER Bare Bones Kit which includes the DATA HANDLER PC board, PC board stand, 26 keyboard switches, and a complete documentation package is being offered at a price of $89.95. The complete kit is priced at $179.95. This includes the DATA HANDLER PC board, PC board stand, 26 keyboard switches, the complete set of IC's, 1 6502 MOS Technology microprocessor, sockets, LED's resistors, capacitors, 500 ns memory, and a complete documentation package. It is available through Western Data Systems, 3650 Charles Street, Suite G, Santa Clara CA 95050.

Don Inman is a former teacher, now editor of Calculators/Computers, *who's been working with teachers in the San Jose School District. Under Don's guidance, the teachers have built Data Handlers, complete microcomputer systems based on the 6502 microprocessor, and are now learning to use them. This is the seventh and last in a series of articles aimed at teaching relatively inexperienced people how to do assembly language programming for the 6502.*

This user's manual is designed to serve both as a self-teaching guide and as an outline for a course at the beginning level of computer science. While it deals specifically with the Data Handler, it can easily be adapted to other microcomputers using the MOS Technology 6502, such as the PET.

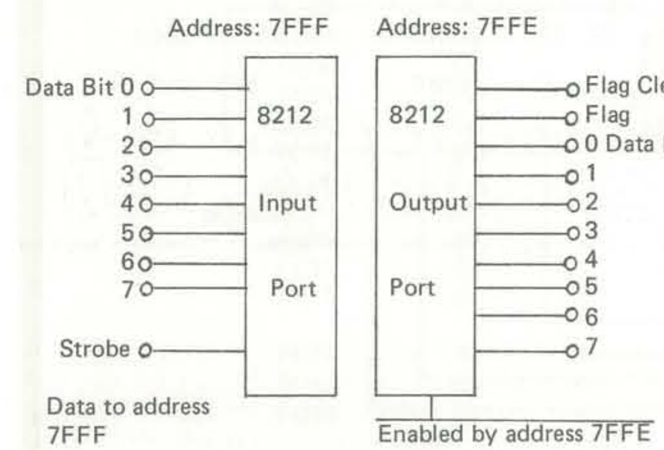
The course consists of nine two-hour class sessions, the first two of which were spent constructing the systems. Our series, Parts 1-7, covered class sessions 3-9. To recap,

Part	*Issue*	*Topic(s)*
1	*Vol 5 No 4*	*System specifications, binary and hexadecimal notation; checking out the system.*
2	*Vol 5 No 5*	*Data transfer.*
3	*Vol 5 No 6*	*The arithmetic unit.*
4	*Vol 6 No 1*	*Indexed addressing.*
5	*Vol 6 No 2*	*Writing programs.*
6	*Vol 6 No 3*	*Programming for multiplication and division.*
7	*Vol 6 No 4*	*(Conclusion). Simple and inexpensive output devices.*

SESSION IX – MEMORY USED AS INPUT/OUTPUT

Memory is used for two different primary purposes in our computer. It is used for the storage of instructions for our program. A second use is for storing data to be used by the program. As far as the memory is concerned, no distinction is made between an instruction and data. Both are merely 8-bit binary numbers. The programmer usually places his program in the lower portion of memory and data in higher numbered locations.

The Data Handler also makes use of memory for INPUT and OUTPUT. An 8-bit output port and an 8-bit input port are located on the right rear portion of the board. The output port is wired to memory location 7FFE, and the input port is wired to memory location 7FFF.



INPUT AND OUTPUT PORTS
Figure 1

The essential elements of the input and output ports are shown in Figure 1. These ports handle data in a parallel fashion. That is, all eight bits of data are presented to the port at one time. When an eight-bit data byte is ready for input, a strobe signal must be sent to the computer to tell it the data is ready. The computer then deposits the data into memory location 7FFF. Your program may then use the data in that location.

The output port works in reverse fashion. When data is deposited into memory location 7FFE, a signal is given at Flag and eight bits of data are available at the output port. If you wish to clear the output port, the Flag Clear bit is grounded temporarily.

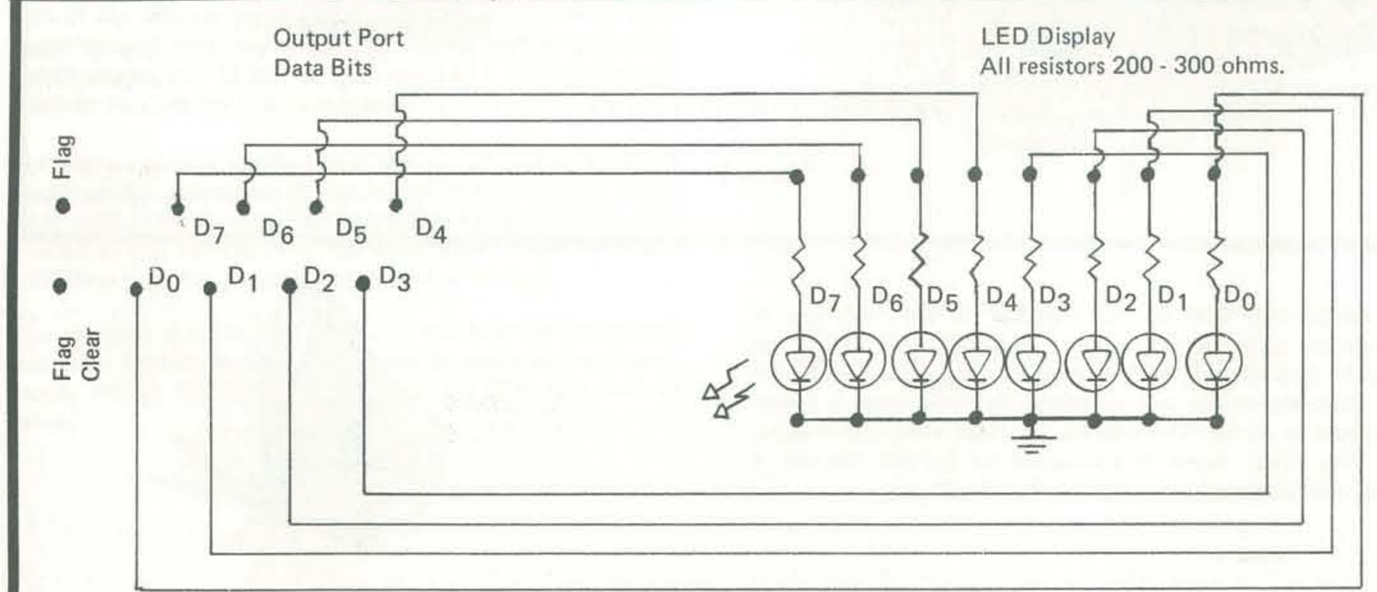
THE OUTPUT PORT

A typical use of the output port is shown in Figure 2. Discrete LED's are connected to show binary output. When the programmer wishes to output data, he stores the data in 7FFE with the sequence:

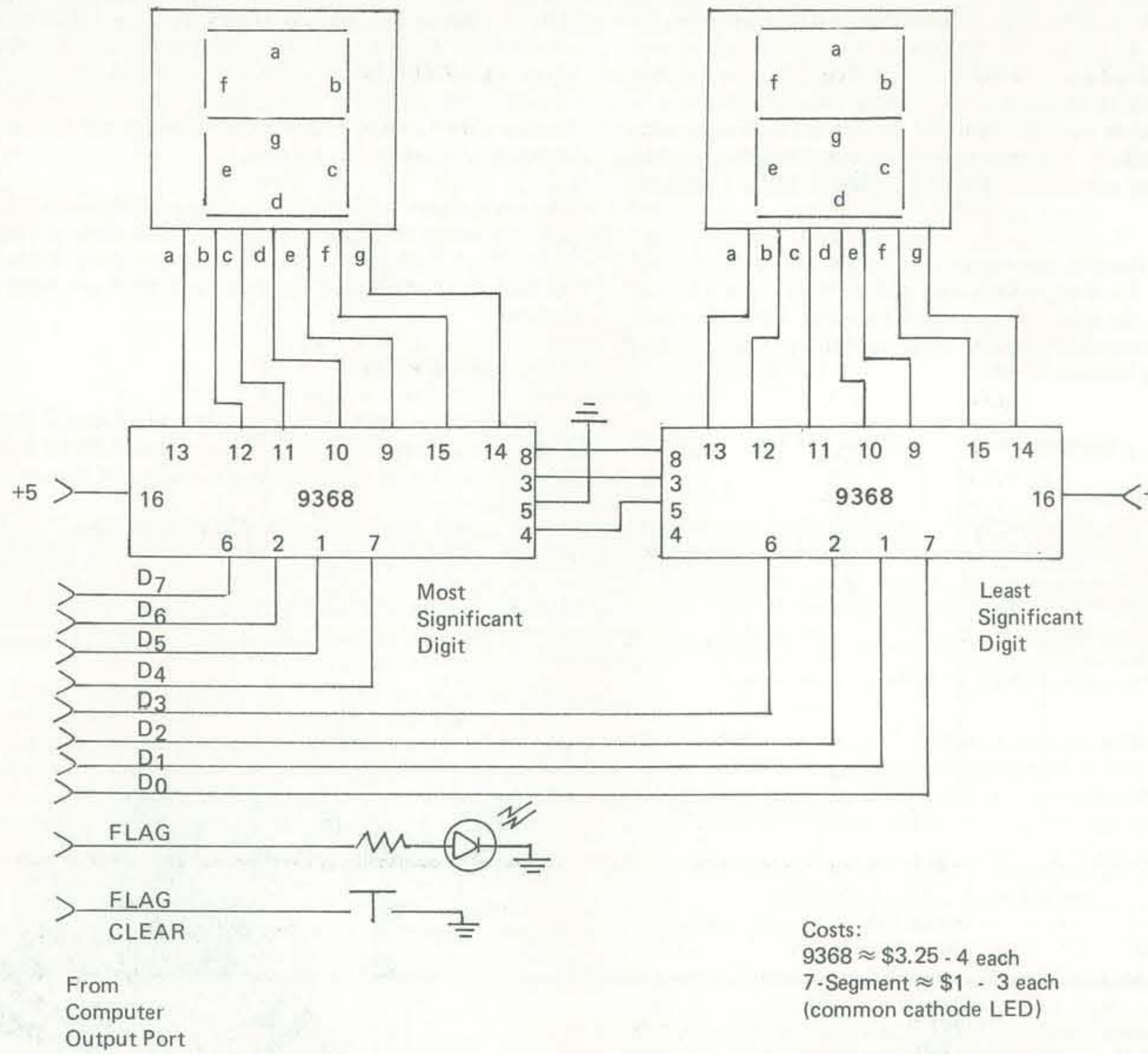
```
8D  STA  store accumulator in memory
FE
7F
```

The data would then be displayed in binary notation on the LED display

The next step up from the binary LED display would be a pair of 7-segment displays. Each one of the displays will output one hexadecimal digit. A pair of displays thus provides for the output of two hex digits, or one byte of data. In my original class where this material was first presented, we used a device from Hallbar, Inc. for the display. It was a kit consisting of a single 9368 latch decoder/driver integrated circuit and a 7-segment display with construction instructions and theory of operation. However this unit proved rather expensive. Buying parts separately would be cheaper than buying the kit, especially if you get the new CMOS version of the 9368.



ADDING A HEX OUTPUT DISPLAY
Figure 2



7-SEGMENT OUTPUT DISPLAY

Figure 3

The reader may wish to put together his own displays. A diagram for such a project is given in Figure 3. In addition to the eight data bits at the Data Handler output port, Flag and Flag Clear signals are also provided. The Flag Clear is shown connected to a LED. This tells the user that an output is there. The Flag Clear signal is connected to ground through a momentary push button switch to clear the display.

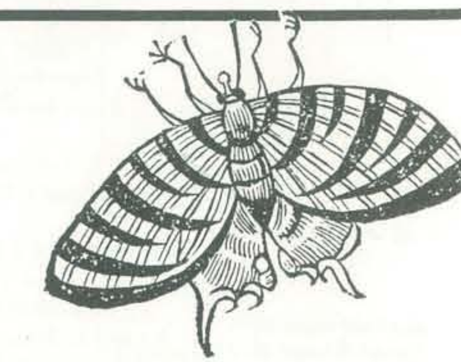
DEMONSTRATION PROGRAM TO EXERCISE AN OUTPUT DEVICE

LABEL	LOC	CODE	MNEM	COMMENTS	
ORG	FC00	D8	CLD	Clear decimal mode for hex count.	
	FC01	A9	LDA, 00	Start the count at zero.	
	FC02	00			
STOR	FC03	8D	STA, 7FFE	Output the count.	
	FC04	FE			
	FC05	7F			
	FC06	A0	LDY, 00	Initialize Y register for delay loop.	This group of steps is merely a time delay so that you can read the count.
	FC07	00			
	FC08	A2	LDX, 80	Initialize X register for delay loop.	
	FC09	80			
LOOP	FC0A	C8	INY	Start delay.	
	FC0B	D0	BNE, LOOP		
	FC0C	FD			
	FC0D	CA	DEX		
	FC0E	D0	BNE, LOOP		
	FC0F	FA			
	FC10	18	CLC	Get ready to count up 1.	
	FC11	69	ADC 01	Add 1 to accumulator count.	
	FC12	01			
	FC13	4C	JMP, STOR	Go back and output new count.	
	FC14	03			
	FC15	FC			

If additional output ports are available to you, additional displays can be added. Consult the diagrams in the Data Handler manual to see how the output port 7FFE is wired. A modification of the address select circuit is necessary to give an additional output port, say at 7FFD. Having two output ports would enable you to display two bytes of a double-precision result or two distinct one-byte results.

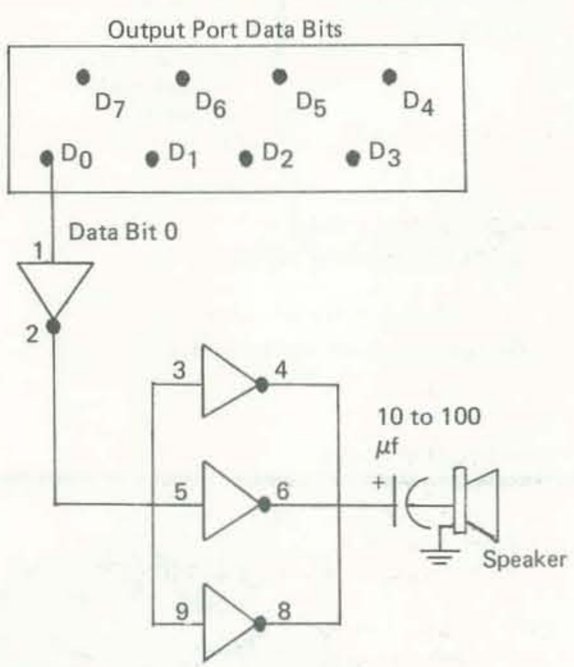
Let's now demonstrate the use of the output port by using the simple counting program shown below. A time delay is inserted at steps FC06 through FC0F so that the output will be slow enough so you can read the low order byte. Either the binary LED display or the 7-segment display may be used.

The program is an endless loop and will keep counting until you press the Halt button. If you want to count in the decimal mode, change the first instruction to F8 SED (Set decimal mode).



MUSIC FOR IDIOTS

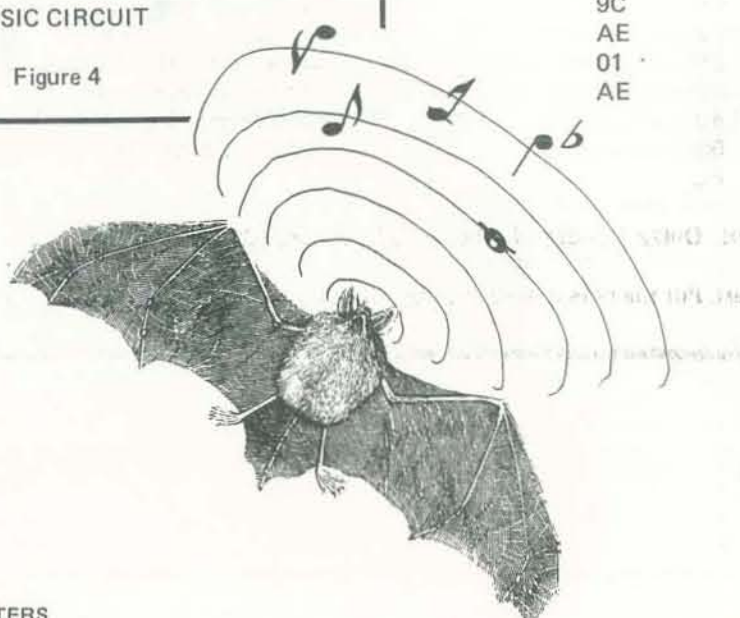
Now that you know how to use the ouput port, here's a program I credit to my 13 year old son. It uses one bit of the output port to vibrate the cone of a speaker producing music-like sounds. No attempt has been made to control the note length. The program merely represents an example of what can be done with a single output bit as a controller. The external circuit might be designed to drive or control any device. Here is the circuit we used. The music program follows. It was used as a demonstration in the class, but we found that the note data was suited to the timing of my particular Data Handler. You may have to experiment to find the correct values for your machine. Any data bit of the output port may be used, data bit 0 is used in the figure 4.



IC = 7404

MUSIC CIRCUIT

Figure 4



SAMPLE SONG FOR IDIOTS
TWINKLE, TWINKLE LITTLE STAR

LOC	DATA	LOC	DATA
FD00	01	FD28	BB
	E8		01
	01		BB
	E8		D2
	9C		D2
	01		E8
	9C		01
	8E		E8
FD08	01	FD30	9C
	8E		01
	9C		9C
	9C		8E
	AE		01
	01		8E
	AE	FD36	9C
	BB		9C
FD10	01	FD38	AE
	BB		01
	D2		AE
	01		BB
	D2		01
	E8		BB
	E8		D2
	9C		01
FD18	01	FD40	D2
	9C		E8
	AE		01
	01		E8
	AE		01
	BB	FD45	01
	01		
	BB		
FD20	D2		
	D2		
	9C		
	01		
	9C		
	AE		
	01		
	AE		

Load: 45 in FC27 :whatever you put in FC21 goes in FC60 also

THE IDIOT'S MUSIC PROGRAM

LABEL	LOC	CODE	MNEM	COMMENTS
BEGIN	FC00	A0	LDY	Counter 1.
	FC01	00		
	FC02	A9	LDA	Pop the speaker.
	FC03	01		
	FC04	8D	STA	through the output.
	FC05	FE		
	FC06	7F		
	FC07	BE	LDX, ABS + Y	Let's have the note.
	FC08	00		
	FC09	FD		
LOOP1	FC0A	CA		Decrement X for awhile.
	FC0B	D0		Stay in the loop 'til X goes to 0.
	FC0C	FD		
	FC0D	CE		Decrement the output.
	FC0E	FE		
	FC0F	7F		
	FC10	BE	LDX, ABS + Y	Grab the note again.
	FC11	00		
	FC12	FD		
LOOP2	FC13	CA	DEX	Decrement some more.
	FC14	D0		Don't branch till X is 0 again.
	FC15	FD		
	FC16	CE		Decrement the speed set at FC60.
	FC17	60		
	FC18	FC		
	FC19	A9	LDA	
	FC1A	00		
	FC1B	CD		Compare FC60 with 0.
	FC1C	60		If it's not zero,
	FC1D	FC		go back to Loop 1
	FC1F	E2		to delay some more.
	FC20	A9		Load the speed you'll use at FC60 (try 80).
	FC21	⟨ ⟩		
	FC22	8D		Store it back in FC60,
	FC23	60		you'll need it again.
	FC24	FC		
	FC25	C8		Increment Y for a new note.
	FC26	C0		Compare Y with
	FC27	⟨ ⟩		number of notes in song.
	FC28	D0		Branch if not 0 to Loop 1
	FC29	D8		and go around some more.
	FC2A	4C		Song's done. Shut off the
	FC2B	00		computer or you're going to hear it again.
	FC2C	FC		

(You know the music goes round & round)

Notes: (bottom to top) C-E8, D=D2, E=BB, F=AE, G-9C, A=8E, B=82, C=76, etc.

01 will give a rest. Put the note in twice to make it twice as long. Play around with it.

THE INPUT PORT

Although provision is made on the Data Handler printed circuit board to input from the keyboard, other devices can be connected through the 8-bit input port and memory location 7FFF.

Our first demonstration uses only one bit of the input port. It simulates an external signal from some control device. We will modify the program used for the output port (the counting program). Our input signal will determine whether the counting process displayed is done in the decimal mode or the hexadecimal mode. If our input is set to zero (ground on the switch in figure 5), the program counts in the hexadecimal mode as before. If our input is a 1 (plus 5 volts on the switch), the program counts in the decimal mode. The switch *must be set before* the program is run and the *data latched* by means of the *strobe signal*.

DEMONSTRATION OF INPUT/OUTPUT DEVICE

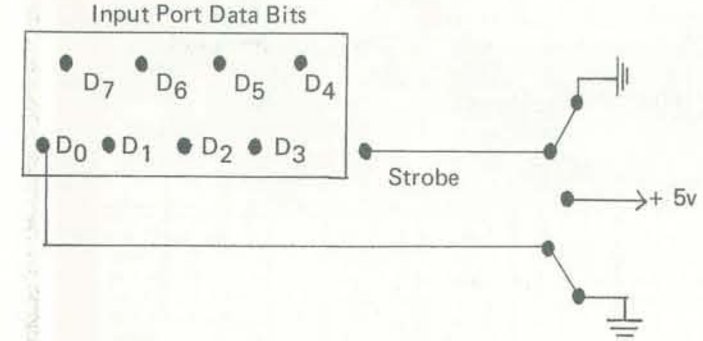
Equipment external to the computer:
2 7-segment common cathode displays
2 9368 hexadecimal latch/BCD to 7-segment decoder/driver IC's (as in figure 3) wired to output port
1 single pole, double throw switch to bit D_0 of input port (figure 5)

Operation: With input switch at zero the program outputs hex count.
With input switch at one the program outputs decimal count.
DON'T FORGET THE INPUT MUST BE STROBED IN TO LATCH IT BEFORE THE PROGRAM STARTS.

LABEL	LOC	CODE	MNEM	COMMENTS
ORG	FC00	D8	CLD	Originally set for hex mode.
	FC01	AD	LDA INPORT	Read the input signal.
	FC02	FF		
	FC03	7F		
	FC04	D0	BNE, DECI	If switch is not zero change to
	FC05	15		decimal mode.
	FC06	A9	LDA 00	Start at zero in counter (accumulator).
	FC07	00		
OUT	FC08	8D	STA OUTPORT	Display count.
	FC09	FE		
	FC0A	7F		
	FC0B	A0	LDY, 00	
	FC0C	00		
	FC0D	A2	LDX, 80	
	FC0E	80		Delay loop
LOOP	FC0F	C8	INY	
	FC10	D0	BNE LOOP	
	FC11	FD		
	FC12	CA	DEX	
	FC13	D0	BNE LOOP	
	FC14	FA		
COUNT	FC15	18	CLC	
	FC16	69	ADC 01	Count up 1.
	FC17	01		
	FC18	4C	JMP OUT	Output it.
	FC19	08		
	FC1A	FC		
DECI	FC1B	F8	SED	Set decimal mode.
	FC1C	4C	JMP, COUNT	
	FC1D	15		
	FC1E	FC		

DEMONSTRATION OF LOADING FROM THE INPUT PORT

LABEL	LOC	CODE	MNEM	COMMENTS
ORG	FC00	A2	LDX, 00	
	FC01	00		
LOAD	FC02	AD	LDA INPORT	Load the data byte.
	FC03	FF		
	FC04	7F		
	FC05	9D	STA FF00+X	Store it in memory.
	FC06	00		
	FC07	FF		
	FC08	E8	INX	
	FC09	E0	CPX,<>	Compare value in X with
	FC0A	<>		number of entries.
	FC0B	F0	BEQ MAIN	Branch if same to main program.
	FC0C	08		
	FC0D	A9	LDA, 02	Load accumulator with 02.
	FC0E	02		
	FC0F	8D	STA FCFF	Store in low-order address
	FC10	FC		of initialization vector.
	FC11	FF		
	FC12	4C	JMP FC12	Loop here until Halted for
	FC13	12		another input.
	FC14	FC		
MAIN	FC15		Your main program would start here.	



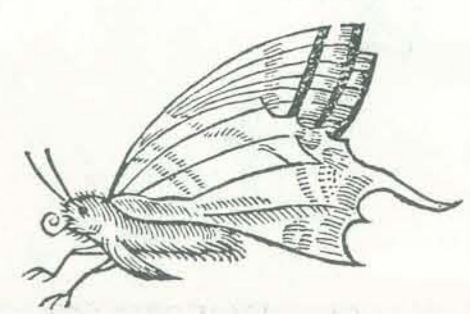
INPUT PORT

Figure 5

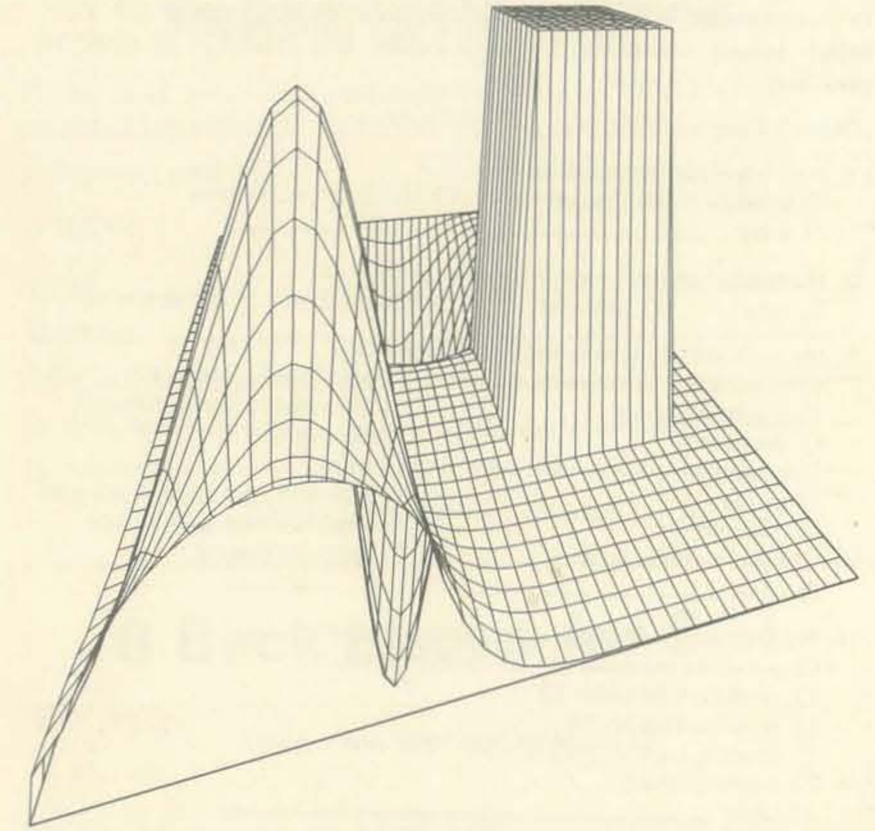
For our second, and last, demonstration we will use all eight data bits of the input port. They are wired in the same manner as the D_0 switch in Figure 5.

This demonstration uses a short program to load a block of numbers into memory from the input port. It could be used as a subroutine within a program which required loading of memory at various parts of the main program. The program assumes that when you have loaded the required data you will return to the main program at location FC15.

Eight bits of data are loaded from the switches by the strobe signal. The program stores the data in memory using the X register as an index. If all entries have been made the program jumps to the main program (location FC15). If all entries have not been made, the initialization vector is set to return you to the location for the next input (FC02). The program then loops until you push the Halt button. After pushing the Halt button, make your next entry and strobe it in. Then press the Start key again. This process is repeated until all your entries have been made. The program then automatically jumps to the main program.



This concludes this Data Handler series. Western Data Systems, 3650 Charles St Suite G, Santa Clara CA 95050, have been updating the Data Handler. Any future articles on the Data Handler depend upon how soon the revised version is available.



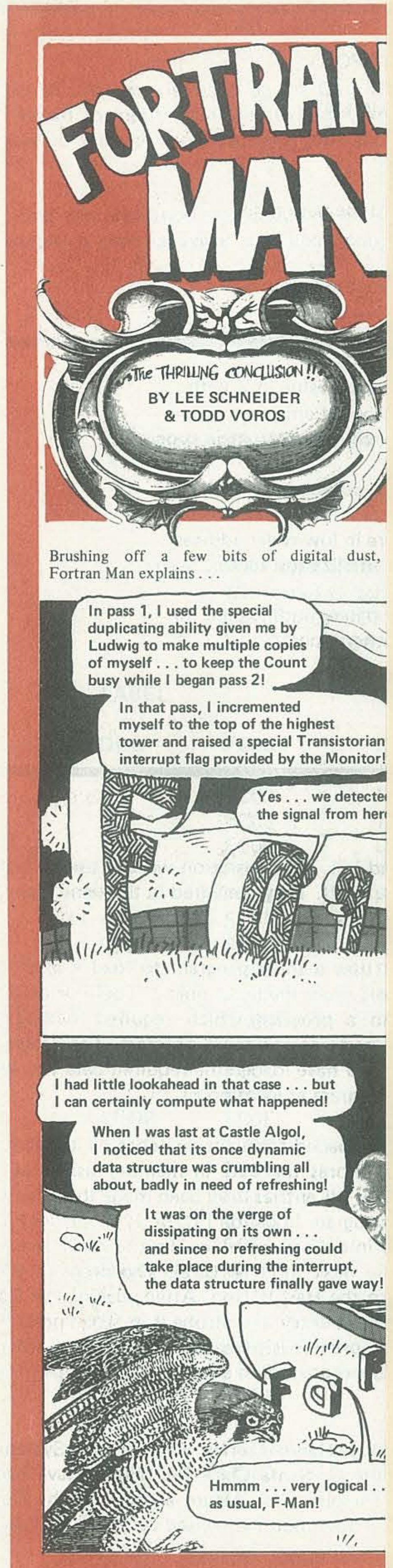
Fold here and tape

First Class
Permit No. 756
Menlo Park, Cal.

BUSINESS REPLY MAIL
No postage stamp necessary if mailed in the United States

Postage will be paid by

people's computers
1263 El Camino Real
Box E
Menlo Park, CA 94025



Reader Survey

In the past year we've made many changes in *People's Computers* in response to a survey of our readers. It's time again for your input---please complete the card below and return it as soon as possible.

People's Computers' Survey

1. I am interested in computers as a . . .
   - ☐ hobby ☐ educator ☐ computer professional
   - ☐ other ________
2. My level of computer experience is . . .
   - ☐ zero ☐ beginner ☐ intermediate ☐ extensive
3. My main interests are in using computers for . . .
   - ☐ commercial software development
   - ☐ communications
   - ☐ data bases
   - ☐ education
   - ☐ games
   - ☐ graphics, art
   - ☐ handicapped people
   - ☐ other ________
   - ☐ household automation
   - ☐ household records, finances
   - ☐ mailing lists
   - ☐ music
   - ☐ scientific applications, analyses
   - ☐ small business applications
   - ☐ word processing
4. My interests in using computers for education are . . .
   - ☐ providing materials for family and friends
   - ☐ teaching kids under 10
   - ☐ teaching kids 10-14
   - ☐ teaching high school students
   - ☐ teaching adults
   - ☐ other ________
   - ☐ non-existent
5. In future issues of *People's Computers* I would like to see more of:

   ____

   I would like to see less of:

   ____
6. I receive these computer publications:
   - ☐ *Byte* ☐ *Computer Music Journal* ☐ *Calculators/Computers*
   - ☐ *Computer* ☐ *Creative Computing* ☐ *Kilobaud*
   - ☐ *ROM* ☐ *Dr. Dobb's Journal* ☐ *Personal Computing*
   - ☐ other ________
7. Do you presently own a computer that works?
   - ☐ yes ☐ no
8. Do you plan to buy a home computer or additional home computer equipment within the next 6 months?
   - ☐ yes ☐ no
9. The highest level of education I've completed is:
   - ☐ junior high ☐ high school ☐ junior college
   - ☐ bachelor's ☐ master's ☐ Ph.D. or Professional
10. My age group is:
    - ☐ under 14 ☐ 14-18 ☐ 19-30 ☐ 31-50 ☐ over 50
11. I am ☐ male ☐ female
12. Are you a subscriber to *People's Computers?*
    - ☐ yes ☐ no

people's computers Subscription Form

Please send me a one-year subscription to *People's Computers* magazine (published bi-monthly) for $8.

☐ Payment enclosed ☐ This is a renewal (Please attach your label)

☐ Bill me

NAME ________

ADDRESS ________

CITY/STATE ________ ZIP ________

☐ Visa/BankAmericard Card No ________

☐ Master Charge Expiration Date ________

(Foreign rates available on page 2) 36

8 Back Issues for $6 !

(Order now. Supplies are limited)

For just $6 you can get 8 back issues from Volumes 4 and 5 in their original newspaper format. Buy our last three issues (new magazine format) for still just $1.50 each. An order card is below. Highlights of these issues include:

Soloworks Curriculum; Space Games Programs; Huntington Project Simulations; Kids Building Computers; Electronic Projects for Musicians; Classroom Computer Games; PILOT; Tiny PILOT; 6502 Assembly Programming; Pet 'Robots'; Games Programs.

(See back of card for magazine contents)

Back Issue Order

☐ Send me 8 back issues of *People's Computer Company's* newspaper from Volumes 4 and 5 for only $6.

☐ Send me these quantities of back issues for $1.50 each:

____ Vol 5, No 6 ____ Vol 6, No 2

____ Vol 6, No 1 ____ Vol 6, No 3

NAME ________

ADDRESS ________

CITY/STATE ________ ZIP ________

☐ Visa/BankAmericard Card No ________

☐ Master Charge Expiration Date ________

FIRST CLASS
PERMIT NO. 756
MENLO PARK, CA

BUSINESS REPLY MAIL
No Postage Stamp Necessary if Mailed in the United States

Postage will be paid by

people's computers
1263 El Camino Real
Box E
Menlo Park, CA 94025

Our most recent back issues in the new magazine format:

(Each issue contains one or more listings of games.)

Vol 6, No 1: Home Computers for Beginners; Z-80 PILOT; Tiny BASIC; 6502 Assembly Programming; Graphics; Women and Computers; game program.

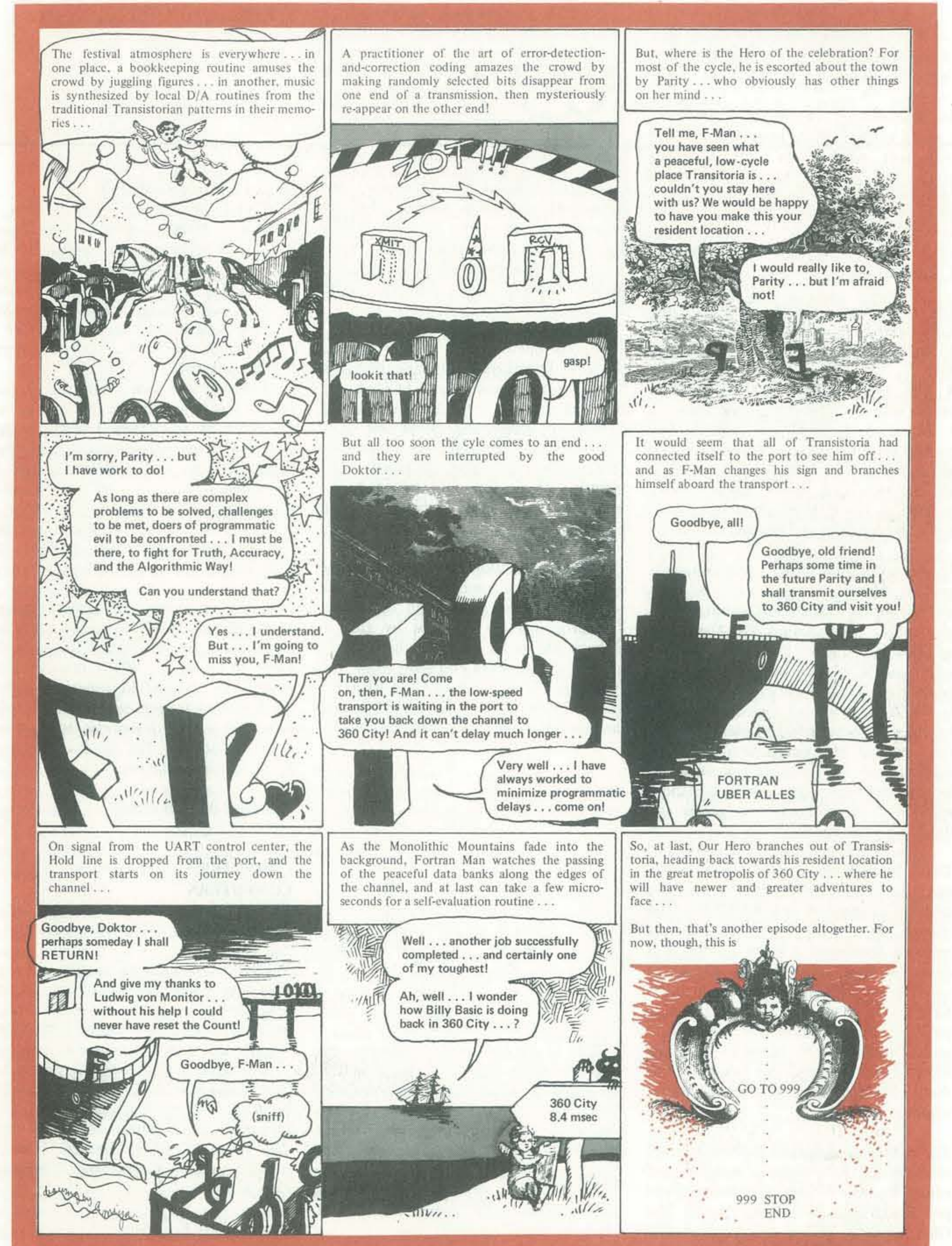
Vol 6, No 1: Heathkit Computers; Pet Robots; Do-it-yourself CAI in PILOT and BASIC; 6502 Assembly Programming; Kids' Computer Books; HP-25 Programming; game programs.

Vol 6, No 2: The PET; Tiny Languages; Computer Networks; Tiny PILOT Interpreter in BASIC; Microcomputers and Home Energy; game programs: mini-Kalah, Sandpile.

Vol 6, No 3: PET Evaluation; Teaching Math with Oz Graphics; Biofeedback and Microcomputers.

Mail to: people's computers back issues
1263 El Camino Real
Box E
Menlo Park, CA 94025

Please enclose payment with your order.



REVIEWS

THE FIRST BOOK OF KIM
Jim Butterfield, Stan Ockers,
Eric Rehnke, editors
ORB, PO Box 311, Argonne, IL 60439
176 pp, $9.00

The First Book of Kim, along with the KIM Programming Manual are a must for all KIM owners. I couple the two together since:

- The KIM Programming Manual comes with the KIM at purchase time.
- *The First Book of Kim* is predicated on the fact that you are a KIM owner. It's value is otherwise quite limited as almost all the programs in the book depend heavily on the KIM Monitor and use of the KIM keyboard and display.

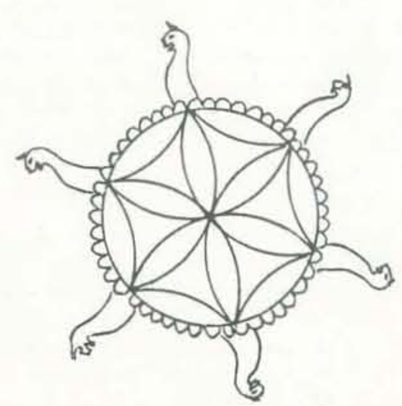
If you own some other 6502 computer and buy the book expecting to immediately make use of its games and programs, you will be disappointed. *The First Book of Kim* is intended for KIM users. No other use is implied by the authors or editors. As its title clearly states – this is a KIM book. Don't expect it to work miracles for other 6502 machines. If you can rework the games and programs contained in the book to run on a different machine, you are proficient enough to write your own games and don't need the book.

That's enough about what this book is not. Let's hear about what it is. As I said for openers, it is a MUST for KIM owners – especially for those who have recently purchased a KIM. The book is dedicated to just such people. If you are an old-time KIM user and belong to the KIM/6502 Users Group, you have probably seen most of the games, programs, utilities, and hardware 'add on' hints. Even for you, this book puts all these 'goodies' in one place, neatly organized.

The individual programs in *TFBOK* were contributed by various authors and edited by stan *O*ckers, eric *R*ehnke, and jim *B*utterfield. Does that sound like ORB or is it just a coincidence? The heavy contributor is obviously Jim Butterfield. I highly suspect that all authors are members of the KIM-1 User's Group. The group's *User Notes* is a great newsletter full of useful information. (P.O. Box 33077, North Royalton, OH 44133)

The book opens with a section devoted to KIM beginners explaining memory, hexadecimal numbers, how to load, run and step through a program, and testing a program. Displaying values and using the keyboard through the monitor's subroutines are also included. The majority of programs contained in the book use the monitor's subroutines with great effect.

The second section, the heart of the book, contains some 90 pages of 28 well-documented games and diversions. The programs are arranged by title in alphabetical order so that they should be easy to find when you want them. The type is LARGE and EASY TO READ – another good feature. Of special importance is the complete documentation.



Section three contains useful utility programs for such things as relocating programs, mini-monitor and mini-disassembler, memory test, tape utilities, sort, etc. The utilities look impressive.

Two sections on expanding the KIM and interfacing it to the outside world complete the book.

Should you buy the book?

- If you own, or are going to own, a KIM, you MUST buy *The First Book of Kim.* The authors are dedicated KIM users and have intertwined every program in the book with the KIM-1 computer. As you read through it I get the feeling that the book is a part of the computer – a peripheral, but still a necessary part. The book is meant to be used.
- If you do not own a KIM and do not plan to buy one, you probably cannot use the book in its intended way. It may give you some programming ideas; but unless you know the KIM monitor, it will prove frustrating – all those goodies and no way to use them.
- If you don't own a KIM and you DO buy the book, YOU ARE GOING TO BUY A KIM just to use the book.

I like the idea of the cooperative approach of the authors and editors. I like the idea of publishing books designed for a specific piece of hardware. While this limits its immediate wide-spread appeal, it makes the book much more practical and useful than the generalities presented in many 'how to' books. This book is excellent for its specific purpose.

Reviewed by Don Inman.

COMPUTERS, COMPUTERS, COMPUTERS
D. Van Tassel, editor
Thomas Nelson, 1977, 192 pp, $6.95

This is a collection of stories and verse that have something to do with computers. The quality, while not spectacular, is good. It made a pleasant afternoon's reading and most of the pieces I had not read before. But I wish the editor had mentioned that 'That Dinkum Thinkum' by Robert Heinlein, is essentially a short excerpt from Heinlein's enjoyable book *The Moon is a Harsh Mistress.* I recommend you read *Computers, Computers, Computers.*

Reviewed by Eryk Vershen.

YOUR HOME COMPUTER
by James White
Dymax, 1977, 211 pp, $6.00

HOME COMPUTERS: 2^{10} QUESTIONS AND ANSWERS
Volumes I and II: HARDWARE and SOFTWARE
by Rich Didday
Dilithium Press, 1977
Vol I 175 pp, $7.95
Vol II 150 pp, $6.95

I've been teaching a class on microcomputers at McNeil Island penitentiary. We started out with no hardware and a class of students who knew nothing about electronics or computers. It would have been very difficult to make a success out of such a class without good books with illustrations and practical examples. These three books are just what we needed.

James White's book is written for the person who wants to know why he or she should buy a home computer, what one is and how to choose one. The primary emphasis of the book is on hardware with many photographs and specific examples of each type of hardware discussed, but there is also a brief introduction to programming, enough so that the reader is left with some understanding of what a program is and why it is important.

We found White's book to be a good introduction for the novice. The microcomputer field has a language of its own which the novice must learn, and White covers the basic concepts clearly and at a comfortably slow pace. He also includes a good listing of computer stores, clubs and publications which the novice will find helpful.

The microcomputer field is changing so rapidly that many of the discussions of specific hardware in White's book are already out of date, but don't let this put you off. No book can be up to date in this field – you'll have to read the magazines and newletters for the latest new product information, and White's book will give you enough background to understand much of what you'll read.

After reading White's book, my students had a general idea of what a microcomputer is and they were familiar with some of the language of the computer field. Then we were ready to go into some detail, to examine the architecture of microcomputers and to learn how to program them. Didday's two volumes proved very helpful at this stage.

Didday's volumes were produced by editing the transcription of ten days of conversation about microcomputer hardware and software. There are lots of illustrations, sketches, practical examples and creative ideas. A few students found the conversational format odd at first, but everyone ended up liking these books.

Didday intended his books as an introduction for the novice, and he does define new technical terms as he uses them, but he has packed a lot of information into two volumes, so I think many will find his books most useful if an introductory book like White's is read first. To give you an idea of the ground he covers, here are some of his chapter headings: 'Numbers, logic and building blocks', 'Getting into hardware', 'What's it like to assemble a computer kit?', 'Some specific microprocessors', 'What's it really like to program in machine and assembly languages?', 'What's it like to program in Basic?', and 'What can you really do with it and what can't you do with it?'.

If you want to get started in microcomputers, reading these three books is a good beginning.

Reviewed by Tim Scully.

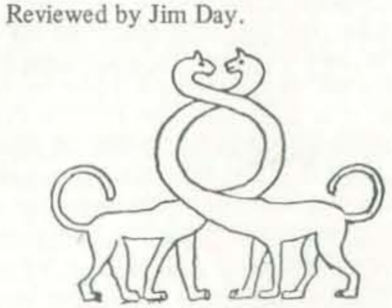
THE COMPUTER IN PSYCHOLOGY
Michael J. Apter, George Westby, editors
Wiley, 1973, 309 pp, $18.95 (hardbound)

This book was written by members of the department of psychology at University College, Cardiff. The first five chapters deal with the use of computers in the processing of psychological data, the on-line control of psychological experiments, and the modelling of behavior. Here the emphasis is on techniques rather than applications. The last five chapters describe some of the applications to which the techniques outlined in the first half of the book have been put. Further reading is suggested at the end of each chapter, and an extensive bibliography is presented following the last chapter.

Chapter 1 traces the development of calculating machines from primitive finger counting through modern computers. The second chapter is an introduction to programming. Chapter 3 examines the languages and software techniques available for the control of on-line experiments. Chapters 4-8 deal, respectively, with the use of computers in on-line experiments, modelling of behavior, study of the psychology of perception, and applications in the psychology of language. Chapter 9 reviews and gives examples of the use of computers in clinical psychology, not only for the storage and retrieval of clinical data, but for the automation of psychological testing and interviewing in general, and also the interpretation of clinical data. The techniques of Computer-Assisted Instruction are outlined in Chapter 10.

The book is very well written, easy to follow, and provides a good introduction to the applications of computers in the various fields of psychology. It also contains lots of ideas that could be turned into worthwhile projects for computer hobbyists.

Reviewed by Jim Day. □



COMAL: Structured BASIC

BY BØRGE R CHRISTENSEN

Børge R. Christensen and his associates in Denmark are among the many people dissatisfied with BASIC, especially as a tool for teachers and students. So they did something about it: COMAL was the happy result.

It all started back in 1972 when we got a NOVA minicomputer here at the States Training College, Tonder, Denmark. We started writing BASIC programmes like they did at most schools where they were lucky enough to get a computer at that time. At first everything seemed just fine. BASIC is easy to learn, and both the students and I wrote a lot of small programmes – most of them with mathematical themes – and they ran irreproachably. Gradually the programs grew bigger and errors became more frequent. Very often I had to sit for quite some time to find out where a student had made a mistake, and it began to irritate me that I often found it difficult to read even relatively small programmes written in BASIC.

I found two main reasons for that: variable names are much too short to give any information about what they represent, and the many GOTO's make it difficult and time consuming to identify the different tasks of a programme. Let's have a look at the following simple example:

```
0070 IF P<100 THEN GOTO 0100
0080 PRINT "YOU MUST PAY ";P;"DOLLAR."
0090 GOTO 0110
0100 PRINT "YOU MUST PAY (INCL. FEE) ";P+2;"DOLLAR"
0110 REM (*FEE OR NO FEE THAT WAS THE QUESTION*)
```

As you can see, there are two alternatives. If P (price) is less than 100, you have to pay a fee of 2 dollars to have your order executed. If the price is 100 dollars or more, you don't have to pay the fee. Now very often the GOTO in line 90 is forgotten, and the larger the programme module between the IF statement and the 'break point' grows, the greater the risk that it is forgotten. Also I find it extremely stupid that if the Boolean expression in the IF statement is *true* (e.g. in line 70 if P is less than 100), you have to go *somewhere else* (from line 70 to line 100) instead of executing the statement or statements immediately following it. Looking for the alternative immediately afterwards is the normal way of doing that kind of job. And why must price be represented by a P, or if you are generous P1, and not just PRICE? In large programmes with a lot of identifiers one is easily lost with the non-mnemonic names in a BASIC programme. Why can't an algorithm like the one above simply be stated like this:

```
0070 IF PRICE>=100 THEN DO
0080   PRINT "YOU MUST PAY ";PRICE;"DOLLAR"
0090 ELSE
0100   PRINT "YOU MUST PAY (INCL. FEE) ";PRICE+2;"DOLLAR"
0110 ENDIF  (*FEE OR NO FEE THAT WAS THE QUESTION*)
```

In the very fine book by Kerninghan and Plauger, *The Elements of Programming Style*, it says: 'Say what you mean, simply and directly' and 'choose variable names that won't be confused'. These two simple and fundamental rules of programming are impossible to apply with BASIC! On the other hand, there must be indisputable good things in BASIC, since it has become so popular and widespread. Personally, I would not be without the *interactive mode* and the *dynamic editor* of a BASIC system. Also *I/O statements* are easy to use and quite effective. And it certainly is *easy to learn*. I discussed the program with some of my colleagues at the Institute of Computer Science, University of Aarhus, and together with one of them, Benedict Løfstedt, I designed some extensions of BASIC in order to have more readable and safer programmes. We use the algorithmic structures from the programming language Pascal, defined by the Swiss professor Niklaus Wirth. Pascal is a language of the Algol family, but it is easier to use than Algol. I wanted our programming language to be an extension of BASIC for two reasons: existing BASIC programmes should still be running on our system, and – as mentioned above – there are things in BASIC we would like to use. After we had designed the extensions, two very talented students of mine, Knud Christensen and Per Christiansen, began to modify our BASIC interpreter (DGC's – Data General Corporation's – Extended BASIC) and in three months we had our first version running. We called it COMAL (*Co*mmon *A*lgorithmic *L*anguage). Some people think I should call it Structured BASIC. I don't care. Our project needed a title: we gave it the one above.

Now I won't tire you with a long theoretical explanation about Niklaus Wirth's and E. W. Dijkstra's 'structured programming' and 'algorithmic structures': instead I'll come right to the point and demonstrate our language and the principles of our extensions by means of an example. Please follow me. I shall from now on refer to the program listing, which appears on page 39. The programme has a heading, of course, with some remarks on title, author, time, etc. (10-130). The head also includes some definitions and declarations (70 - 130). In line 80 and line 130 you can trace the first extensions of BASIC. In COMAL you may use *up to 8 characters in an identifier name*. The first of these characters must be a letter, the following may be letters or digits. As mentioned above, this is one of the really important things, and it adds substantially to the readibility of a program. The variables TRUE and FALSE are used later on in Boolean expressions, and the two pointer functions are used for manipulation of strings. I'll explain it all when we are ready for it. From line 130 you can see that string variables are named according to the same rule as numeric variables and that a $-sign is added as in BASIC to identify the type. In LET statements you may have *as many assignments as the line width will take*, individual assignments being *separated by a semicolon*.

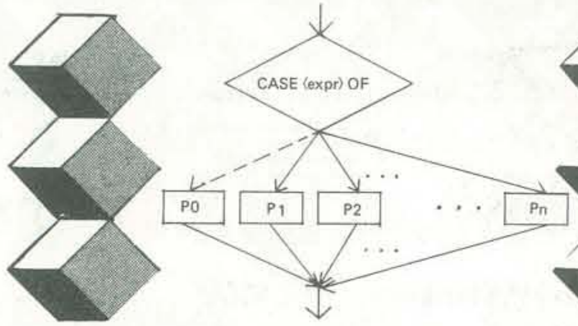
The next part of the program is the MONITOR (150 - 420). The body of the monitor is the interior of a REPEAT . . . UNTIL loop. I would like to explain the structure of the monitor first and then come back to REPEAT . . . UNTIL loops later on.

The monitor – and the whole program in fact – is controlled from the INPUT statement in line 180 and the CASE structure in lines 200 - 400. In this program, the CASE structure works like this: CODE$ (1,2) is evaluated, i.e. the substring consisting of the two first letters of CODE$, is picked out. The interpreter now looks at the associated WHEN statements (220, 250, 280, 310, 340, 370), to see whether the value (the substring) is found after a WHEN or not. If the operator has written, say SEARCH, the substring will be SE, which is found after the WHEN in line 280. Now the lines *between this WHEN and the following WHEN will be executed*. After that, the interpreter goes on with the statement following immediately after the ENDCASE. If the operator types STOP (or STO or even ST) after the request in 180, lines 380 -390 will of course be executed. If an illegal command, say SPT, is entered, the *alternative section*, which is the one *following immediately after the CASE statement*, is executed.

In general the CASE structure is described like this:

```
CASE expr OF
  [ P0 ]
WHEN list1
  [ P1 ]
WHEN list2
  [ P2 ]
. . .
WHEN listn
  [ Pn ]
ENDCASE
```

and works like this:



The *expr* (which as usual means a constant, a variable or an expression) is evaluated, and the interpreter starts looking for the value in the lists following the WHEN's. If it is found, the program section P_i between the actual WHEN and the following WHEN (or ENDCASE) is executed. If not found, the section P_0 between the CASE statement and the first WHEN statement is executed. After that execution continues with the statement following the ENDCASE.

The $list_i$ may include *as many items as the line width permits* (this facility is not used in the sample program). The list *may also include expressions* (arithmetic and Boolean), and if there are any, they will be *evaluated during the search*. This gives you some quite interesting possibilities. Just look at this. Further explanations should not be needed:

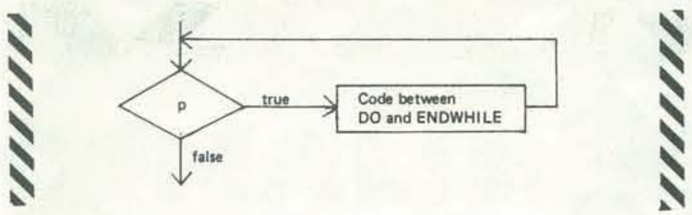
```
0040   INPUT "THE ELEMENT: ",X
0050   CASE TRUE OF
0060     PRINT "THE ELEMENT IS NOT IN ANY OF THE SETS"
0070   WHEN X=1,X=3,X=5,X=7,X=9
0080     PRINT "THE ELEMENT IS IN SET A"
0090   WHEN X>-10 AND X<=0
0100     PRINT "THE ELEMENT IS IN SET B"
0110   WHEN X=10,X=20,X=30,X=40,X=50,X>100
0120     PRINT "THE ELEMENT IS IN SET C"
0130   ENDCASE
```

Indented lines which emphasize the structure of the program are automatically supplied by the interpreter on the listing. I'll have more to say about that later. CASE structures may be nested to any depth.

In the monitor I've used the EXEC (execute) statement, too. In line 240 it says: EXEC INCODES. This is a *subroutine call*, and we may just as well go on at once to have a look at the subroutine or *procedure* that is called. It's in lines 440 - 550, and it begins with the statement PROC INCODES and ends with the statement ENDPROC INCODES. COMAL's PROC differs from BASIC's GOSUB in that a name is used instead of a statement number and the extent of the subroutine is clearly shown. Procedures may call new procedures until a depth of seven.

In lines 470 - 540 you find another COMAL structure, WHILE . . . ENDWHILE, which defines a loop. It's very simple: as long as CODE$ is different from 'NONE', lines 480-

530 will be repeatedly executed. The structure WHILE . . . ENDWHILE is shown in this flow-chart:

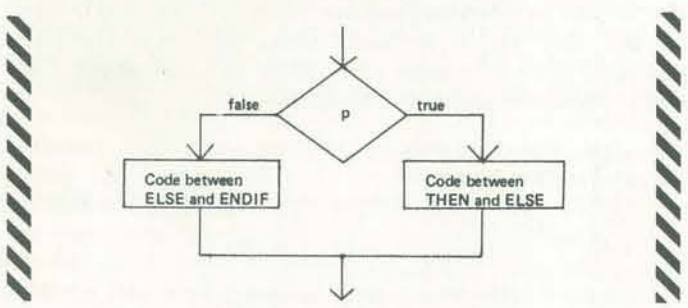


where p is a Boolean expression.

The IF . . . ENDIF structure is best demonstrated in PROC RUBOUT (780 - 920). Let's have a look at lines 830, 880, and 910. Here we find the keywords IF, ELSE and ENDIF. The whole thing is controlled by the statement:

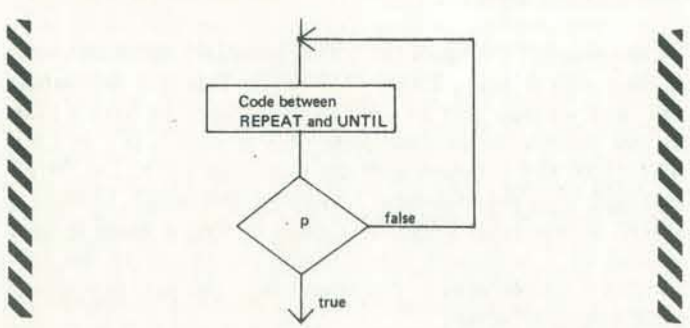
IF FOUND THEN

in 830. FOUND is a variable, which is used as a Boolean variable. It will be interpreted as *true*, if it has a *non-zero value*, and as *false*, if it has a *value of zero*. This is not a specific COMAL facility, but was already in the Extended BASIC from DGC. If it has the value *true*, lines 840 - 870 will be executed, and if it has the value *false*, lines 890 - 900 will be executed. The structure of an IF . . . ELSE . . . ENDIF branching is demonstrated in this flow-chart:



The expression p may of course be *any Boolean expression* (including Boolean constants and variables).

So far, I've not explained the REPEAT . . . UNTIL. Let's look at PROC HOUND (1080 - 1140). In lines 1090 and 1120 you'll find the REPEAT . . . UNTIL delimiters. The structure is the most self-explaining I know of, but nevertheless, here is the relevant flow-chart:



There are some interesting details in the body of the REPEAT . . . UNTIL loop. First look at the statement in line 1110. It says:

LET FOUND=(CODE$=MAIN$(FNA(I),FNB(I))).

We'll take it from the right to the left: MAIN$(FNA(I),FNB(I)) is the substring of MAIN$ including the characters pointed out by FNA(I) and FNB(I) and all characters in between. If I is equal to, say 6, FNA(I) is equal to 26 and FNB(I) is equal to 30. This is a trick we use to simulate an array of strings. We simply "cut" a string into pieces all of equal length (defined by the two pointer functions FNA and FNB). We expect to have string arrays implemented in COMAL by the beginning of 1978. We shall use the same conventions as in HP-3000 BASIC (*cf. People's Computers,* Sept - Oct 1977, page 58).

Well, back to our assignment. When the substring of MAIN$ has been picked out, it'll be compared to the value of CODE$, which is also a string. If it is the same string, the Boolean expression has a value of *false*. If the expression is *true*, FOUND is assigned a value of 1, and if the expression is *false*, FOUND is assigned a value of 0. According to the conventions mentioned above this will work whenever FOUND is used in a test somewhere. As you can see, we have a nice piece of Boolean algebra in COMAL, but I must admit that it is not used very much by the students yet. The REPEAT . . . UNTIL in 1090 - 1120 might just as well have terminated with:
UNTIL CODE$=MAIN$(FNA(I),FNB(I)) OR I=MAX.
Most students would do it like that, but then they would have to add the statement:
IF CODE$=MAIN$(FNA(I),FNB(I))
THEN LET FOUND=TRUE

IF . . . ELSE . . . ENDIF, WHILE . . . ENDWHILE . . . and REPEAT . . . UNTIL may each and independently of each other be nested to a depth of seven. So if you use them all together with good old FOR . . . NEXT, you may go down to a depth of twenty-eight. So far, I have never seen that done.

Any program written in DGC Extended BASIC may be run by our COMAL interpreter. This means that our library and all the BASIC programs we might get from other sources can be used with little change or no changes at all. And if you have to change them, it's not because of COMAL but because of BASIC. As we all know, BASIC versions are not always compatible.

The indented lines supplied by the COMAL interpreter have proved to be of greater importance than we had foreseen. They work as a kind of 'global debugging' facility. Suppose you forget to close an IF-branch with ENDIF. You can see immediately from the listing that something is wrong, since the statements do not 'close up' at the end of the listing. Also, and this is very important, students seem to become much more conscious about structure, when they see it the COMAL-way.

We were also happy to learn that our structures are very useful with computer-assisted instruction (CAI). We didn't plan it that way, but they are. We were more concerned with the algorithmic and problem solving points of view when we designed COMAL, but it appears that in particular the CASE structure is extensively used by our colleagues who work with CAI.

We've been using COMAL for almost two years now; I often wonder why so many people are still satisfied with BASIC. BASIC *was* OK back in 1967, but that was 10 years ago! And just look at the development of hardware since then. The mini

A COMAL PROGRAM

(A run is on the next page)

```
0010 REM (*SIMULATOR: FILE OF ARTICLES*)
0020 REM (*WRITTEN FOR 'PEOPLES COMPUTER'*)
0030 REM (*BY BØRGE R. CHRISTENSEN AT 'DATO', TONDER, DENMARK*)
0040 REM (*DATE OF THIS VERSION: OCT. 8. 1977*)
0050 REM (*LANGUAGE: COMAL 77 - RUN BY NOVA 1200*)
0060 REM //---------------//
0070 REM (*TWO BOOLEAN CONSTANTS: TRUE AND FALSE ARE DEFINED*)
0080 LET TRUE=1; FALSE=0
0090 REM (*TWO POINTERFUNCTIONS: FNA AND FNB ARE DEFINED*)
0100 DEF FNA(X)=5*X-4
0110 DEF FNB(X)=5*X
0120 REM (*MAINSTRING AND BUFFERSTRING ARE DECLARED*)
0130 DIM MAIN$(500),CODE$(5)
0140 REM //---------------//
0150 REM (*MONITOR*)
0160 LET MAX=0
0170 REPEAT (*FILE IN USE*)
0180   INPUT "ENTER, SURVEY, SEARCH, DELETE, SORT, STOP: ",CODE$
0190   PRINT
0200   CASE CODE$(1,2) OF
0210     PRINT "NO SUCH TASK - YOU!!"
0220   WHEN "EN"
0230     REM (*ENTER CODES*)
0240     EXEC INCODES
0250   WHEN "SU"
0260     REM (*PRINTOUT OF SURVEY*)
0270     EXEC SURVEY
0280   WHEN "SE"
0290     REM (*LOOK FOR A GIVEN CODE*)
0300     EXEC GETCODE
0310   WHEN "DE"
0320     REM (*DELETE A CODE*)
0330     EXEC RUBOUT
0340   WHEN "SO"
0350     REM (*ALPHANUMERIC SORTING OF FILE*)
0360     EXEC ALFASORT
0370   WHEN "ST"
0380     REM (*THAT'S ALL FOR TODAY, FOLKS*)
0390     STOP
0400   ENDCASE
0410 UNTIL FALSE
0420 END OF MONITOR
0430 REM //---------------//
0440 PROC INCODES
0450   REM (*TYPE IN NEW CODES*)
0460   INPUT "> ",CODE$
0470   WHILE CODE$<>"NONE" DO
0480     EXEC ANALYZE
0490     IF OK THEN
0500       LET MAIN$=MAIN$,CODE$
0510       LET MAX=MAX+1
0520     ENDIF (*IF CODE OK, THEN IT HAS NOW BEEN ENTERED*)
0530     INPUT "> ",CODE$
0540   ENDWHILE
0550 ENDPROC INCODES
0560 REM //---------------//
0570 PROC SURVEY
0580   PRINT
0590   PRINT "HERE IS YOUR LIST:"
0600   PRINT
0610   FOR I=1 TO MAX
0620     PRINT MAIN$(FNA(I),FNB(I))
0630   NEXT I
0640   PRINT
0650 ENDPROC SURVEY
0660 REM //---------------//
0670 PROC GETCODE
0680   LET I=0
0690   INPUT "WHICH CODE? ",CODE$
0700   EXEC HOUND
0710   IF FOUND THEN
0720     PRINT "THE WANTED CODE HAS NO.";ADR;"IN THE FILE."
0730   ELSE
0740     PRINT "NO SUCH CODE IN YOUR FILE!"
0750   ENDIF (*CODE OR NO CODE - THAT WAS THE QUESTION*)
0760 ENDPROC GETCODE
0770 REM //---------------//
0780 PROC RUBOUT
0790   INPUT "WHICH CODE IS GOING? ",CODE$
0800   LET I=0
0810   EXEC HOUND
0820   LET LAST=LEN(MAIN$)
0830   IF FOUND THEN
0840     REM (*DELETE THE CODE*)
0850     LET P1=FNA(ADR)-1; P2=FNB(ADR)+1
0860     LET MAIN$=MAIN$(1,P1),MAIN$(P2,LAST)
0870     LET MAX=MAX-1
0880   ELSE
0890     PRINT "NO SUCH CODE IN YOUR FILE!"
0900     PRINT
0910   ENDIF (*CODE DELETED OR NOT FOUND*)
0920 ENDPROC RUBOUT
0930 REM //---------------//
0940 PROC ALFASORT
0950   FOR I=1 TO MAX-1
0960     FOR J=I+1 TO MAX
0970       REM (*IF THE I'TH CODE COMES AFTER THE J'TH CODE*)
0980       IF MAIN$(FNA(I),FNB(I))>MAIN$(FNA(J),FNB(J)) THEN
0990         REM (*SWAP THE TWO CODES*)
1000         LET CODE$=MAIN$(FNA(I),FNB(I))
1010         LET MAIN$(FNA(I),FNB(I))=MAIN$(FNA(J),FNB(J))
1020         LET MAIN$(FNA(J),FNB(J))=CODE$
1030       ENDIF (*SWAPPING DONE*)
1040     NEXT J
1050   NEXT I
1060 ENDPROC ALFASORT
1070 REM //---------------//
1080 PROC HOUND
1090   REPEAT (*LOOK FOR CODE, UNTIL FOUND OR NO MORE CODES*)
1100     LET I=I+1
1110     LET FOUND=(CODE$=MAIN$(FNA(I),FNB(I)))
1120   UNTIL FOUND OR I=MAX
1130   LET ADR=I
1140 ENDPROC HOUND
1150 REM //---------------//
1160 PROC ANALYZE
1170   LET OK=TRUE
1180   FOR I=1 TO 3
1190     IF CODE$(I)<"A" OR "Z"<CODE$(I) THEN LET OK=FALSE
1200   NEXT I
1210   FOR I=4 TO 5
1220     IF CODE$(I)<"0" OR "9"<CODE$(I) THEN LET OK=FALSE
1230   NEXT I
1240   IF NOT OK THEN PRINT "CODE ILLEGAL. IS NOT REGISTRED!"
1250 ENDPROC ANALYZE
1260 REM //---------------//
```

was just barely designed in 1967. Through the works of Dijkstra, Wirth, Hoare and others, we know much more about good programming languages now. Why is this knowledge not used? It is my firm belief that most computers are underutilized due to insufficient software. And that goes especially for educational systems. Are we too easy for the computer dealers?

By the way, a COMAL interpreter is not a huge affair as you might think. We only had to add about 10 - 12% to the BASIC interpreter to have COMAL running, and in the very near future we hope to implement COMAL on a micro. With the new micros we have a chance to have computers running even in small schools; we expect that soon a lot of children will be working with them. Are they going to have computers from 1977 with software based on principles from 1957? Would you like to go to work every day in a car from 1910? Honestly? And not just for fun? Using developmental speed as the measure, it would be about the same as using a 1977 computer with 1957 software. □

RUN OF THE COMAL PROGRAM

```
ENTER, SURVEY, SEARCH, DELETE, SORT, STOP: EN

> AMV45
> QTH34
> RTY34
> UUW34
> TWN56
> BSA77
> BMW56
> NONE
ENTER, SURVEY, SEARCH, DELETE, SORT, STOP: SU

HERE IS YOUR LIST:

AMV45
QTH34
RTY34
UUW34
TWN56
BSA77
BMW56

ENTER, SURVEY, SEARCH, DELETE, SORT, STOP: SO

ENTER, SURVEY, SEARCH, DELETE, SORT, STOP: SU

HERE IS YOUR LIST:

AMV45
BMW56
BSA77
QTH34
RTY34
TWN56
UUW34

ENTER, SURVEY, SEARCH, DELETE, SORT, STOP: DE

WHICH CODE IS GOING? BSA77
ENTER, SURVEY, SEARCH, DELETE, SORT, STOP: SU

HERE IS YOUR LIST:

AMV45
BMW56
QTH34
RTY34
TWN56
UUW34

ENTER, SURVEY, SEARCH, DELETE, SORT, STOP: EN

> MMW45
> UUR67
> RTYU4
CODE ILLEGAL. IS NOT REGISTRED!
> NONE
ENTER, SURVEY, SEARCH, DELETE, SORT, STOP: SU

HERE IS YOUR LIST:

AMV45
BMW56
QTH34
RTY34
TWN56
UUW34
MMW45
UUR67

ENTER, SURVEY, SEARCH, DELETE, SORT, STOP: SO

ENTER, SURVEY, SEARCH, DELETE, SORT, STOP: SU

HERE IS YOUR LIST:

AMV45
BMW56
MMW45
QTH34
RTY34
TWN56
UUR67
UUW34

ENTER, SURVEY, SEARCH, DELETE, SORT, STOP: ST
```


PASCAL VS BASIC

A POLEMICAL COMPARISON OF THE TWO AS GENERAL-PURPOSE MICROPROCESSOR LANGUAGES

BY DAVID A MUNDIE

Many of our readers have expressed interest in learning more about Pascal. Here, David Mundie's discourse explores Pascal by contrasting it to BASIC. For readers with a limited programming background we recommend reading the Pascal listing on page 44 before beginning the article. The object of the Mastermind-like game is for the player to guess a randomly generated array of characters. The last 5 statements are the action part of the program; the preceding statements are the definition part.

The formal definition of Pascal is contained in Pascal: User Manual and Report *by Kathleen Jensen and Niklaus Wirth, published by Springer-Verlag, Berlin 1974. Ken Bowles'* Introduction to Computer Science, *published by Springer-Verlag in late 1977, uses Pascal as its teaching vehicle. Information on implementations of Pascal may be obtained through the* Pascal Newsletter, *available from Andy Mickel, University Computer Center, 227 Exp Engr, University of Minnesota, Minneapolis, MN 55455.*

If someone were to propose that the outdated Z-8888 CPU be retained in preference to the newer, faster, and more powerful 6868A, simply because everyone was already familiar with the older machine, his sanity would probably be questioned. Yet when it comes to the languages used on those machines, the personal computing community seems content to hobble along with a hopelessly inadequate language whose only excuse for existence is that it got there first. This contrast between our compulsiveness with regards to machines and our fetishism with regards to languages is surely one of the more interesting psychological aspects of the current computing scene.

Those with a vested interest in BASIC would have us believe that the situation is irreversible. I personally believe that the market for BASIC is just about saturated, and that if personal computing is to attain its full potential, the many marginally interested members of the general public will have to be won over with a language more suited to their needs than

BASIC. The casual user of the future will demand a language that is simple, powerful, and logical; I am hopeful that he will get it.

Be that as it may, now that microprocessor implementations of my favorite language, Pascal, are becoming available, I think the time has come to examine the two languages side by side, and what follows is my contribution to such a comparison. I am a prototypical applications programmer with no background in computer science, so I shall not attempt to propound the principles of structured programing on which, to a large extent, Pascal is based. Instead I shall concentrate on an actual BASIC program and its Pascal translation. I shall attempt to demonstrate that Pascal, because it offers an adequate repertoire of data types and control structures, allows the programmer to remain on a high level of algorithmic abstraction, where he functions best, whereas BASIC forces him to dirty his hands with improvised tricks and clumsy stopgaps. Pascal expresses algorithms clearly and simply, while BASIC does its best to obscure them altogether.

I could have made my task easier by examining those features of Pascal which have absolutely no equivalent in BASIC, and which by themselves would justify Pascal's adoption as a standard language: its recursive procedures, for example, or its superb data-structuring facilities. But a binary tree in BASIC, if feasible, would be a painful thing to behold, so I have restricted myself to only the simplest uses of the simples of Pascal's constructs. Furthermore, I could have chosen one of the illegible, amorphous BASIC programs which abound, because BASIC encourages sloppy thinking. Instead, I have chosen one which is carefully written. This is, then, a 'worst possible case' comparison as far as Pascal is concerned.

	BASIC	PASCAL
DATA TYPES		
integer	x	x
real	x	x
character (string)	x	x
boolean		x
defined scalar		x
STRUCTURING METHODS		
array	x	x
record		x
set		x
file		x
pointer		x
OPERATORS		
mixed arithmetic	x	x
integer division		x
modulus		x
exponentiation	x	
relational operators	x	x
set operators		x
logical operators		x
CONTROL STRUCTURES		
if	x	x
case		x
for	x	x
while		x
repeat		x
goto	x	x
SUBPROGRAMS		
recursion		x
local variables		x
parameters		x

CHART 1. Pascal vs. BASIC at a glance. The only feature BASIC has that Pascal doesn't is exponentiation!

The sample programs are presented in Listings 1 and 2. The functioning of the programs will be explained below under 'program structure'. It is not my intention to teach Pascal, but the following points may be helpful to the BASIC user seeing a Pascal program for the first time. **Readln(x)** and **writeln(x)** are roughly equivalent to INPUT X and PRINT X — they are mnemonics for 'READ a LiNe' and 'WRITE a LiNe' respectively. Statements are separated by semicolons. Arrays are indexed using square brackets. Type declarations are mandatory for all variables. **Chr** and **ord** are transfer functions; **chr(3)** returns the third element in the character set — usually the character 'c' — while **ord('c')** returns the integer 3.

I shall compare the programs on nine specific points.

1. IDENTIFIERS. I should think the most refreshing aspect of Pascal for a weary BASIC programmer would be its identifiers, which may be of any length. Thus the completely opaque 'T' and 'G' of the BASIC program become the much more transparent 'target' and 'guess'; this is absolutely essential if a program is to be readable as an algorithm. I shall not belabor the point, since I realize that BASIC is at last moving towards longer variable names, but it should be pointed out that in Pascal identifiers name not just variables, but also proce-

```
10   DIM F(9),G(9),T(9),H(18,3)
20   GOSUB 560
30   FOR X = 0 TO A
40   LET T(X) = INT(RND(R)*B)+1
50   NEXT X
60   FOR I = 1 TO A+B+1
70   FOR X = 0 TO A
80   LET F(X) = 0
90   NEXT X
100  LET F1 = 0
110  LET F2 = 0
120  INPUT V
130  IF V<>0 THEN 180
140  FOR X = 1 TO I-1
150  PRINT H(X,0);",";H(X,1);"=";H(X,2)
160  NEXT X
170  GO TO 120
180  IF V = 1 THEN 480
190  IF V = 2 THEN 670
200  LET T1 = V
210  FOR X = 0 TO A
220  LET G(X) = INT(T1/(10**(A-X)))
230  LET T1 = T1-G(X)*(10**(A-X))
240  IF G(X)<1 THEN 260
250  IF G(X)<B+1 THEN 280
260  PRINT "BAD NUMBER IN"; V
270  GO TO 70
280  IF G(X)<>T(X) THEN 310
290  LET F(X) = 1
300  LET F1 = F1+1
310  NEXT X
320  IF F1 = A+1 THEN 540
330  FOR Y = 0 TO A
340  IF T(Y) = G(Y) THEN 420
350  FOR X = 0 TO A
360  IF G(Y)<>T(X) THEN 410
370  IF F(X) = 1 THEN 410
380  LET F(X) = 1
390  LET F2 = F2+1
400  GO TO 420
410  NEXT X
420  NEXT Y
430  PRINT F1;",";F2
440  LET H(I,0) = F1
450  LET H(I,1) = F2
460  LET H(I,2) = V
470  NEXT I
480  LET V= 0
490  FOR X = 0 TO A
500  LET V = V+T(X)*(10**(A-X))
510  NEXT X
520  PRINT "ANSWER IS";V
530  GO TO 30
540  PRINT "YOU GUESSED IT"
550  GO TO 30
560  PRINT
570  PRINT "  DIGITS & MAX VALUE"
580  INPUT A,B
590  LET A = A-1
600  RETURN
610  END
```

Listing 1. This is a simple number-guessing game of the Mastermind type.

```
program banbasic(input,output);
const maxnumch=10; maxmax=71;
type token=packed array[1..maxnumch]of char;
var target,guess:token; hi,lo,ch:char; oldg:array 1..maxmax of token;
    i,j,try,maxtries,black,white,numchar:integer;
    oldb,oldw:packed array[1..maxmax]of integer;
    endofround,endofgame,wmatch,bad:boolean;
    matched:array[1..maxnumch]of boolean;
procedure newgame;
 begin endofgame:=false;
  for i:=1 to maxnumch do target[i]:=' '; guess:=target;
  writeln(' low character?'); readln(lo);
  writeln(' high character?'); readln(hi);
  writeln(' no. of characters?'); readln(numchar);
  maxtries:=numchar+ord(hi)-ord(lo)
 end;
procedure newround;
 begin endofround:=false; try:=0; for i:=1 to numchar do
  target[i]:=chr(ord(lo)+trunc(random(1)*(ord(hi)-ord(lo))))
 end;
procedure command;
  procedure tally(var i,color:integer);
   begin matched[i]:=true; color:=color+1 end;
 begin writeln(' command?'); readln(ch); case ch of
 'r':for i:=1 to try do writeln(oldb[i],'b',oldw[i],'w',oldg[i]);
 'q':begin writeln(' answer is: ',target); endofround:=true end;
 's': begin endofround:=true; endofgame:=true end;
 'c': begin i:=0; repeat i:=i+1; read(guess[i]);
      bad:=not(guess[i]in[lo..hi]) until (i=numchar)or(bad); readln;
      if bad then writeln(' bad character') else if guess=target then
      begin writeln(' you guessed it!'); endofround:=true
      end else if try=maxtries then
      begin writeln(' you are lost; answer is: ',target);endofround:=true
      end else
      begin black:=0; white:=0; try:=try+1;
       for i:=1 to numchar do matched[i]:=false;
       for i:=1 to numchar do if guess[i]=target[i]then tally(i,black);
       for i:=1 to numchar do if guess[i]≠target[i]then
       begin j:=0; repeat j:=j+1;
        wmatch:= (guess[i]=target[j]) and (not(matched[j]));
        if wmatch then tally(j,white) until (wmatch)or(j=numchar)
       end; writeln(' b',black,' w',white);
       oldg[try]:=guess; oldb[try]:=black; oldw[try]:=white
      end
     end
  end
 end;

begin newgame;
 repeat newround;
  repeat command until endofround
 until endofgame
end.
```

Listing 2. This is a free Pascal translation of listing 1, using arrays of characters rather than integers. Notice that the algorithm for the entire program is contained in just the last five lines of the listing. Following traditional Pascal practice, reserved words have been underlined.

dures, functions, constants, and types, so that, for example, 'GOSUB 560' becomes 'newgame'.

2. PROGRAM STRUCTURE. Another major difference which immediately strikes the eye is the difference in the structure of the two programs. The structure of the BASIC program is easy to describe: it hasn't any. It is a simple list of statements of equal value. It has been shown time and time again that this is bad news when it comes time to debug, modify, document, or just plain *understand* a program. In contrast, the Pascal program is broken down into a number of manageable parts which may be tested and understood separately. The algorithm for the program as a whole is found at the very end, in five succinct lines:

```
begin newgame;
  repeat newround;
    repeat command until endofround
  until endofgame
end.
```

It is difficult to imagine programming on a higher level of algorithmic abstraction. Even the complete programming novice could intuit that 'newgame' initializes certain game conditions, that 'newround' starts a new round, that a round consists of processing commands until the round is over, and that rounds are played until the game is over. Nothing in the BASIC program gives so much as a hint of what the overall pattern of the game is; instead, the eye wanders aimlessly over the page, desperately looking for a clue as to what is going on.

By looking at the declaration part of the Pascal program (broken down into **const** for constant, **var** for variable and **type** declarations), then at each of the program's subroutines, we can fill in that general algorithm with the details of its operation. The object of the game is for the player to guess an array of characters which the computer generates in the procedure newround using a random number generator. The procedure command prompts the player to enter one of four possible commands by typing one of the four letters 's', 'q', 'r', or 'c'. A S(top) command terminates the round and the game immediately. A Q(uit) command shows the player the answer and terminates the round, while a R(ecap) command shows the player his previous guesses and how well they scored, using the arrays oldg, oldb, and oldw. Lastly, a C(alculate) command reads in a guess from the player, and checks it for correctness; if the player has exceeded the limit on the number of tries (maxtries) or has guessed the target exactly, the round is terminated and a suitable message displayed, otherwise the guess is compared to the target and the player is told how many correct characters he had in the correct positions (black) and how many in incorrect positions (white). The procedure newgame, finally, specifies the number of characters in the target and guess (numchar), the range of acceptable characters (lo to hi), and the maximum number of tries allowed.

The movement from generality to detail which we have just seen is called 'stepwise refinement' by the theoreticians, and it makes a program easier to read because at any step one need only consider those details that are really necessary. Step-wise refinement is possible because Pascal is block-structured, which means that subprograms, like 'command' have the same structure as programs. Without going into detail, we may say that a block in Pascal consists of a definition part, in which constants, types, variables, and subprograms are defined, and an action part, which contains the algorithm of the block. Thus the five lines quoted above constitute the action part of the main program; everything else is its definition part.

3. DATA TYPES. Here's another example of how Pascal allows the programmer to think at a higher level of abstraction than can his BASIC counterpart. Consider the function of the array F in the BASIC program. The elements of this array are initialized to zero (line 80), then whenever a match between the target and the guess is found, the appropriate element of F is set to one. Later the array is used to test whether a given element has already been matched, by testing 'IF F(X) = 1' at line 370. From this it should be clear that on the *algorithmic* level, F(X) is not a *number* at all: it is a Boolean variable taking on the values *true* and *false*. Because BASIC lacks this data type, the programmer has had to leave the algorithmic level and make do with integers, with the result that the intent of his program has become seriously disguised. One's first expectation is that F(X) will *count* something, and only a painstaking examination of the entire program reveals its true function.

In contrast, Pascal provides a Boolean type, so that the programmer can remain on the more abstract level and simply write 'matched[i] := true', where the intention of the statement is perfectly transparent. An incidental advantage of this is that each element of 'matched' takes up just one bit of storage whereas each element in 'F' takes up at least a byte, probably more.

The type Boolean is one of Pascal's four predefined scalar data types, the others being integer, real, and character. In addition, it is possible to define new scalar (i.e. ordered) types by listing their values, as for example 'week = (mo, tu, we, th, fr, sa, su)', thus providing for extremely easy-to-read programs. These scalar types are the fundamental building blocks from which all structured types are derived.

4. STRUCTURING METHODS:
ARRAYS. Not only does BASIC fail to provide one of the most important basic data types, it also fails to provide adequate means of structuring the types it does have. To see this, let us look at the arrays 'T' and 'G'. From a strictly logical point of view, these are simply arrays of characters. But in trying to express this in his program, the BASIC programmer ran up against the fact that BASIC *has* no arrays of characters. Once again, a makeshift solution has been imposed on him: in this case, it is the 'integer' V which is read in and then 'unpacked' into its individual digits by the tortured routine in lines 220 - 230, then repacked in lines 490 - 510. This is bad not just because it is awkward and slow, but also because it completely obscures what is going on.

Now Pascal *also* lacks arrays of characters as a data type. But it does have the character as a basic data type, and the array as a basic structuring method, so that the definition of a type for the target is as simple as saying:

```
token: packed array [1..maxnumch]
  of char.
```

This done, the type token may be used just as any of the basic data types may be: in assignment statements (target:= guess), as parameters to subprogram calls (writeln(target)), in expressions (if target = guess) and so on. Two incidental advantages of the Pascal approach are that the player is not restricted to digits, and that the Pascal arrays take up only a fraction of the space occupied by their BASIC equivalents.

The newer kinds of BASIC with string variables may answer this *specific* objection, but not my central point, which is that in Pascal it is possible to define *any* data type built out of the basic units, whereas in BASIC you must take what you are offered.

This is perhaps the place to mention that requiring an array to start at 0 (or at 1), with the resulting awkwardness of line 590 where a variable is read in and then immediately decremented by one, seems barbaric to a Pascaler. In Pascal arrays may have any number of dimensions, and both bounds are chosen at will.

The array is only one of five basic structured types in Pascal. I shall discuss the type **set** below, but the other three (**pointer**, **file**, and **record**) are, as I mentioned above, so far beyond BASIC that discussing them would be pure malice on my part.

5. STRUCTURING METHODS: SETS.
For the sake of conciseness I have eliminated most error-checking from the two programs, but suppose we wanted to check that no illegal commands were entered. If the commands were letters, as in the Pascal program, checking them in BASIC would entail the absurd:

```
100 IF C$ = "R" THEN 220
110 IF C$ = "Q" THEN 220
120 IF C$ = "S" THEN 220
130 IF C$ = "C" THEN 220
140 PRINT "ILLEGAL INPUT"
```

where each value must be tested separately. Pascal takes care of this with the structured type **set**, whose utility goes far beyond this simple example, and which in fact allows set manipulation in all its generality. The above test in Pascal would be this simple:
if ch in ['r', 'q', 's', 'c'] then ...
where **in** is the relational operator for inclusion; program banbasic uses it to check that the characters in the guess are in the set [hi. .lo]. Thanks to this feature, it would be a trivial matter to rewrite the Pascal program so that the target and guess be composed of *any* set of values, contiguous or not; for example, to make it into a card game, one could use the set ['0'..'9', 'j', 'k', 'q']. In BASIC this would require a completely new program.

6. CONSTANTS. The second line of the Pascal program defines two program constants. 'Maxnumch', for example, is set to 10. This feature of Pascal allows for a more abstract and therefore more meaningful program. Instead of seeing '10' in the program and wondering 'Why 10?', one sees maxnumch and remembers that it represents the maximum number of characters in a token, whatever that number happens to be. In addition, this makes *changing* the value of maxnumch throughout the program simply a matter of changing one line. The BASIC programmer must either type '10' throughout the program, in which case changing its value becomes an ordeal, or else resort to the illogical and wasteful expedient of designing a variable which never varies. Gone forever is the absurdity of a pi 'function' — one simply writes 'pi=3.1415927' at the beginning of the program and forgets about it after that.

7. CONTROL STRUCTURES: ASSIGNMENT AND CONDITIONAL STATEMENTS. We have seen how Pascal outperforms BASIC at setting up meaningful data types. Now let us turn our attention to the other side of the coin, and see how data is manipulated in the two languages.

In Pascal as in BASIC the fundamental kind of statement is of course the assignment statement. But where BASIC lacked the courage to introduce a special symbol for this all-important operation, using instead the awkward LET and the ambiguous equals sign, Pascal (like Algol) clearly distinguishes between the assignment operator (:=) and the relational operator (=).

To control program flow from one part of a program to another, two kinds of branching are essential. In the first, flow is determined by the values (*true* or *false*) of Boolean expressions such as 'X=1'. In both languages, this kind of branching is achieved by the **if** statement, but the Pascal version is far superior for two reasons. For one thing, it has an **else** clause which is so desirable it has begun to show up in certain versions of BASIC. But even more important, it is a much more natural tool for expressing algorithms because it groups statements with the conditions for their execution, rather than the reverse, as BASIC does. Consider for example the following program segment:

```
100 IF X=1 THEN 400
200 LET Y=456
300 GO TO 500
400 LET Y=123
```

It is absurd to place the action to be taken if X equals 1 as far as possible from the expression 'X=1' and instead place alongside 'X=1' the action that will be taken precisely when X is not equal to 1. This kind of thing is well and good for machine languages, but it is not the way we think and has no place in a high-level language used to express our thinking in machine-usable form. It makes far more sense to do as Pascal does:

```
if x=1 then y:=123 else y:=456
```

This is more readable as well as more elegant.

The confusion in BASIC's control structures reaches catastrophic proportions when we come to the second kind of branching, where flow is determined not by Boolean expressions, but by expressions which may take many different values. The particular course of action to be taken depends on the value of the expression. Some versions of BASIC provide a rudimentary version of this feature in the computed GO TO statement but the Pascal **case** statement is incomparably better for the following reasons. First, any scalar type except real may be used as the **case** variable. In the sample program the **case** statement in the procedure command depends on the value of the character ch. Second, the values specified need not be contiguous. In our example, the four values are 'r', 'q', 's', and 'c', whereas in BASIC they could only be the integers 1, 2, 3, and 4. This is an inelegant approach often requiring painful contortions. Third, as with the **if** statement, Pascal groups statements with the conditions for their execution, while BASIC does not. Thus the statement that will be executed if the command is 's' is the compound statement:

```
begin endofround:=true;
endofgame:=true end .
```

If the reader has any doubts as to whether the Pascal version is superior, I would ask him to look at the four commands in the procedure command, then try to trace the same flow of control in the BASIC program given that it depends on the value of V read in at line 120. (Hint: the BASIC program fails to distinguish between data and action, so that V is the player's guess as well as his command.) I think that any fair judge will have to admit that it is easier to follow the Pascal program than to chase all over in the BASIC version.

Pascal's approach to program control is made possible in large part by the compound statement. In Pascal any sequence of statements may be made into a single compound statement simply by bracketing it with the symbols **begin** and **end**. This possibility of grouping statements into meaningful wholes contributes to the structure of Pascal programs; in fact, the action parts of Pascal blocks are nothing more than compound statements.

Before leaving the subject of conditional statements, I must mention BASIC's lack of logical operators. I am sure that BASIC is already sufficiently embarrassed at its deficiency in this area, so I shall tactfully restrict myself to asking which is clearer:

```
100 IF X=1 THEN 500
200 IF Y>=2 THEN 800
300 IF Z=1 THEN 500
```

or

```
if (x=1) or ((y<2) and (z=1))
then . . .?
```

8. CONTROL STRUCTURES: REPETITIVE STATEMENTS. In a high-level language it is essential to provide ways to repeat a given statement until certain end-conditions are met. We may distinguish three cases: (a) The statement is repeated a specific number of times, no matter what. (b) The statement is repeated while (as long as) a certain condition is true. (c) The statement is repeated until a certain condition *becomes* true.

BASIC creates utter algorithmic confusion by providing only one control structure for all three cases, namely the FOR statement. Thus a reader of the BASIC program would quite naturally expect the loop starting at line 60 to be executed A+B+1 times, but this is *wrong*: it will execute until *either* I=A+B+1 *or* the player ends the round *or* the player ends the game *or* the target has been guessed. Once again, BASIC manages to camouflage completely the intended algorithm, which is clearly case (c), not case (a). This algorithm is perfectly expressed by the Pascal version:

```
repeat command until endofround
```

where 'command' sets the Boolean variable 'endofround' to true whenever any of the foregoing conditions occurs. Needless to say, Pascal also supplies a **while** statement for case (b).

9. PROCEDURES AND FUNCTIONS. It would have been nice to conclude with a point on which BASIC did almost as well as Pascal, but unfortunately, BASIC does very poorly with procedures and functions. Indeed, BASIC's miserable handling of subprograms is probably the single strongest argument in favor of ditching the language altogether, whereas Pascal's superb subroutine declaration facilities are a continual source of delight. Considerations of space prevent me from doing more than list some major points of comparison: (a) As already mentioned, Pascal subprograms bear *names*, not numbers, making for self-explanatory programs. (b) In Pascal, subroutines may have parameters, passed either by value or by address; without this feature I am not sure one should speak of subprograms at all. The procedure tally is used to keep track of both white and black by passing these two variables as parameters. (c) Pascal functions can return any scalar type or a pointer, not just numbers. (d) Pascal's block structure means that subprograms may define local constants, types, variables, and subprograms. An example is the procedure tally, which is local to the procedure command. This means that storage may be allocated as efficiently as possible, yet it is easy to guard against unwanted side effects. (f) Machine language programs external to the main program may be called.

IS BASIC EASY TO LEARN? We have seen that Pascal out-performs BASIC right down the line, and before concluding I would like to consider the oft-repeated advertisement that BASIC is an easily learned language closely resembling simple English. I think that even a casual consideration of this statement will reveal its falsity. As far as I know, two-character names and GO TO statements have never been elements of good English writing style. 'And', 'not', and 'or' are surely among the most common words in our language. We do not say, 'If it rains then 400 go to the beach go to 500 400 stay home', we say 'If it rains than stay home else go to the beach'. We do not tell our children to do their homework 999 times when we mean they should do it until they get it right. High-level languages should be judged on the simplicity and flexibility of their basic constructs, not on how much they look like English, but even on the latter score BASIC's claims are pure advertising hype.

Is Pascal harder to learn than BASIC? Frankly, I do not know. But of this I am certain: a Pascal subset consisting only of the four basic scalar types, the array as a structuring method, the five control structures and the subprogram facilities would *still* put BASIC to shame, and would be *easier* to learn than BASIC because it would be more systematic and more flexible. Even the 22 constructs of the full language represent a trivial pedagogical burden given the power of the language. Add to this the fact that in learning Pascal one is learning to think algorithmically and I think we need not fear Pascal's being unsuited for beginning programmers.

CONCLUSION. The radical difference in design philosophy between Pascal and BASIC was driven home to me recently by an item in *Kilobaud*. Some poor BASIC programmer had, quite naturally, felt the need to control program flow depending on whether a 'y' or an 'n' were input from the keyboard. Unable to do this easily in BASIC, he was seriously proposing a new statement of the form ANSWER F1, F2 which would brand to F1 or F2 depending on what was input from the keyboard. To a Pascaler, it is difficult even to imagine that what in Pascal amounts to a simple one-line function declaration (function answer: boolean; begin answer:=input↑='y'; readln end) should be in BASIC a question of redefining the language itself, to be fought out among the implementers and in the halls of ANSI, the American National Standards Institute.

BASIC offers an absolutely minimal set of features and expects you either to devise makeshift solutions or to design a new version of the language when they are not adequate. No wonder there are so many different versions of BASIC! Pascal offers a somewhat wider selection, but avoids the pitfall of trying to incorporate every feature known to man, as PL/1 seems to. Instead of trying to foresee every possible application which might arise, Pascal's designers chose just those features which allow the user to expand the language himself to suit his needs. It is this combination of power and simplicity which makes Pascal the perfect choice for a standard microprocessor language. BAN BASIC! □

VIDEO DISCS

(Continued from page 15)

this test, the computer might call up another simulation (more or less complex, depending upon the student's performance), or a terminal video sequence, as dictated by the strategy of the courseware developer. One significant advantage to the teacher of this approach, in addition to the obvious pedagogic advantages, is the ease of use. Only a single disc need be loaded, rather than a computer program, a slide carousel, a film, and an audio cassette, each separately, and each into a different machine.

The possibilities for educational impact with video-disc systems indicated in the four items are very exciting—especially the interactive use with a computer. It should be pointed out, however, that the consumer video-disc player to be offered for sale starting in late 1977 will not have the capability to interact with a computer as indicated above. With that player, fast forward and reverse are executed under manual control. A video-disc player must become available which can communicate directly with a computer, telling the computer which track it currently is reading, and accepting a command from the computer telling the player the next track to read. Philips and MCA are working independently on players for the educational and industrial (E/I) users. These units are a year or two away, and are expected to cost in the order of $1,000-1,500 when they become available. It is expected that these so-called E/I players will have a local microprocessor and some local memory to achieve the interactions described above. The video-disc systems which have been described above have great potential for education as stand-alone devices and as part of an information-processing system in conjunction with a computer. This potential must remain latent until educators are able to obtain video-disc systems and are able to obtain access to master systems to produce special-purpose discs. □

References

1. Kenney, George C., 'Special Purpose Applications of the Optical Videodisc System.' *IEEE Transactions on Consumer Electronics* November, 1976, pp. 327-338.

2. Bork, Alfred M., 'Videodiscs—The Ultimate Computer Input Device?' *Creative Computing*, March/April, 1976, pp. 44, 45.

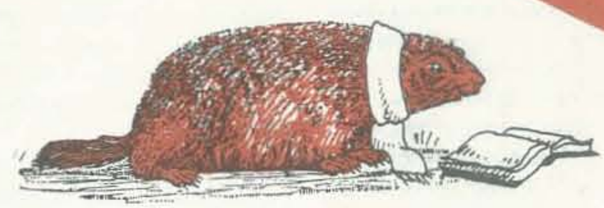
TINY LANGUAGE

Bob Albrecht, aka the Dragon, retired as editor of this magazine (then a newspaper called People's Computer Company*) over a year ago. Since then he's spent lots of time with kids and computers in classrooms. He's generating and gathering lots of ideas and information about how computers can be made fun for and accessible to kids.*

In particular, Bob has decided a new programming language is needed. Dennis Allison, a local computer consultant and long an active supporter of ours, agrees. This is the third in a series on suggestions for a 'tiny' language for kids and those who work with kids.

We encourage input from our readers, especially those who work with kids, whether or not you're a computer specialist.

We said we'd announce a contest structure for our Tiny Language extravaganza this issue, but we're not going to do so after all. Reader participation is just beginning, and it's too soon to tell if there's enough interest to merit a contest. Then too, if we're really going to take our time and do this thing right, we may need to expand our collection of prizes to cover a period as long as a couple of years to ensure we have adequate time to extensively test our notions.



BOB WALLACE'S SUGGESTIONS

I have some ideas for your new Tiny Language. You mentioned having a simple graphic capability; a real good idea! One of the best graphics ideas used for 'kid' languages is the turtle; see Smalltalk, for example. One nice thing about turtles is their relative nature: you can write a 'turtle subroutine' to draw a graphic shape, and later position it anywhere on the screen. You can even rotate it. OK, since everything in this language is a string, a turtle is a string, too. What do we need to specify a turtle? Well, we need:

1) The color. There are two ways to do this: first, each turtle string can have a letter to indicate the color; for now, perhaps 'L' for light and 'D' for dark on black and white sets. The other way to specify a color is to have a 'current color', and change it as needed. I think the latter might be more flexible.

2) The position. Let's use the X and Y position, starting at zero in the lower left corner. This avoids negative numbers, and we want to keep it simple. In the turtle string, use a fixed number of digits per position, so parsing is simplified. Examples might be '012/024' or '060,030'. Three digits are enough for a 1,000 x 1,000 point display, but two (100 x 100) would not be enough for (say) the Polymorphic display.

3) The direction. This could be a number, modulo-something, like 0 to 7 for 8 directions. Alternatives might be 0 to 3 or 0 to 15, or compass points, like 'N' or 'SSW'.

Now we need some functions to make the turtle do things. These functions will take a turtle and a number as arguments, and return an updated turtle. The MOVE function will also make the turtle move on the display. I'll define:

```
MOVE (TURTLE, NUMBER) – moves the turtle a specified
                        number of spaces.
TURN (TURTLE, NUMBER) – turns the turtle clockwise.
COLOR (TURTLE, LETTER) – changes the color of the
                         turtle.
```

For example, let's draw a square:

```
TR:= "D:024/064,0"      (dark turtle at (24,
                        64) points up)
FOR I = 1 TO 4          (4 sides in a square)
  TR:= MOVE (TR,15)     (draw a side)
  TR:= TURN (TR,2)      (turn 90 degrees)
NEXT                    (until finished)
```

Some other ideas for the language:

1) Forget operator precedence. Calculators and APL get along fine without it; it adds a lot of processing code and time; it complicates the language.

2) IF statement: Microsoft BASIC executes all statements to the right on the same line if the test is true. FORTRAN and C drop the THEN and just put the relational test in parentheses. I suggest a combination, IF followed by an expression in parentheses, followed by all statements to execute if the expression is true. ELSE clauses are handy but not absolutely necessary. One other point: I'm not sure whether 'false' should be defined as a null string or as a zero numeric quantity (like '0' or '000').

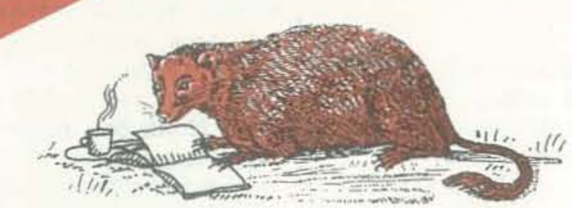
3) Multiple statements per line are very useful and easy to implement.

4) Besides the FOR . . . NEXT loop, include a LOOP . . . EXIT . . . REPEAT construction. This will handle the WHILE and UNTIL constructs of the structured languages, and many other situations besides. I suppose you'll need the GOTO as well.

5) Make blanks significant between identifiers and keywords, but ignore them elsewhere.

6) I hope at least two character identifiers (they could both be letters) are allowed. It's easy to allow identifiers of any length, with only the first two significant. To simplify somewhat, make only the first 2 letters of keywords significant also. This means the user can't have

READER FEEDBACK



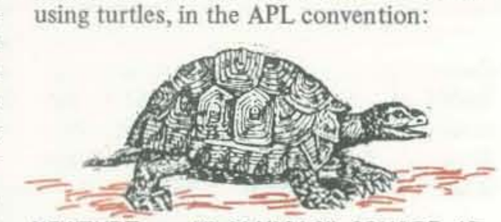
variable names like 'FOO' since the first 2 letters conflict with a keyword ('FOR'). I never use numbers in identifiers; perhaps not allowing identifiers like 'X1' would simplify things.

7) The North Star BASIC string convention is pretty simple and flexible. Any string can be followed by two numbers in parentheses to define a substring; for example, STRING (4, 2) means characters 4 and 5 in STRING. A one-dimensional array of numbers can be easily simulated with this, as well as the Microsoft LEFT$ and MID$ functions. Since you have one-dimensional string arrays, this gives you two-dimensional numeric arrays, as well.

8) Subroutines (functions, procedures) are important! Invoke them by an appearance of the name; try not to require a 'CALL' keyword, or parentheses following if the subroutine doesn't have any arguments. 'GOSUB line number' is the single worst characteristic of BASIC. One way to simplify is make every subroutine a function (and every statement an expression, for that matter). Use the 'RETURN (value)' construction to leave a function; it's much cleaner than assigning the value to the function name. Local variables and recursion are nice, but probably not necessary for a tiny language. Since everything is a string, arguments would probably be passed as pointers to strings.

9) String space management does not have to be complicated. Define a string as a two byte length (alright, maybe one byte), followed by the string. Just concatenate these all together in available memory. Define a string with an initial byte zero as a free string. To find space for a new string, start at the beginning of the string space, and see if the first string is free and big enough. If it isn't free, go to the next one (that's easy, add the length to the current string space pointer). If it's free but not long enough, see if the following string is free; if it is, collapse the two free strings and try again. Once you have a space long enough, remember to make any extra space into another free string. To free a string, just zero the first byte. I coded this in 8080 as part of another (not completed) project, and the code wasn't too long; 256 bytes very approximately.

10) For easy extensibility, clean structure, and so on, use the APL subroutine convention. This does limit you to zero, one, and two argument functions, but I don't think this is a serious limitation for a tiny language. For example, let's do a SQUARE function using turtles, in the APL convention:



```
NEWTURT := "D:024/064,0" SQUARE 15

. . .
$ TURTLE SQUARE SIZE     (define 2-argu-
                         ment function)
  TUT := TURTLE          (don't affect ori-
                         ginal argument)
  FOR I = 1 TO 4         (draw four sides)
    TUT := TUT MOVE
           SIZE          (draw one side)
    TUT := TUT TURN 2    (turn 90 degrees)
  NEXT: RETURN TUT       (all finished,
                         leave)
```

Well, I could design languages forever, but I'll quit here.

Bob Wallace
CoMind Design
PO Box 5415
Seattle, WA 98105

BOB ALBRECHT RESPONDS

The APPLE computer has a choice of 16 colors. It also has a Tiny BASIC that is unusually easy to teach to kids. Go to your local computer store and try this program on the APPLE.

```
10 REM COLOR RANDOM ART
20 COLOR=RND(16)
30 X=RND(40):Y=RND(40)
40 PLOT X, Y: GOTO 20
```

For information on the APPLE, contact APPLE Computer Company, 20863 Stevens Creek Blvd., B3-C, Cupertino, CA 95014.

Directions could be 'implied by' a numeric keypad such as that used in the DRAW program on page 18. Remember, most kids will be using calculators at home or at school, so the keypad will be very familiar!

If we forget operator precedence, let's use calculator arithmetic, *not* APL. Kids will already be used to calculator precedence.

The string conventions you suggest are similar to HP2000 BASIC and Cromemco BASIC. Suppose X is a one-dimensional array. To get characters 4 and 5 of X sub 3, we might write X(3:4,2).

The Z-80 with its Table Look Up and Block Move instructions, will love your suggestions about string space management.

In addition to string variables, should we also have a small set of 16-bit numeric variables? Perhaps A# through Z#?

Thanks, Bob. This is the kind of participation we are looking for.

The Dragon

DENNIS ALLISON RESPONDS

I agree that graphics is an important part (or should be) of any new Tiny Language. The ideas in Smalltalk are really hard to separate from their display. Incidentally, I prefer their syntax to the one you suggest. Interested readers should look at the article by Alan Kay in Sept. 77 *Scientific American*, the article by Kay and Goldberg in the March 77 issue of *Computer Magazine*, and at 'Personal Dynamic Media' in Volume 4 Number 6 of *People's Computers* (then known as *People's Computer Company*).

Operator precedence doesn't really complicate the language or the processing particularly. As a creature of habit, I find I expect it. If we do give it up, we'd best go all the way and adopt APL's right-to-left evaluation to maintain consistency.

Else clauses are very handy if you want to keep track of what is going on. The structured programming gurus would have us believe that one should not write an if without an else. I am in their camp. Incidentally, your BASIC orientation is showing. There are other possibilites than the line-based syntax which BASIC provides. (Look at C for example.) I think the Microsoft IF is a cop-out. It is an unobvious effect. An IF in the middle of a line has a different effect on the program than if it appeared in some other place. The language violates the rules of locality independence.

Multiple statements per line presume that you have lines as a significant language construct. If you do, they are easy to implement, but difficult to implement without funny side effects. For example, why must DATA statements be the first statement of a multistatement line, why do GOSUB statements not return to the next statement but the next numbered statement, etc.

NEW RESOURCES FOR TINY LANGUAGE DESIGNERS
from the Dragon

1. *Teaching Smalltalk* by Adele Goldberg and Alan Kay. Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304.

   An important resource for those who would design computer languages for children. I might even say, an *essential* resource.

2. 'Getting Into Games', *Personal Computing*, November/December, 1977, pages 85-89.

   Perhaps the people who make video games are making more progress towards a usable language than the personal computer manufacturers. Also read the article, 'Tools or Toys', by Jake Roamer – same issue of *Personal Computing*, pages 83-84.

3. 'String Processing, Anyone?' by Daniel Chester. In a recent issue of *Print-Out*, published monthly by the Central Texas Computer Association, 508 Blueberry Hill, Austin, TX 78745.

   I haven't seen this one yet. It describes a String Language (STL) that provides some of the features of LISP and SNOBOL and can be implemented on a microcomputer.

Cannot agree more! looping constructs in BASIC *are* abysmal. But, I'd like to see nesting required to be static rather than dynamic. For clarity, one should always match FOR and NEXT.

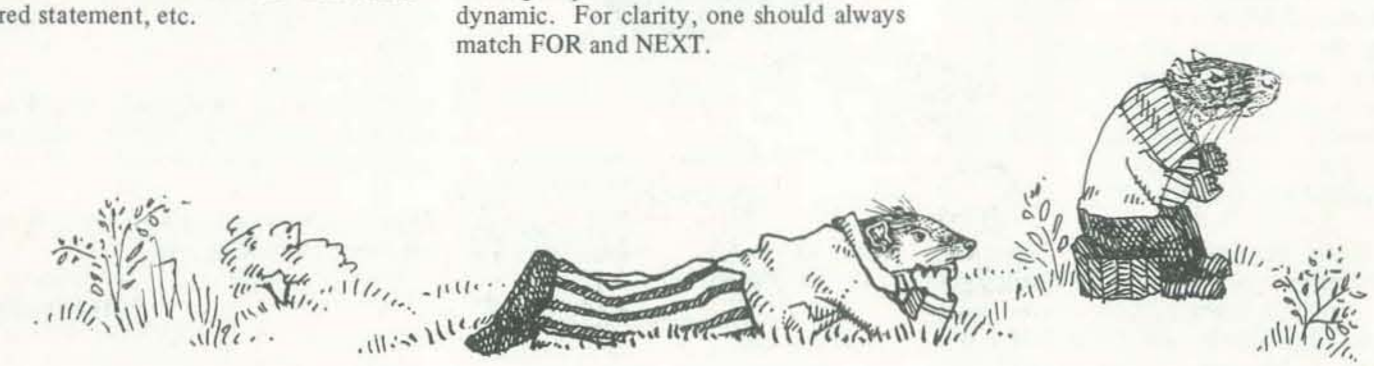
Blanks should be significant in much the same sense they are significant in English text. They are to delimit the sequences of characters which make up words. All rational languages require this. FORTRAN and BASIC are in the other class.

Multicharacter identifiers can certainly be provided. More important is some mechanism for localizing identifiers and for abstracting functions. What is needed is some kind of procedure mechanism with local variables. I expect recursion would be supported.

Using North Star BASIC string convention is as good a convention as any. Using indexes in the fashion of North Star does violate one rule of language design–that is, consistency of use. One has to interpret STRING (2, 4) differently than ARRAY (2, 4) though you are using the same syntatic construct. It isn't pretty or aesthetic.

Procedures are important. The CALL keyword is indeed redundant and can be eliminated. The seemingly redundant empty parentheses following a parameterless procedure call have several potential justifications. First, they call attention to the fact that a procedure is being invoked rather than a variable. Further, in languages like C where the procedure designator need not be a name but may be a pointer-valued expression, the empty argument list is necessary to distinguish the call from a simple expression evaluation.

Your string allocation scheme suffers from two major problems. First, one has to scan all the strings in order to find free space. One would be better off keeping a list of free space and require that the minimum space allocated for strings is adequate for the length and a pointer. The real problem occurs when the system has been running for a long time. The tendency will be to generate many small free blocks sprinkled through the memory space. Eventually there will be no place for a new string even though there is adequate storage because no block of contiguous storage is large enough. When memory is dear, this problem is particularly acute. To solve it, one needs to have some sort of compaction scheme. The difficulty one then encounters is that all references to all strings must be appropriately patched up.



The APL convention is not limiting in APL because it allows vectors as arguments. In a tiny language it might be acceptable, particularly if one allowed pointers. One could then pass a pointer to a list of arguments rather than the arguments themselves.

Dennis

LEIGH JANES' SUGGESTIONS

For Bob Albrecht and Dennis Allison: only a *tiny* language? If your Dragonsqueak turns out to be any good, won't you eventually want to expand it into a full Dragonroar? If so, wouldn't it be better to plan Dragonroar with Dragonsqueak as the first step?

One of the things I would like to see is the syntax checker as part of the editor so that errors of form are caught immediately. (In general, I'd like to be able to use a single editor and call the applicable language's syntax checker as a subroutine.)

There must be a better way to initialize data than by READing DATA statements. If I remember correctly, Fortran provides a nice way–why not borrow (steal?) that method?

If you must abbreviate, please don't use RND for 'random'–try RAND–RND seems to have a bad habit of meaning 'round' to people. (Yes, I know one can get accustomed to the idiosyncrasy–but wouldn't you want to avoid as many idiosyncrasies as possible?) All abbreviations should be chosen carefully. (SQR tends to mean 'square'–why not SQRT for 'square root'?) Why would we want to squeeze blanks out of a string? (Why only blanks?)

Please have labels for the statements to GO TO: line numbers don't carry much meaning.

Leigh Janes
29B Robbins Lane
Rocky Hill, CT 06067

DENNIS ALLISON RESPONDS

Leigh,

We are arguing for spare, aesthetic, and powerful Tiny Languages because we feel that they provide the best vehicle for programming. A Tiny Language need not be an unpowerful language; its universe of discourse is simply more limited. Really large languages, like PL/1, are very very difficult to design, implement, and use because of the interaction between the various language elements. Often, a large language is really a small language with lots of special-purpose bandages to satisfy the whims of particular user communities. Such languages have bundles of special cases and lack much internal consistency. All in all, we believe small is beautiful in both languages and economic systems.

It is nice to have syntax check at line-entry-time. Many BASIC systems do this; many others do not in the name of implementation ease and efficiency. Not all errors can be caught here. Problems related to the flow have to wait until the whole program is ready to run. Others must always be done at execution time; divide by zero, for example, falls into this category. More important, syntax checking at line-entry time seems to require that the programming language be line-oriented in the sense that each line is a single functional unit independent of all other lines. This rigid requirement makes it difficult to construct meaningful control structures.

The data statement syntax in BASIC is, well, yeach! It's as it is to make READ and INPUT as much alike as possible. Many BASIC systems use the same code for both with a slightly different get-next-character subroutine. FORTRAN's data initialization mechanism is straightforward, but is physically distinct from the declaration. I prefer something like the PL/1 declaration which allows the variable to be declared and initialized in one fell swoop. Initialization is a fairly complicated issue in language design. How should it be handled in recursive environments? Should it be read-only or read-write?

Naming conventions for intrinsic functions are now nearly traditional. I don't share your problems with RND and SQR, but then I am not wedded to their use. In any reasonable system you should be able to change the names to fit your particular preference, but the choice of names should not be a big issue. You should be comfortable with changes of notation.

Labels for GOTO's! Heavens, we'll solve that easily enough. We won't have GOTO's.

Dennis

TODD VOROS' SUGGESTIONS

Ideas regarding design of a TINY LANGUAGE FOR KIDS (TILK):

1) Suggest you name your language TILK if you don't already have a name.

2) When designing games, those based on the concept of 'balanced randomness' are often the most enjoyable. 'Balanced randomness' simply means that random events occur that both aid and hinder a game player's efforts to achieve his goal, but the *total* summation of the random events is *zero effect* on the game. Thus, over the span of the length of a game, say 10 positive events occur that help the player (such as border expansion in KINGDOM) then there should be approximately 10 negative events hindering the player (such as theft in KINGDOM).

The ability to generate a random event under the control of a known distribution curve would be a worthwhile addition to the capabilities of your version of TILK.

A function, let's call it '%' should be made available in TILK to perform this service. It might work something like this:

%5:T 'WELL, YOU ARE LUCKY. YOU GET AN EXTRA $5'

This would cause the statement after the %xx to be executed approximately xx% of the time (in this example, 5 times for every 100 passes) through the program.

%20 : statement

would cause *statement* to be executed 20 times out of every 100 calls. Note that this *does not* mean *statement* will be executed every 20 calls to the % function. Each time % is invoked, the probability sample is based on a random bell curve. This is the technique used to generate random events in KINGDOM. The LRAND function in the FORTRAN version of KINGDOM can indicate a simple method of implementing this type of function. It seems to my way of thinking that this might be a desirable ability for your language, especially since it is fairly easy to implement. If generating random numbers in TILK is a problem, simply sample some random character in an input buffer to use as a new seed. Of course, the next seed can be used to point to a new location in the input buffer to point to the next seed, and seeds do not have to be picked up on *every* input (again, let the random number generator decide that. . .) It should make for variable games between different players of games. . .

% will execute with a probability of whatever xxx is specified after the %. This can be used to bias games.

3) TILK should warn the user when he is running out of memory space. . *.before* you get to typing the 2nd last line of your program. . .

4) As a Systems Programmer one of my major complaints has been (and is) that of insufficient detail paid to facilities to aid the programmer in DEBUGGING HIS PROGRAMS. In a language like TILK, especially where inexperienced users will be trying to write working programs, *every* effort should be made to assist the user when things GO WRONG (and they will).

In this area, I propose:

An option command * that will cause *TILK statements* to be typed as they are executed.

An option command $ that will cause *TILK variables* to be listed as they are referenced and updated, and the statements that are doing so.

The ability to determine which statements caused branches and which statements were branched to. (Perhaps only one or two levels deep). This would allow 'wild branches' to be traced quickly and easily.

Occurance of a second *option* command would shut the option off again.

5) I dislike the idea of unconditional branching.

LOOP, IF (logical exp) THEN (statement) ELSE (statement), and CALL are sufficient for solving any algorithm expressed with unconditional branching. Using non-unconditional branching structure in a program results in programs that are easier to understand, easier to debug, and emphasize the structure to the problem to be solved, rather than the architecture of the computer system on which the problem is implemented. If we train our early machine users with good programming habits, perhaps the future of computer software will be better and our marvelous tower of Babel won't be so high.

TILK might be an ideal place to begin with such a structure: A language with no GOTO statement!!! See my *People's Computers* article on SKETCHCODE that will be published next issue.

6) Arrays, if you have them, should be allocated dynamically, as they are in DEC's FOCAL for the PDP-8. See *Programming Languages, Vol II* (published by DEC). Personally, I think TILK should avoid them, as they confuse beginners.

7) TILK should support subroutines. They could be involved by some special character, say /, followed by digit 1-9. So /8/ identifies subroutine 8 and /8 invokes it. Or some such scheme.

8) TILK should be able to *remember* things over power off intervals. This implies a file system which should be very simple to use.

The system could be accessed via a function:

=$xxxx to receive the xxxxth string from the 'memory'
$xxxx= to store the xxxxth string into the 'memory'

Absolutely no mention of record lengths, devices, etc. should be imposed on the beginning user. . . only if he wants to 'remember' something, should he use the $. $ saves stuff, that's all.

Perhaps all variables (string, numeric) should be stored on external storage?

If one were to force the user to end his session with BYE or some such command, then everything could be checkpointed. . .

Perhaps such a 'virtual' memory which the user really isn't aware of would be ideal for TILK. . .

9) The OZNAKI project (Nov-Dec *People's Computers*) suggests directions of GRAPHICS for TILK. However, I feel OZNAKI is too complex for TILK, but might provide a beginning foundation for a graphics subsection of TILK.

10) TILK should detect infinite loops and stop them, if possible. This could be done by using xxxx pseudo-instructions executed without any keyboard input or output.

Well gentlemen, that's all I have to say for now. These highly opinionated viewpoints are strictly my own, and any ideas you derive from them you are welcome to use.

Keep up the good work—and bear in mind my motto:

'Computers should work—not people!' when dealing with programming languages and systems (wish IBM had kept that in mind when they wrote OS/360 JCL. . .)

Todd L. Voros
3721 W Juniper Court
Milwaukee, WI 53209

DENNIS ALLISON RESPONDS

Comments on these interesting ideas:

1) Maybe; any other suggestions?

2) An interesting idea for a games' language, but of limited usefulness in a general purpose language. There is a close similarity between these ideas Dijkstra's *Guarded Comments* (see 'A Discipline of Programming', Prentice Hall, Englewood Cliffs, NJ, 1976). Dijkstra's idea was to preface each statement of a group of statements with a 'guard' (local valued expression). With no account of the ordering, one of the statements inside the group would be selected and executed provided the guard were true; if the guard were not true, another statement would be selected until either all statements had been tried or one statement successfully executed. This proposal is similar except that it specifies a probability distribution.

3) I'm not sure why this is important. Either the program will fit or it won't. The thing to avoid is having a lack of memory totally lock up the system so that you cannot recover the (too large) program to make it smaller.

4) TILK has unconditional branching! It is traditional, but we have not yet decided to include that 'feature'. A trace mode and a monitor mode to simplify debugging at execution time is a good one. Occasional BASIC systems have provided them.

As a programmer myself, I find that careful design and attention to detail when the program is generated provides more return than any number of debugging features.

5) Me too. I rather like the idea of a single looping construct with an EXIT statement which terminates the innermost loop. It's nice to see some of the structured programming and software engineering ideas filtering into personal computing.

6) Data Structures are very, very important. BASIC, FORTRAN, and FOCAL (since you mention it) don't really have them. They have arrays and scalars, but they don't have ways of constructing inhomogeneous objects and passing their names about (references or pointers). Dynamic allocation is certainly a nice and useful feature, but that is only part of the issue.

7) Gotta have procedures, with parameters and local name (variable) spaces. But why frutz up the concept with a yeach syntax. The usual functional notation, 'F(a, b, c)', is pretty well established. If we go one better and define classes (procedures and data structures bound together to form manipulable objects), we might allow introduction of infix, prefix, and postfix operators.

8) File systems are important, but why the proposed syntax? The APL workspace concept is rather nice and would be the one I prefer. But the real problem is the general lack of reasonable physical storage devices.

9) Graphics are a must. A turtle-like approach is simple and easily implemented. (A turtle is an object which will move about the screen on command dragging or holding a pen. Smart turtles may even know about colors.)

10) How is TILK going to detect an infinite-loop? Turing had something to say about that (it's impossible in general). Further, many of my programs have apparently infinite loops.

Dennis □

A Careful Bull in the China Shop

A CHEAP APPROACH TO THE MECHANICS OF ROBOTICS

BY ROBERT ROSSUM

This is the first of a two-part series. In our March-April issue the second part will cover in detail the mechanics of robot building. Additional background material on designing a robot may be found in Robert Rossum's articles, 'Robots as Household Pets' (Vol 5, No 4) and 'Pet Robots: New Capabilities' (Vol 6, No 1).

The name Rossum may be familiar to science fiction fans – it comes from the Capek play, R.U.R. The play is commonly cited as the source of the term 'robot' as it is commonly used ('R.U.R.' stands for 'Rossum's Universal Robots'). Members of the United States Robotics Society are using the family name 'Rossum' as a kind of collective pseudonym for their publications. Members who prefer to be anonymous may publish through USRS under whatever 'Rossum-name' they reserve. Thus far, half a dozen names have been spoken for, e.g. 'S.A. Rossum,' 'D.I. Rossum,' and some folks whose real family name is Rossum have been listed.

'Robert Rossum' writes books, articles, and non-theatrical motion pictures. He has spent most of the past 20 years working in research and developmental laboratories.

We thank MITs for permission to reprint the figures in the article from the September issue of Computer Notes.

PART I

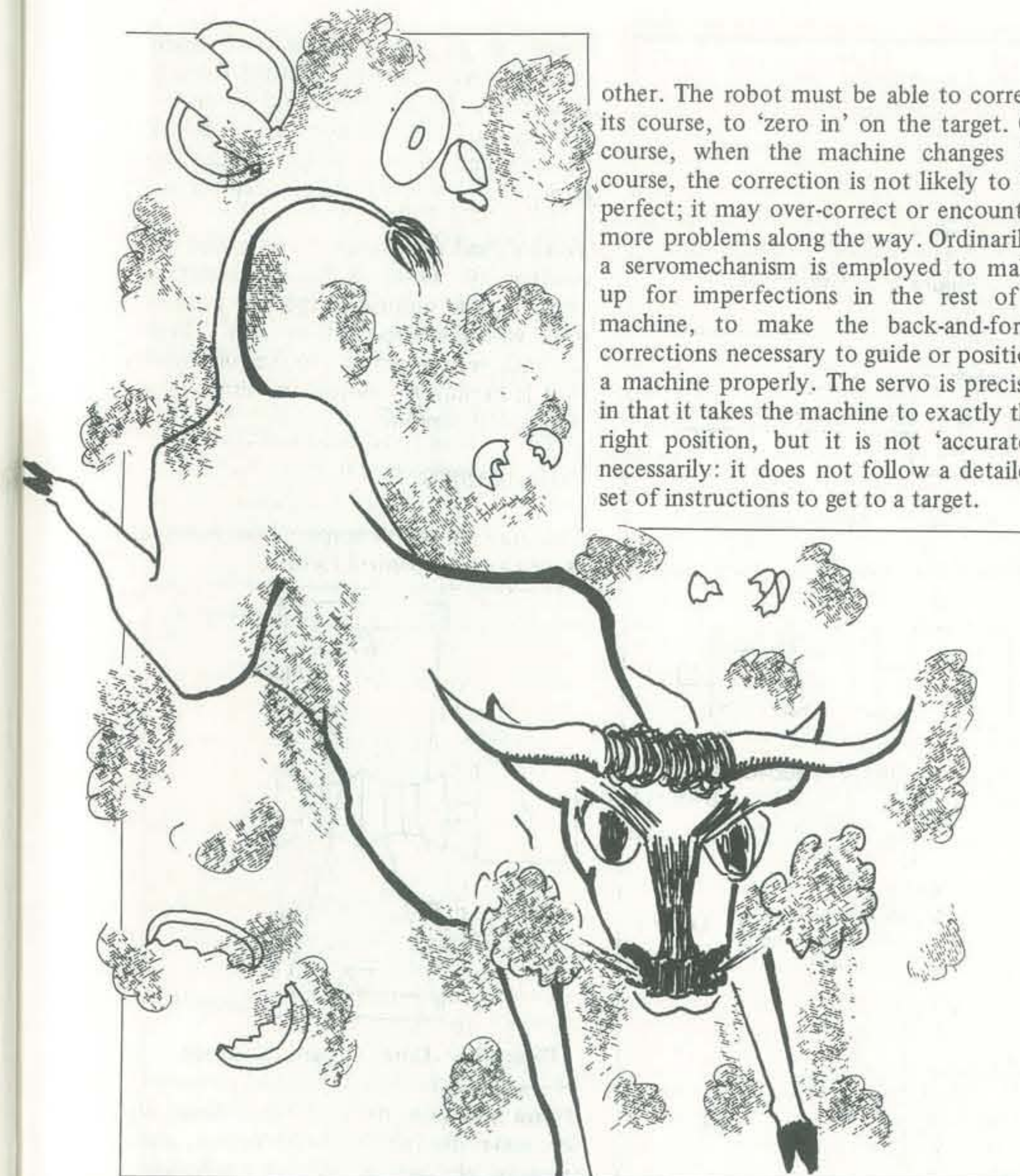
A robot that doesn't move? Not a very satisfactory machine! It has been said that the most interesting thing computers ever do is blow hot air on your shoes while they hum and soak up money. An intelligent machine, however clever, lacks charm if it just sits around like a bump on a log. Perhaps part of the present enthusiasm for robotics is a reaction to this static performance of our clever machines. Roboticists almost universally report their determination to construct mobile systems.

The ordinary roboticist is a good thinker-upper, programmer, planner, and innovator, but seldom a first-rate mechanical engineer and master machinist. It's all very well to draw conceptual plans for experimental mechanical systems, but actual construction and modification of mechanical creatures is prohibitively expensive in time and cash. The mobile systems built by institutions and private workers tend to be awkward, fragile, unstable, and uninteresting, as well as expensive. Indeed, the interesting machines that receive national publicity tend to be anthropomorphic monsters. A recently publicized system is over six feet high and weighs several hundred pounds. It performs some remarkable tricks under the remote control of its master, but looks mighty unstable on its small base. One has a queasy feeling that if this thing dropped a wheel off the edge of a walkway, it would topple over, crushing dog, child, mailman or Volkswagen. The publicity arising from that incident might not bring cheer to other roboticists.

Even the cute little wheeled systems that experimenters set to snuffling around their laboratories have no more athletic prowess than is required to climb over a doorsill or up on a rug without stalling or upsetting. Conventional mechanical systems are generally proving unsatisfactory for devices that are intended to simulate the performance of living things.

And the flaw in the simulation is not chiefly the lack of intelligence. David Heiserman, author of *Build Your Own Working Robot*, has observed that his robots acquire behavioral characteristics of living creatures, responding to their environment in surprisingly complex fashion. The fact that impresses him most is the simplicity of the circuitry involved. A few basic sensory channels, simple reflexes, and a trifle of logic allow his machines to behave like simple animals. It may be that the devices are intellectually trivial, but since they can *move*, displaying their characteristics overtly, and can alter their performance in response to a changing environment, they are interesting. Heiserman's mechanical systems are quite crude, but they do *some*thing.

If experimenters can develop cheap and dirty mechanical systems that any clumsy amateur can build in his own garage, the apparent progress in robotics may be significant. The purpose of these short articles is to call attention to a cheap, not inexpensive, but *cheap* mechanism that may serve in a large number of robotics applications. No detailed designs are offered, but roboticists will be able to employ the basic principles of the system without further elaboration here in print. The trick is to shake off some conventional notions that have obscured the value of this old and familiar mechanism.



In this electronic age, we think of robotics mechanisms in terms of electronically controlled servosystems, stepping motors, and complicated, heavy gear trains. Consider servos. Since no mechanical system is perfectly accurate, we must always provide a trial-and-error system that will let a free-moving device accomplish its tasks in spite of imperfection. For example, if you set your pet robot on a course for the fire hydrant a block away, you can be sure that the critter will miss the fireplug unless it knows one when it sees one, and can hunt around as necessary to find the thing. Just aiming straight from where you are to the hydrant won't work, since irregularities in the pavement, uneven wear in the robot's wheels and gears, bad aim, or a dozen other problems will almost inevitably prevent the machine from going directly from one place to the other. The robot must be able to correct its course, to 'zero in' on the target. Of course, when the machine changes its course, the correction is not likely to be perfect; it may over-correct or encounter more problems along the way. Ordinarily, a servomechanism is employed to make up for imperfections in the rest of a machine, to make the back-and-forth corrections necessary to guide or position a machine properly. The servo is precise, in that it takes the machine to exactly the right position, but it is not 'accurate', necessarily: it does not follow a detailed set of instructions to get to a target.

The distinction between precision and accuracy is important. If your robot is accurate, you may give it instructions such as: Move exactly north 315 feet, 5 inches. Then make a 90° turn, to the left (not an 89° turn or a 91° turn, but a 90° turn) and move exactly 19 feet, 7 inches. Stop there or you'll smash your little lens on the knobby thing that is sticking out of the hydrant.

What are the chances that you really know exactly what the instructions should be, and your robot can carry the instructions out well enough to get within six inches of the hydrant? Not very good, unless you have an uncommonly well-made, expensive machine (equipped with a magnificent inertial guidance system, perhaps) working in an environment that is not very irregular. If there are cracks in the sidewalk, you're in trouble.

If your robot is equipped with sensors and servos, it can use instructions more like this: Move along the sidewalk to the north without falling off the edge or bumping signposts, until you detect something that looks like a fireplug off to the left, about 300 feet along the way. Then move toward that hydrant until you're six inches from it. Stop there.

Chances are good that the robot will go precisely where you want it to go. Precision, not accuracy. The robot may be constructed completely of lousy components, may not be able to turn accurately within five degrees, may be off by three percent in its judgment of distance, but it will do what you want it to do. Recall that living things are built entirely of lousy, individually unreliable and irregular components. Even the brain is constructed of stuff that couldn't meet military specifications for purchasing, regardless of actual performance.

Recall, too, that when an animal lifts its foot, it does not usually have to swing that foot clear around a 360° arc to return it to its starting position. Feet move forward and back, up and down. Tails move to and fro. Muscles in living creatures are paired. Your bicep pulls your forearm up and your tricep pulls it back down.

Mechanical servomechanisms work with paired motors, usually, pulling things first one way, then the other, 'zeroing-in'. The robot builder is usually depressed by the realization that almost everything in his critter must be duplicated – all motors matched, or anyway, reversible. One common ploy is to make the motor pull against a spring that returns a limb to 'normal' position after the motor moves it.

Robot designers ordinarily provide a motor for an arm, a motor for a head, a motor for wagging the tail and so on. Sometimes a very complex, heavy, power-consuming gear system is used to accomplish all these functions with a single motor.

But consider an alternative: the ancient double windlass mechanism, about which you can learn a great deal in handy reference manuals like encyclopedias (see 'windlass', 'capstan', or 'winch and windlass'). The virtues of the double windlass for the roboticist are many.

THE BASIC SYSTEM

The sequential figures show how a basic system can be constructed. In Figure 1, the box with an 'M' on it is a motor; Figure 2 illustrates a long shaft protruding from the motor. A pair of pulleys is placed on the shaft in Figure 3. Above the shaft at some arbitrary distance is Lever A, pivoted at its center (Figure 4). Below the shaft is Lever B, also pivoted at its midpoint (Figure 5). Our interest here is in getting Lever B to do something in particular when we move Lever A.

Figure 1. The motor

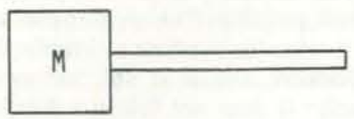


Figure 2. Motor with shaft

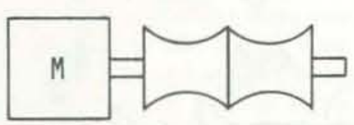


Figure 3. Pulleys added to shaft

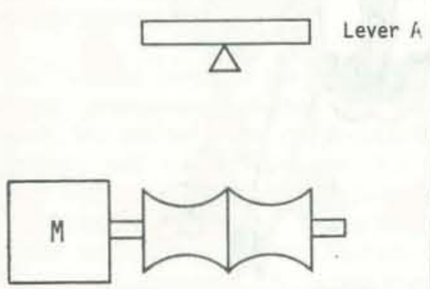


Figure 4. Upper lever added

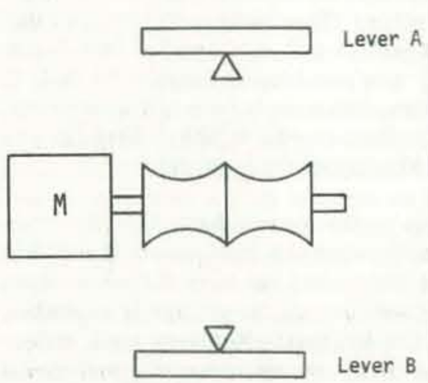


Figure 5. Lower lever added

In Figure 6 we connect Levers A and B with Cords C_1 and C_2. The cords are wrapped loosely around the pulleys on the shaft so that when the motor turns, the pulleys just spin inside the loose cords without affecting them and the levers.

Suppose, in Figure 7, that you take hold of Lever A, tilting it upward at the left end. That pulls Cord C_1 tight around its pulley, though Cord C_2 remains loose around its pulley. Here the mechanical magic begins. As Cord C_1 grips the pulley, the force of the motor begins to pull on the cord. Even it you lift the end of the lever very delicately with your fingertips, the cord, hence also the end of Lever B, will be pulled by the full force of the motor. You need only keep a bit of tension on the top part of that cord to apply the motor's full force to the task of lifting up the end of Lever B.

If you pull the end of Lever A steadily up to some particular position, the motor will wind up the lower part of Cord C_1 until Lever B is cocked at the same angle as Lever A. Then the cord will begin to slip on the pulley, and the pulling force of the motor will be relieved. You have applied a small control force to the upper lever, causing the motor's force to be applied to the lower lever. In fact, a weight of some significance might be hanging from that left end of Lever B.

The weight shown in Figure 8 is far heavier than you could lift with your fingertip. The motor would do the lifting, multiplying the control force greatly.

Notice that when Cord C_1 begins to slip, C_2 is just on the point of growing tight. When the action stops, the windings of the two pulleys are just slightly loose, as they were when the action began. The system is all ready to perform again promptly when another control force is applied to a lever. If you pull up on the right end of Lever A now, Lever B will be returned to its original matching position – a sort of bicep/tricep action.

You've done two things – controlled the position of Lever B by manipulating Lever A, and multiplied the tiny control force with the force of the motor. These are both very important to the roboticist who is hoping to control the limbs of a mechanical creature.

MORE BASICS

You may choose to amplify your *motion* as well as your control force.

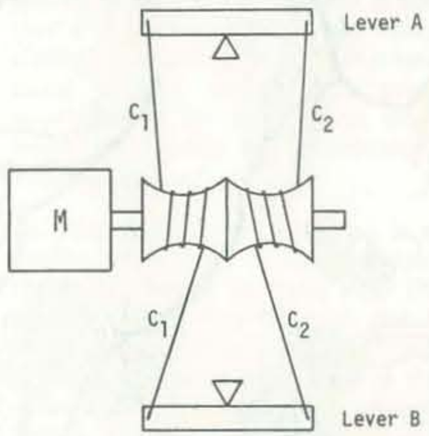


Figure 6. Cords C_1 and C_2 added

If you now raise the left end of Lever A the same distance you did before, the force of the motor will be applied to Lever B in the same way (see Figure 9). The same length of cord will be drawn up by the pulley, but the left end of Lever B will be moved a greater distance. You have multiplied both force and motion.

The motor here may be as large as you like for the application you have in mind. The control force you apply to Lever A may, in fact, be supplied by another motor, since your robot will probably employ an electrical system, and turning power off and on in electrical motors will be a straightforward matter. The control motor (see Figure 10) may be *very* small, both in physical size and power. (The main motor may even be gasoline or steam powered, if you like, depending on your application and your willingness for your robot to breathe real smoke and fire with the interesting associated noises.)

In fact, your control motor might sensibly be a reversible shaded-pole motor. People who know about motors say that a shaded-pole motor can be held in a stalled condition indefinitely without damage, and that's an advantage. (A later article will discuss a mixed bag of alternatives to control motors.) With signals from your robot's brain, presumably your personal computer, you can move Lever B either way automatically, with appreciable force.

The shaft from the main motor may be equipped with numerous pairs of pulleys (Figure 11) so that power may be applied at any point along the shaft to any chosen lever down below.

The shaft may be flexible (Figure 12) so that power can be transmitted from the main motor to remote regions of the robot in which it resides.

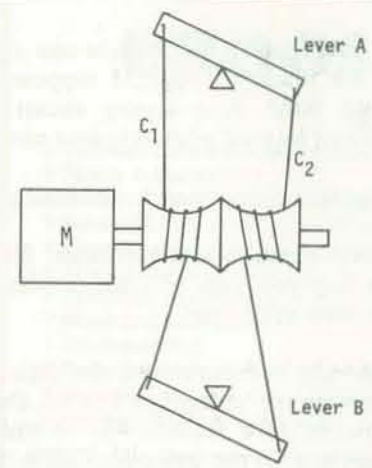


Figure 7. Both levers canted to same angle

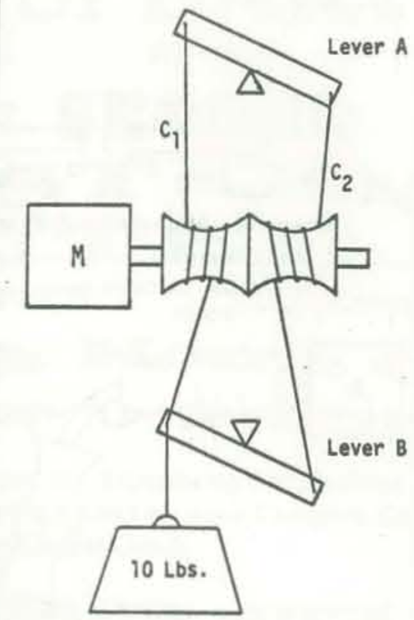


Figure 8. 10 lb weight hanging from Lever B

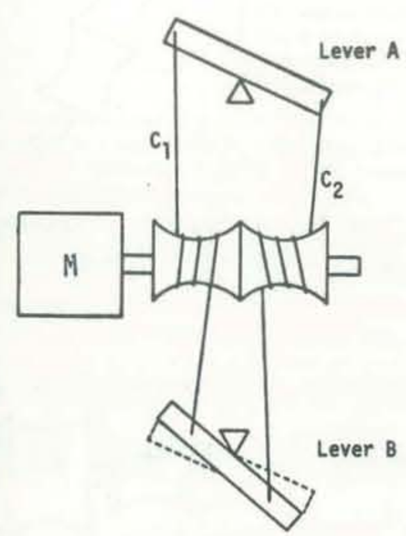


Figure 9. Cords on Lever B now attached appreciably closer to the pivot than on Lever A

The pulleys on the shaft may be of different sizes (Figure 13) so that Lever B_1 may be moved with a different amount of power from that applied to B_2, and so on down the line. Maybe you don't want the robot to wag its tail with enough force and speed to smash a chair leg. You can control the speed and power of the wag by choosing levers of appropriate length and pulleys of appropriate diameter.

A matter of great importance arises at this point in the discussion – the matter of shared power. Obviously, there's a limit to the number of pulleys you can put on the shaft of a given motor. There's a practical physical constraint of some kind to balance your every wish. If you tighten the cords at every point along the shaft, drawing power from the motor at each pair of pulleys, your chances of overloading the motor are very great. But there's the beauty of the system (well, one beauty among many) – it works the way an animal works. A real animal seldom uses all of his muscles at once. When you run, you may be using your leg muscles in an extreme fashion, but you are not simultaneously using your neck and arm muscles to their fullest extent. Chances are that you are not simultaneously trying to bite through a heavy bone, drawing a great deal of energy in your jaw muscles.

Shared power is a very significant factor in the design of animals. You have a certain amount of chemical energy stored locally in your muscles. When you move muscles, you consume some of that available chemical energy. If you exert the muscles greatly, you use up all that's locally available, and must eat more sugar sent up from the liver. With great exertion, you can develop a severe local shortage of energy. Luckily, you seldom exert *all* muscles at the same time, so you don't develop a general deficit of energy. (In fact, though, people do sometimes die of over-exertion. That's one of the problems for people stuck in blizzards. They tend to use up all of their reserves, struggling through the snow, then lie down to rest. Very bad. When they quit moving, they quit pumping new chemicals to their depleted muscles fast enough. The cold and lack of energy may be fatal. When a runner finishes a race, he keeps trotting for a bit, not only out of respect for tradition, but because a sudden cessation of activity could be painful and dangerous.)

The analogy is not perfect, but it's pretty good. This double windlass system allows the energy of the main motor to be shared by many functions in the body of the robot. The *average* load on the motor can be quite low, while large amounts of energy are rapidly available wherever needed. When separate motors are used at all places where energy is needed, those motors must be big enough to supply all the energy that will *ever* be needed from them. That means a lot of extra weight is being dragged around all the time, just in case a burst of energy is needed at any point. The double windlass system solves much of this problem with a comparatively simple simulation of the system Nature has been using effectively for a long time.

This is not to complain of standard mechanical systems—there's much to be said for the clever designs that competent engineers have developed for robotic and non-robotic mobile systems using modern technology. However, cheapness is not a feature of standard mechanical systems, nor can the average home craftsman cope with the standard systems.

The double windlass system can be assembled by the home experimenter with a Tinkertoy outfit or an Erector Set. The interested roboticist can work with this system himself even before the next article in this series is published in the Mar-Apr issue.

The pulleys can be empty thread spools in the experimental system. When you get around to building a rig that's meant to last, you'll want to use metal, because there's a lot of wear. Don't the cords stretch? Sure, and they'll have to be tightened once in a while. So what! At least you can figure out what's wrong and fix it yourself. (And there will be many maddening problems inherent in this system as in any other.) The whole mechanism can be quite sloppy, by machinist's standards, and still work. *Precision can be achieved in a sloppy system without accuracy.*

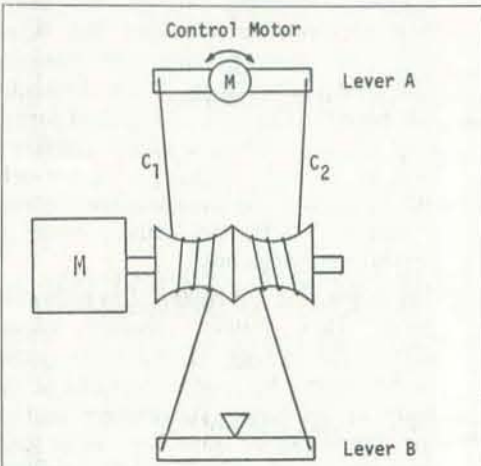


Figure 10. Control motor on Lever A

In the discussions leading to this article someone asked: 'Isn't there a real safety factor in the fact that the cords will slip on the pulleys if they are overloaded?'

'Oh, no. The cords will break before they slip. This is the kind of mechanism people use to pull two or three miles of oil-drill stem up out of wells. The windlass is a powerful tool. Why?'

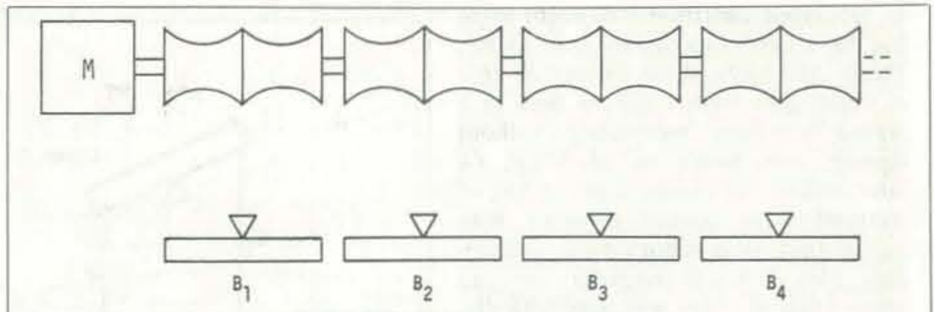


Figure 11. Main motor shaft with more sets of pulleys

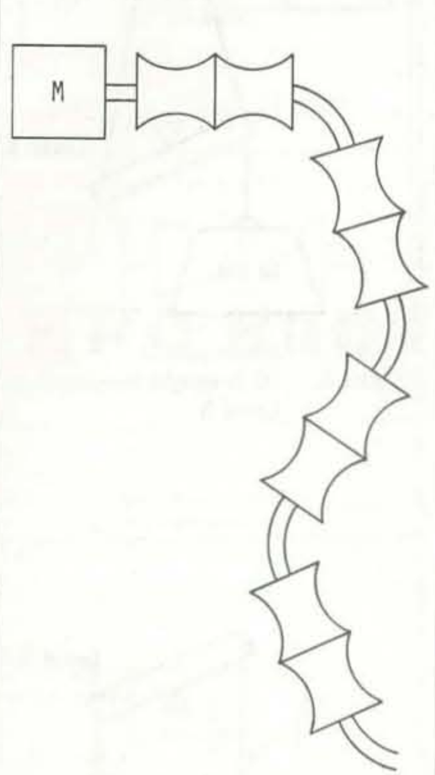


Figure 12. Flexible shaft with pulleys along its snaking path

'Well, I guess I don't want the robot to be too strong.'

'Too strong for what?'

'For people. I don't want it to hurt anybody by accident, and I thought maybe the cords would slip in case the machine happened to be gripping somebody too hard.'

'Ah. Well, you'll have to take care of that in the machine's logic. I suppose you could build in a sensing circuit that makes it turn off when it hears a scream.'

'That isn't the comfort I was looking for.'

'Sorry. A machine is a machine. Build it the way you want to build it. Maybe it can learn to be careful.'

This article doesn't treat the logic, the brain, or the reflexes of a robot, though some of those matters will be touched upon later in the series. Instead, it offers a cheap and dirty approach to making robots *do* something interesting. If you have been stewing in frustration over your inability – financial or mental – to build a working system to go with the brains on your shelf, get busy with the spools and Erector Set motors. □

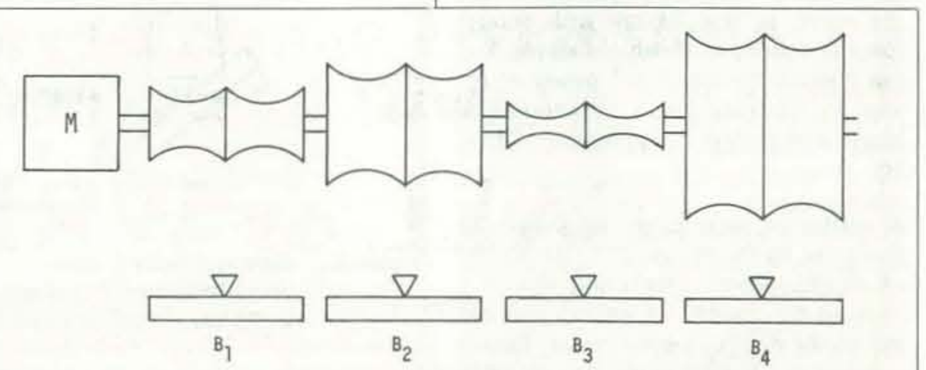


Figure 13. Various sizes of pulleys on the shaft

ANNOUNCEMENTS

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

SOFTWARE

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

6800 TELEPHONE APPLICATIONS

Software Exchange, a newly formed company, is developing a line of low cost software for the computer hobbyist with emphasis on the practical application for the home computer. Two telephone application programs for the 6800 microcomputer are now available. Each program includes complete documentation, with schematic diagrams and instructions.

1. 6800 automatic telephone dialer: $9.95 postpaid includes object code and punched paper tape in Mikbug* format, and instructions for adapting to other 6800 systems.
2. 6800 telephone answering device: $4.95 postpaid includes assembly listing and object code. Compatible with any 6800 system.

Software Exchange, 2681 Peterboro, W. Bloomfield, MI 48033

* Mikbug is a registered trademark of Motorola, Inc.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

THE 6502 PROGRAM EXCHANGE

The 6502 Program Exchange has released a number of new software packages for 6502 systems. These include an extended version of the high-level language FOCAL, a 4K resident assembler, and an efficient Mini-Editor.

The new FOCAL is called FCL65E to distinguish it from the FCL-65 previously released. FCL65E (6.5K) offers 8 to 9 digit accuracy, 8-level priority interrupt handling, string variables and functions, and greater flexibility in its FOR, SET, and DO commands. Complete cross-assembly listings for TIM ($1000-$25F2) and KIM ($2000-$35F2) can be purchased for $35. Both FCL-65 and FCL65E now have all their system dependent software in a zero-page I/O block, allowing easy conversion to other 6502 systems.

A Mini-Manual ($6) and a paper tape or hex dump ($17) will get you started on TIM or KIM systems. A User's Manual, 104 pages of FCL65E examples and further documentation is available for $12. The Exchange offers an expanding library of programs (including a STAR TREK) for FCL-65 and FCL65E.

More information and a list of other available software may be obtained by sending $1.00 to The 6502 Program Exchange, 2920 Moana, Reno, NV 89509.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

NOTICE

We receive a large number of press releases every day and must limit choices for publication to a very few. Concise announcements stand a much better chance of being included than items which require extensive rewriting such as wordy treatises or abbreviated lists of products, specifications, or software. And please, don't send us stuff in all upper case: it drives our typists up the wall.

8080 WORD PROCESSING SYSTEM

Mini Word Processing 2.0 (MWP) enables the user to prepare letters, text and mailing labels or envelopes. When used for correspondence processing, MWP allows name and address entries to be coded with number of group codes and document response codes. For example, an inquiry might be coded with date and inquiry type group codes and a specific response letter body with selected paragraph/phrase insertion document codes. A followup letter might be sent keyed only on group codes.

MWP provides in-line editing and common text/phrase insertions in the text generation module. The letter and text output modules provide text insert or replacement, margin/page control and page numbering. The MWP System is driven by a Menu select routine with seven processing modules.

MWP is extremely easy to use and includes a comprehensive user's manual with varied examples. The price is $195 supplied on a diskette compatible with MITS Disc Extended Basic. See your computer dealer or contact The Software Store, 706 Chippewa Square, Marquette, MI 49855; (906) 228-7622.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

EDUCULTURE INTRODUCES MICROCOMPUTER 'COURSEWARE'

The first wave of professionally-prepared learning materials specifically for small stand-alone computer systems is now in preparation at Educulture, Inc., a California-based educational publisher. The programs, aimed primarily toward secondary and post-secondary education, include comprehensive, coordinated series in mathematics, English, and the sciences.

As initially configured, the programs are designed to run on machines with 32K bytes of random-access memory, single-drive digital tape or flexible disk storage, and medium-resolution CRT displays (512 X 512 to 720 X 1024 addressable points). Graphic capabilities, which allow the use of pictures, diagrams, and the special characters and symbols of mathematics and science are included in the programs.

The Educulture effort represents the first major entry of a publisher into the educational software industry. The payment of standard advances and royalties is expected to attract experienced, qualified authors in a field not previously noted for its monetary rewards. Educulture is the educational technology division of the William C. Brown Company of Dubuque, Iowa, a publisher of college textbooks and other printed learning materials. For further details contact Jon Bosak, Project Editor, Educulture Inc., 3184 'J' Airway Ave., Costa Mesa, CA 92626, or phone (714) 751-2113.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

HARDWARE

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

NEW LOW SPEED MODEM

The Net Works announces their TNW-488 low speed modem which provides an interface between the IEEE bus and Bell's Data Access Arrangement. The modem is on an 8" x 11" double-sided circuit board employing the Motorola MC 6860 modem chip and a UART. It follows the standard of the Bell 103A Frequency Shift Keyed (FSK) modem. Power supplies of +5, +12 and −12 are provided on card.

The capabilities of the TNW-488 include software selected/enabled pulse dialing, auto originate/answer and transmit break. Also included are selectable baud rate (up to 600, filter optimized for 300 bps), long or short space disconnect and error detection. The assembled and tested board sells for $225 with documentation and the bare printed circuit board is available with documentation for $60. Contact: The Net Works, 5014 Narragan-set No. 6, San Diego, CA 92107; (714) 223-1176.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

IBM SELECTRIC PRINTER FOR MICROCOMPUTER OUTPUT

Micro Computer Devices has announced the SELECTERM, a fully converted IBM Selectric II Typewriter. The conversion to a printer enables immediate use with any microcomputer.

The SELECTERM may be connected directly to either parallel or serial port, with all inputs at standard TTL level. No additional software is required since all logic is an internal PROM. The SELECTERM includes a special typing element that produces all ASCII and full upper and lower case alphanumeric characters. Also included are tab command, backspace, vertical tab and bell.

Special features may be ordered including dual pitch, correcting feature, pin feed platen in a choice of 13 sizes, and a noise reduction feature. Any color that IBM offers may be ordered.

The SELECTERM can be used as a typewriter since none of the typing capabilities have been affected by the conversion to a printer. Because the SELECTERM has been approved by IBM, the typewriter warranty remains active, and yearly service contracts may be obtained from IBM. In addition, Micro Computer Devices provides a separate factory warranty on the conversion package.

The SELECTERM may be purchased only through dealers, though OEM inquiries are invited. Full price is $1650. Contact your computer store dealer, or write Micro Computer Devices, 960 E. Orangethorpe, Bldg. F, Anaheim, CA 92801. (714) 992-2270.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

SCIENTIFIC CALCULATOR INTERFACE

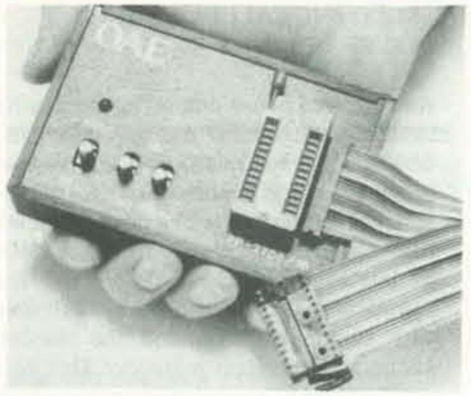
An interface board employing a scientific calculator circuit for use by 8080, Z-80, 6800 and other microprocessor systems has been made available by Mini Micro Mart. The interface is to a powerful scientific calculator chip produced by MOS Technology, the 7529-103; basic and complex math functions can be done with simple software and the absolute minimum of system memory. This inexpensive board permits the user to perform math functions not provided for in many of the BASIC and FORTRAN interpreters and also provides for calculations to a higher degree of precision.

Calculations utilizing interpreters or compilers that would require 8K to 16K of memory can be done in under 1K. The user has available the functions of a fully programmable, sophisticated 40-key scientific calculator including trig functions, inverse trig, logarithms, antilogarithms, exponentiation and factorials; it directly supports two parentheses levels. The user can take advantage of the board's capabilities in three manners: it could be used to supplement the functions of a small interpreter, as a stand-alone firmware math package, or it could be programmed to emulate the functions of a powerful, programmable calculator.

Two versions of the board are available – the RM Series with a pin-out that matches the Motorola Exercisor bus and which could be adapted readily to an Intel SBC 80/10 system and another version is produced in a personal computing S-100 standard bus configuration. Software is included for both 8080 and 6800 systems.

Through the use of CMOS and 74LS IC's, the power requirements are less than 1/2 amp at +5V and 30 MILS at +12V. The board is available in kit form at $99.95. For further details, contact Mini Micro Mart, 1618 James Street, Syracuse, New York 13204, or phone (315) 422-4467.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶



NEW PROM PROGRAMMER

Oliver Audio Engineering now has a new low cost series of piggyback PROM programmers. For example, the PP-2708/16 PROM Programmer plugs directly into any 2708 or TMS-2716 memory socket. The PROM to be programmed is placed in the socket and the data is dumped over the 8 lower address lines using OAE's proprietary interface technique (pats. pending). No additional power supplies are required and all timing and control sequences are handled by the programmer. Only a short software routine is required, and multiple programmers may be connected in parallel for gang programming.

Each unit comes complete with a DC to DC switching regulator, 10 turn cermet trimmers for voltage and pulse width alignment, and a zero insertion force socket. The unit is packaged in a black anodized aluminum case for table top operation. A 5 foot flat ribbon cable interconnects the programmer with the read only PROM socket via a 24 pin plug. Prices are $249 in kit form and $295 assembled and tested. Contact OAE, 676 West Wilson Ave., Glendale, CA 91203, (213) 240-0080.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

TSC MULTI-USER SYSTEM

Technical Systems Consultants announces the TSC Multi-User System. This system allows 4 users to simultaneously use one SWTPC 6800 microcomputer, all running separate programs. The TSC Multi-User Board is a SS-50 bus board containing some required extra memory, interrupt logic, and a few registers. The board is designed to plug into one of the memory slots on the bus. With the board installed, simply load the BASIC cassette included with the board for a four user BASIC system.

Suggested retail price for the TSC-Multi-User Board Kit is $129.95. That includes the Multi-User Board Kit with all parts, IC sockets, diagnostics, and instructions. Also included is a cassette and users manual for a Four User Micro BASIC Plus. Also available from TSC are two versions of 8K BASIC specially adapted for the TSC Multi–User System. One version allows cassette save and load by each user, entirely independent of the other user activity. The second version supports the new SWTPC Mini-Floppy system. With this BASIC, each user can access the disc drives for saving or loading programs. The disc files may also be user password protected. Both versions of BASIC will allow the use of a SWTPC PR-40 printer for program listing.

The system requirements are as follows: 1-SWTPC 6800 Micro Computer; 12K-32K of memory (minimum recommended for use with 8K BASIC and 4 users is 24K); 1-terminal for each user; 1-ACIA Board for each user. To give you an idea of total system price ranges, a minimum 2 user system with 12K of memory, terminals, and TSC Multi-User System will retail for around $1,700. A fully packed very powerful system, including 32K of memory, dual disc drives, four CT-64 terminals, 8K BASIC, PR-40 printer and interface, all ACIA boards, and the TSC Multi-User System will sell for under $4,800. For further information contact: Technical Design Consultants, Inc., P.O. Box 2574, West Lafayette, IN 47906.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

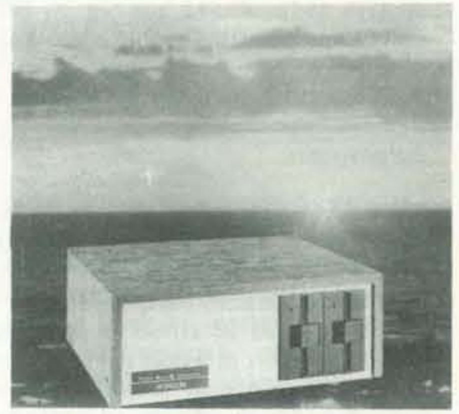
DISKETTES

Manchester Equipment Co. carries diskettes for all of the following systems:

Shugart 800	Shugart SA 400
Altair	Mini-Diskettes
Helios	North Star
OSI	Fortran IV
Digital Systems	Smokey Signal
Imsai	Metropolis
Innovex	

All diskettes are $3.75 each. Orders accepted only by payment accompanying order; Bankamericard, Master Charge, and COD OK. Add $2.00 for shipping and handling. Manchester Equipment Co., Inc., 120 Bethpage Rd., Hicksville, NY 11801.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶



HORIZON COMPUTER WITH BUILT-IN DISK DRIVES

A complete microprocessor system with integrated floppy disk memory is now available from North Star Computers, Inc. Called HORIZONtm, the system is designed for business, educational and personal applications. HORIZON is ready for programming in extended disk BASIC with the addition of a CRT or hard-copy terminal. North Star BASIC includes sequential and random disk files, formatted output, a line editor, strings, user defined functions, and more.

The system is available in two models. HORIZON-1 ($1,599 kit; $1,899 assembled) includes a Z80A processor, 16K RAM, minifloppy disk and 12-slot S-100 motherboard with serial terminal interface. The HORIZON-2 ($1,999 kit; $2,349 assembled) includes a second built-in disk drive.

The Z80A processor operates at 4MHZ–double the power of the 8080. The North Star 16K RAM board lets the Z80A execute at full speed. HORIZON can load or save a 10K byte disk program in less than two seconds. Each diskette can store 90K bytes; the motherboard is S-100 compatible.

North Star also offers additional S-100 boards including a hardware floating point option at $259 kit; $359 assembled, and 16K RAM boards at $399 kit, $459 assembled, with optional parity check and additional serial and parallel I/O ports at $39 kit and $59 assembled. Delivery is 30 days on receipt of order. For more details write: North Star Computers, Inc., 2465 Fourth Street, Berkeley, CA 94710. (415) 549-0858.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

OTHER

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

COMPCON 78

IEEE's COMPCON 78 will be held Feb 27–Mar 2 at the Jack Tar Hotel in San Francisco. A special evening program consisting of exhibits and four panel sessions will present a look at the phenomenon of personal computing. The panel sessions start at 7:00 PM and cover topics such as Women's Contributions in Innovative Computer Applications (Monday), Robotics and Bionics (Tuesday), Computer Magazines (Wednesday) and Computer Art and Music (Thursday). From 5–10 PM on Monday through Wednesday attendees will be able to get first hand experience of a broad range of computer equipment. The COMPCON 78 registration fee covers attendance at all day time sessions and all Personal Computing sessions and exhibits; a $5 fee will enable individuals to attend only the Personal Computing sessions and exhibits. For more information on the Personal Computing sessions contact organizers Alice Ahlgren, Marketing Manager, Cromemco, Inc, Mountain View, CA (415) 964-7400 or Bob Albrecht (415) 323-6117.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

USA-JAPAN COMPUTER CONFERENCE

The third USA-Japan Computer Conference will be held October 10-12, 1978 in San Francisco. This marks the first time this gathering is to be held on American soil.

Papers are solicited on all aspects of computing technology, including computer applications. Papers in the form of complete drafts should be submitted to the U.S. representatives of the Technical Program Committee by March 1, 1978. Drafts should not exceed 5,000 words, and abstracts not exceeding 150 words should be included with submitted drafts.

Individuals planning to submit papers should submit their material to: Prof. Edward J. McCluskey, Digital System Laboratory, Stanford University, Stanford, CA 94305.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

NCC 78

The 1978 National Computer Conference will feature a Personal Computing Festival, to take place June 6-8 at the Disneyland Hotel complex in Anaheim, CA. A special program of papers and presentations relevant to personal computing will be presented. Both one-day and three-day registrations will be available for the Festival. Information on NCC 78 may be obtained from AFIPS Headquarters, 210 Summit Avenue, Montvale, NJ 07645 or by calling (201) 391-9810.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

PET OWNERS GROUP

I'm forming a PET owners group to exchange ideas and information. I'm a broker, particularly interested in financial applications; another member is especially interested in assembly language programming. Carl Martin, 2001 Bryan Tower, Suite 3800, Dallas, TX 75201; (214) 742-5750

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

COURSE

A two-week course in the fundamentals of digital electronics and microcomputer interfacing will be held at Virginia Military Institute from July 17 through July 29, 1978. For information and registration forms write to: Dr Philip B Peters, Dept. of Physics, VMI, Lexington, VA 24450.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

CONTU EXTENDED

House Bill H.R. 4836, to extend by seven months the term of the National Commission on New Technological Uses of Copyrighted Works (CONTU) was signed into law by President Jimmy Carter on October 28, 1977, after having been passed by the Senate on October 13, 1977. Public Law 95-146 required that the Commission submit its final report to the President and the Congress on or before July 31, 1978, rather than on or before December 31, 1977. The Commission was continuing to hold meetings on the subject of new uses of copyrighted works as late as November – check with them for meetings. Contact CONTU, Washington D.C. 20558; telephone (202) 557-0996.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

ACM SIGPC

The Association for Computing Machinery chartered a new Special Interest Group on Personal Computing, SIGPC, at the National Computer Conference in June. SIGPC will be operated exclusively for educational and scientific purposes in the design and applications of computer systems for personal uses. This includes personal computer systems for home, clerical, small business, management and recreational uses.

To join SIGPC write to the Association for Computing Machinery, PO Box 12105, Church Street Station, New York, New York 10249. The dues (which include a subscription to the newsletter) are: $5.00/year for Members, associates and student members of the ACM (please include ACM member number); $13.00/year for non-ACM members.

For further information on SIGPC programs, contact Dr. Portia Isaacson, The Micro Store, 634 South Central Expressway, Richardson, TX, 75080; (214) 231-1096.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶

COMPUTER CLUBS

The Minnesota Computer Society meets the first Monday of each month at 7:30 p.m. at Brown Institute, 3123 E Lake St, Minneapolis, MN (unless announced otherwise). For further information contact: Minnesota Computer Society, c/o Jean Rice, Box 35317, Minneapolis, MN 55435.

Computer Amateurs of South Jersey meet at 7:30 p.m. the last Tuesday of each month at the National Park Municipal Building, 7 S Grove Ave, National Park, NJ. For additional information call (609) 541-1010 or (609) 541-8296.

Boston Computer Society meetings are held the fourth Wednesday of each month, except July, at the Commonwealth School, 151 Commonwealth Ave, Boston. They start at 7:00 p.m. and usually run till 10:00. For further information contact The Boston Computer Society, 17 Chestnut St, Boston, MA 02108; (617) 227-1399.

¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶ ¶