also including LOGO CAI TION

$2.95 in USA

99'er magazine™

May/June 1981 Vol. 1, No. 1

For Users of the Texas Instruments TI-99/4 and other 16-Bit TMS9900-Based Personal Computer Systems

THE BIRTH OF A NEW MAGAZINE

Business Projections in BASIC

Software Development with Pascal, Assembly & Extended BASIC

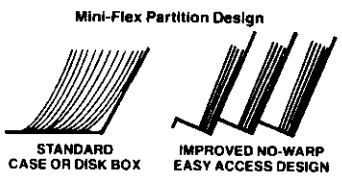
TI's Text-to-Speech Unveiled

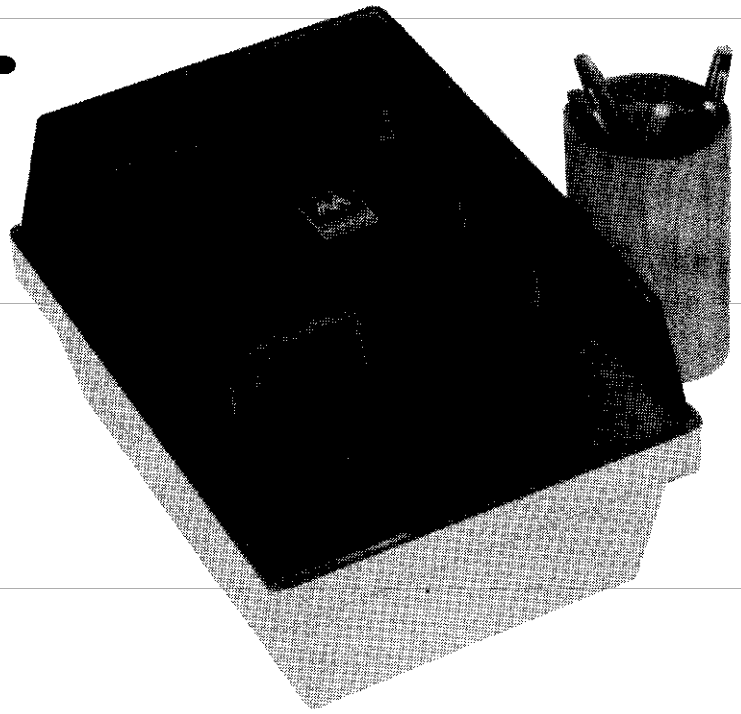
BASIC Battles — An Arsenal of Combat Games & Techniques

Learning with LOGO

Adding an External Keyboard

FAST and EASY RETRIEVAL OF THE DISKETTE YOU NEED...





The MINI-FLEX is styled with low sides and snag-proof edges... making removal and replacement of diskettes easy and safe.

The design of the MINI-FLEX case is a unique contribution to diskette safety. Many owners keep their disks in the carton they came in. Deprived of proper support, diskettes get into a slumped-over posture. But, the MINI-FLEX case is built with inclined sections molded into its base. The inclines keep diskettes standing upright, assuring no-sag, no-warp storage. There's no need to pack diskettes tightly. The MINI-FLEX case construction eliminates the need for file drawer type follower blocks, which can actually cause compression damage.

Warped or otherwise distorted disks result in data loss, and can even cause costly machine damage and "down" time. The inclined sections in the MINI-FLEX bottom prevent all that.

The MINI-FLEX is a stylish appointment in your office environment.

Its' color is impregnated...it wipes clean with just a damp cloth. The case's rounded corners won't snag clothing or mar furniture.

The MINI-FLEX is compact and convenient. It occupies less desktop space than a piece of typing paper, making for more efficient use of any work station. And since the top nests with the bottom, you don't need extra space to put it down when you take it off.

An office environment poses some of the worst dangers to the good health of your diskettes.

Dust, dirt, and smoke are in the air.
Heat and magnetic interference generated by office machines create more hazards. They can destroy valuable information stored on a disk's magnetic recording material—a risk your diskettes run between the time you store data and the time you want it back.

There are other possible problems. Unregulated vertical storage can lead to sagging, slumping or warping of diskettes. Horizontal stacking can cause compression damage. Bending or creasing a disk while sorting through a jam-packed box can result in data loss.

So, you can see, there are a lot of factors that can lead to losing valuable information...information that will always cost you time and money to replace.

MINI-FLEX doesn't conduct heat or electromagnetism like metal cases do.

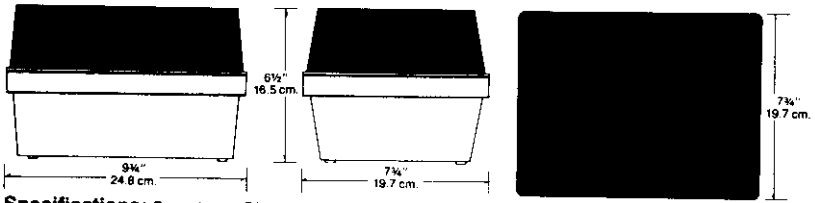
Spacious enough to store up to 50 mini-diskettes in their protective envelopes as recommended by all media manufacturers, yet small enough to conserve work space. The base nests in the lid for a compact configuration in use.

There's a modern modular concept behind the MINI-FLEX design. It's adaptable to any quantity of information...compact enough for smaller storage requirements...flexible enough for dividing even the largest libraries into convenient units, the way separate drawers do in a file cabinet.

Improper handling of diskettes can cause data loss. With the MINI-FLEX, handling is cut down to a minimum.

In storage, your diskettes are separated by heavy-duty partitions, they won't wear out. And they can be moved—or removed—as your file grows. When an operator searches for a specific disk, the partitions open naturally into V-shaped compartments. An operator can view the entire front of each disk's envelope without having to remove the disk from the case—or bend it for visibility. That makes for fast accurate search and instant identification with no danger of bending or creasing valuable disks.

. . .While Guarding Against Data Dropout Caused By Improper Storage and Handling.



Specifications: Capacity: 50 Diskettes with Envelopes
Shipping Weight: 3 Lbs./1.35 Kg. Partitions: 3 Color: Smoke and Sand 1 Package MINI-FLEX Guides

MINI-FLEX DISKETTE FILE

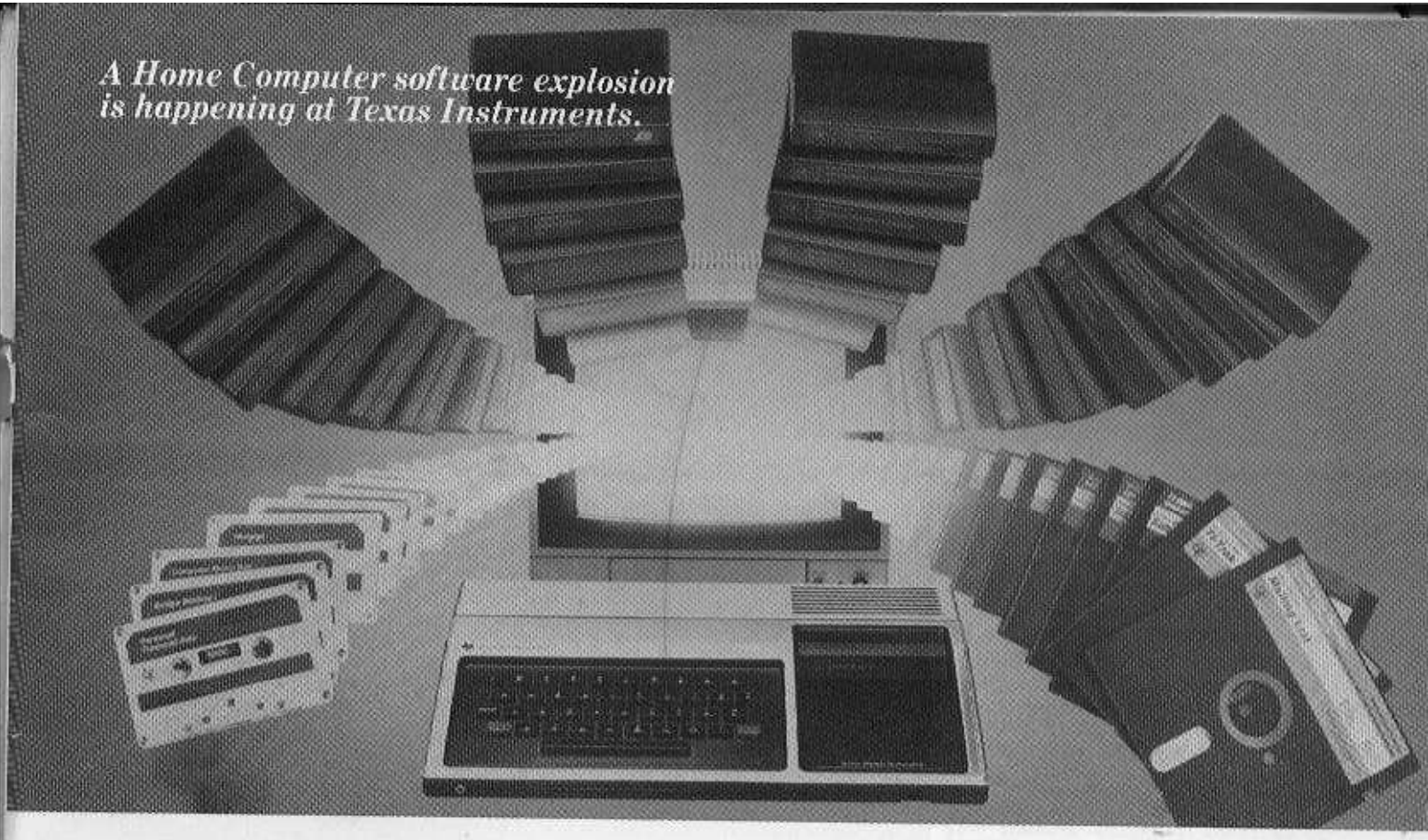
List Price: $29.95

☆ Special Price $24.95

Add $2.50 shipping and handling. Use the bind-in card in the back of the magazine for your convenience in ordering. Telephone orders accepted if charged to credit cards.

99er-ware™
INNOVATIVE PRODUCTS FOR TMS9900-BASED PERSONAL COMPUTING

P.O. Box 5537
Eugene, Oregon 97405
Tel. (503) 485-8796



Now there are hundreds of additional new uses for the Texas Instruments TI-99/4 Home Computer– something for every member of your family.

Our library of application programs for the TI Home Computer is growing rapidly. And that's great for TI-99/4 owners. The more programs to choose from, the greater the value and utility of this remarkable home computer.

Programs are available in three different, easy-to-use formats — Solid State Software™ Command Modules, diskette and cassettes — in a wide range of application areas: • Youth Education. • Home Management and Personal Finance. • Entertainment. • And much more.

Many independent software companies are also creating programs for the TI-99/4 Home Computer. Companies like: Scott, Foresman and Company; Micro-Ed, Inc.; Program Design; 99/4 National Users' Group; Creative Computing; Microcomputers Corporation; Image Computer Products; Instant Software; Charles Mann and Associates; and many more.

Get a copy of our Application Program Directory.

You can pick up a copy of our latest catalog of TI-99/4 software today at your nearest Texas Instruments Home Computer dealer.

To receive a copy by mail, call toll-free **1-800-858-4565.** In Texas call **1-800-692-4279.**

Congratulations to the winners of our Author Incentive Program contest.

We received many innovative and exciting application programs from individuals who entered our recent Author Incentive Program. Our thanks to all those who submitted entries.

First Prize Winner $3000.00
Charles M. Ehninger
Ft. Worth, TX
(Program: Household Inventory)

Second Prize Winner $1000.00/Each
Wayne O. Williams
Mesa, Arizona
(Program: Metric System Tutor)

Fred J. Baker
Motley, MN
(Program: Kepler Planetory Motion)

Norman S. Kerr
St. Paul, MN
(Program: Mendelian Inheritance)

Glen M. Kleinman
Toronto, Ontario CANADA
(Program: Addition Teaching Tools)

Jack Kirby
Brockport, NY
(Program: Math Remediation)

Third Prize Winners $500.00/Each
Melvin W. Mills
Hancock, MD

Colin James III
Denver, CO

Richard L. Loggins
Albuquerque, NM

Richard A. Ertl
Houston, TX

Alan Winkley
Oakland, CA

Suzanne M. Hawkinson
Lexington, MA

Charles W. Engel
Tampa, FL

Morris C. Burkhart
Alexandria, VA

Robert M. Klehn
Luling, TX

Chris and Craig Christianson
Tulsa, OK

Williams D. Christianson
Tulsa, OK

Don C. Rich, Jr.
Thomasville, NC

Donald E. Bertrand
Wallingford, CT

Janet Rieker
Los Altos, CA

John M. Anderson
Wilmington, NC

Richard I. Larson
Wilmington, NC

Robert J. Prandzinski
Northbrook, IL

Edward G. Blakeway
Raleigh, NC

Rajeev Jayavant
Lyndhurst, OH

John and Norma Clulow
Perrysburg, OH

Texas Instruments technology—bringing affordable electronics to your fingertips.

TEXAS INSTRUMENTS
INCORPORATED

221-40A

99'er magazine™

Publisher / Editor
Gary M. Kaplan

Technical Editors
William K. Balthrop
Mike Kovacich
G. R. Michaels

Contributing Editors
Norma & John Clulow
Steve Kemp
Mark Moseley
Regena
George Struble
Dennis Thurlow
Jerry Wolfe

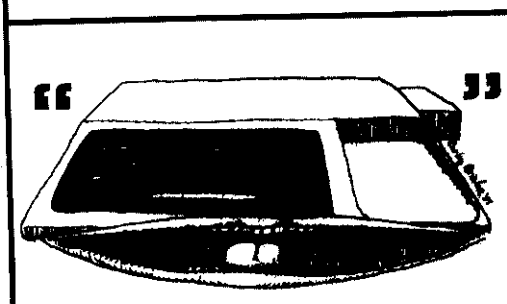
Production Manager
Pat Kaplan

Production & Design
Greg Davis
Rod Hoyle
Chuck Mantell
Corby Poticha
Joe Saputo
Cheryl Vigna

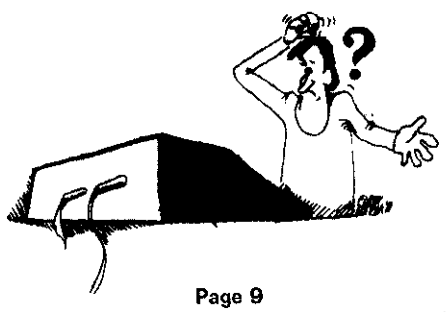
Circulation
Benjamin Kaplan
Pat Kaplan

Advertising
Patana Ratanapreux
Tel. 503-485-8796

Page 67



Page 30

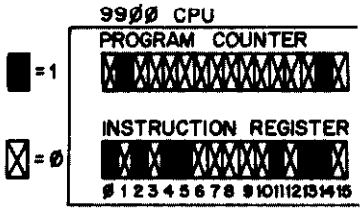


Page 9

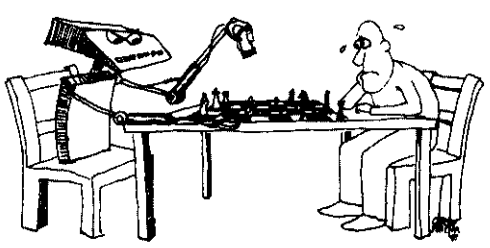
Table of Contents: May/June 1981

ON LOCATION™
THE INTERNATIONAL JOURNAL OF COMPUTER ASSISTED INSTRUCTION

Volume 1 No. 1



Page 48



Page 76

THIS ISSUE'S COVER:
The theme of this charter issue, *The Birth of a New Magazine,* is depicted by the TMS9900 microprocessor cracking open the egg from which the new magazine is emerging. The familiar TI-99/4 keyboard provides the support for the new publication's recursive vision of the exponential growth of 16-bit personal computing that will parallel the maturation and growth of the magazine itself. The cover was designed by Gary Kaplan and executed by Joe Saputo.

99'er Magazine is published bimonthly by Emerald Valley Publishing Co., P.O. Box 5537, Eugene, OR 97405. The editorial office is located at 2715 Terrace View Drive, Eugene, OR 97405 (Tel. 503-485-8796). Subscription rates in U.S. and its possessions are $15 for one year, $28 for two years, and $39 for three years. In Canada and Mexico $18 for one year, $34 for two years, $48 for three years. Other foreign countries $25 for one year surface, $40 for one year air delivery. Single copy price in U.S. and its possessions is $2.95, and $3.50 in Canada and Mexico. Foreign subscription payment should be in United States funds drawn on a U.S. bank. Application to mail at controlled circulation postage rates is pending at Eugene, OR 97401. POSTMASTER: Send address changes to **99'er Magazine,** P.O. Box 5537, Eugene, OR 97405. Subscribers should send all correspondence about subscriptions to above address.

Address all editorial correspondence to the Editor at **99'er Magazine,** 2715 Terrace View Drive, Eugene, OR 97405. Unacceptable manuscripts will be returned if accompanied by sufficient first class postage and self-addressed envelope. Not responsible for lost manuscripts, photos, or program media. Opinions expressed by the authors are not necessarily those of **99'er Magazine.** All mail directed to the "Letters to the Editor" column will be treated as unconditionally assigned for publication, copyright purposes, and use in any other publication or brochure, and are subject to **99'er Magazine's** unrestricted right to edit and comment. **99'er Magazine** assumes no liability for errors in articles or advertisements. Mention of products by trade name in editorial material or advertisements contained herein in no way constitutes endorsement of the product or products by **99'er Magazine** or the publisher unless explicitly stated.

PUBLISHER'S MESSAGE

By Gary M. Kaplan

Welcome to *99'er Magazine* and the world of 16-bit personal computing. Since this is a new publication, I'd like to take this opportunity to introduce myself and tell you something about what you'll find in this issue, and what you can expect to find in future issues of this magazine.

As both publisher and editor, it is my responsibility to see that what you want to read in *99'er Magazine* does, in fact, get published. We want you to always remember that *this* magazine is written for *you*. Please feel free to call or write me any time with your suggestions and comments.

We at *99'er Magazine* want to sincerely thank our charter subscribers for their support, faith, and best wishes. Thanks are also due to our contributing authors -- those whose articles appear in this issue, as well as those whose articles were accepted for publication, but because of space limitations in this issue, had to be postponed. We've attracted a talented group of authors, and I know that we are all going to learn a lot from them in the months ahead. Incidentally, three of our contributing authors (Charles Ehninger and Norma & John Clulow) have won prizes in the recent Texas Instruments software contest. We're very proud of them.

Our advertisers also deserve special thanks. Their sight-unseen support and faith in *99'er Magazine* has been especially gratifying and encouraging. The financial health of a new magazine is largely dependent on its advertising support, and judging by the warm welcome we have received from current and future advertisers, it is evident that *99'er Magazine* is off to a good start.

As more advertisers become aware of this new medium and the readership it serves, we all will benefit by having access to (1) a *larger* variety of new products that are developed especially for this market, (2) *better quality* products resulting from a more competitive market, and (3) a larger, *monthly* magazine that is made possible by an increase in advertising revenue.

In extending my appreciation to everyone who made this publication possible, I certainly cannot overlook the cooperation we have received from Texas Instruments and the various users groups around the country. Everyone involved was supportive in disseminating word of our forthcoming magazine.

99'er Magazine is indeed fortunate to already have a dedicated staff, talented authors, and supportive readers and advertisers. What we now need to turn this winning combination into a smashing success is a dynamic increase in the number of subscribers and single-copy sales. Publishing economics favor larger print runs: The more copies you come off the press with, the lower the unit production cost; this, in turn, allows for a more elaborate magazine -- extensive use of color, special bind-in supplements, and a "fatter" editorial product.

The fastest way to accomplish this is with your help. If each one of our subscribers were to find just *one* additional person to subscribe -- by giving a friend or associate one of the magazine's bind-in subscription cards to use -- we'd be well on our way to achieving our projected subscription growth far ahead of schedule. Keep in mind that many potential readers of this magazine will not yet have a 9900-based personal computer system. For example, your friends, business associates, and neighbors might be interested in microcomputers and want to know more. Or, if they already have equipment, they might want to leave their 8-bit world behind and step up to the many advantages of our 16-bit family. *99'er Magazine* would also make interesting reading and be a valuable resource in a school or library. And don't forget to let your local computer shops know about us. Have them write or call for information on our bulk subscription plan for single-copy sales.

The first thing you have probably noticed about *99'er Magazine* is that it's not only *99'er Magazine*! We've grouped together the articles concerned with computer-assisted instruction (CAI) in a special "magazine within a magazine" and called it *On LoCAltion*. It will include news, applications, and activities from all around the United States and abroad. I'd like to encourage our readers outside the U.S. to submit articles on how the Texas Instruments TI-99/4 is being used as an educational device. In this way, *On LoCAltion* can truly become the *International Journal of Computer Assisted Instruction* -- the first publication of its kind to provide a medium for the international exchange of practical CAI applications.

The second thing you'll notice about *99'er Magazine* is that there's no shortage of articles! We've included over two dozen entrees on the menu. You can choose from tutorials, software listings, product reviews, "how-to" articles, book excerpts, on-the-scene reports, and explanations of state-of-the-art technologies and products. Whether your interest is learning how to write programs, modifying your computer hardware, playing computer games, composing music, improving your chess prowess, learning a new computer language, accessing printer graphics, helping your children with their homework, developing software to sell, or putting your computer to work in your business . . . we have something in the magazine for you. Our goal in every issue is the proverbial-yet-elusive "something for everyone." And "everyone" means *you* . . . so let us hear what you want to know.

In our second issue (July/August), we'll be starting a "Letters to the Editor"

column. With your input, there will be some interesting correspondence to publish. Additionally, you can expect to see book reviews, news from the users groups, and a "99'er Bulletin Board" for posting of your non-commercial messages. (You can start sending them in to 99'er Magazine/Bulletin Board, P.O. Box 5537, Eugene, OR 97405.)

Other regular columns that will soon be started include:

- *Microcomputing for the Handicapped*
- *The 99'er Ham Meet* (for promoting the use of microcomputers in amateur radio)
- *The Micro Librarian* (covering library usage of the TI-99/4)

Several exciting projects are presently underway at *99'er Magazine*, and I thought that this first Publisher's Message would be a good time to give you some idea of what's coming. First of all, we will be acting as a clearinghouse for the dissemination of information and activities concerned with the LOGO language. LOGO users groups can use the pages of *On LoCAItion* to share their discoveries and experiences with others around the country. But education isn't the only thing on our minds. We've also planned several book publishing projects including *What You Always Wanted to Know About the TI-99/4 . . .* (there will be a user questionnaire in our next issue), *Extended BASIC Programming on the TI-99/4*, and *9900 Assembly Language Programming on the TI-99/4*.

In the coming months ahead, the magazine's R&D department, *99'er-ware*, will be releasing several new hardware and software products. And plans are also underway for providing TEXNET users with the ability to access and download the software listings in this magazine. In the July/August issue, we'll provide you with more details about the soon-to-be-available TEXNET, but in the meantime, read the article on page 30 to learn about its text-to-speech capabilities.

Speaking about software, we've got some great surprises for you. As you explore the magazine, type in the listings, and run the programs, you'll see what I mean. All the programs are well documented, and have been chosen for either their entertainment value or utility, as well as for their instructive value in demonstrating certain programming techniques. Incidentally, if you see a program listing with a small picture of a floppy disk enclosed in a circle with an X through it, the program as listed will completely fill available memory and cannot be run with the disk controller turned on (even with the CALL FILES (1) command executed).

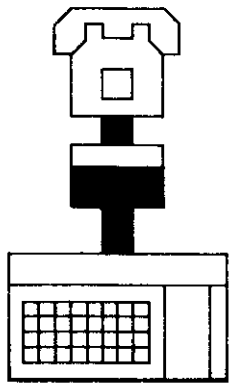
We have a pleasant surprise for TI-99/4 users who have been handicapped by having to use a cassette recorder with a remote control jack that isn't compatible with the TI-99/4. It isn't much of a problem unless you want to do file processing on cassette — where it's necessary to have the tape recorder automatically turned on and off under program control. We wanted to give you a music text editor program (page 68), but knew it would be a problem for many readers using recorders with an inoperative remote control jack. So we therefore went ahead and produced our own inexpensive TI-SETTE™ adaptor to make the incompatible compatible. You'll find it described at the end of the music data file on page 70. In future issues, we expect to do more with file processing on cassettes, so even if you're not interested in the music software, it might be a good idea to get set up for remote control operation now.

INSTRUCTIONS FOR BEGINNERS

The editorial staff at *99'er Magazine* has tried to put together a well-balanced issue — one written on several levels, for the novice and pro alike. We realize that many beginners to microcomputing will *not* be able to understand everything that's in the magazine. My advice to beginners, therefore, is to put away the issue after you feel you've gotten all you're presently capable of absorbing from it. Later, several issues down the road, go back and re-read any material that gave you trouble the first time through. Many times you'll be surprised to find that you are later able to understand it. This is all part of the natural learning process, and we at the magazine will try our best to guide you along on that journey of knowledge and discovery.

Readers With Access to The Source May Send Electronic Mail To:

ID TCU975



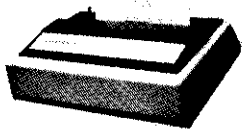
Orange Micro

"THE COMPUTER PRINTER SPECIALISTS"

UP TO 25% DISCOUNTS! — SAME DAY SHIPMENT!

CENTRONICS 737 (RADIO SHACK LINE PRINTER IV)

Word Processing Print Quality



• 18 x 9 dot matrix; suitable for word processing • Underlining • proportional spacing • right margin justification • serif typeface • 50/80 CPS • 9½" Pin Feed/Friction feed • Reverse Platen • 80/132 columns

CENTRONICS 737-1 (Parallel) ..... (List $995) **$765**
CENTRONICS 737-3 (Serial) ..... (List $1045) **$815**

VISTA — C. ITOH

Daisy Wheel Letter Quality

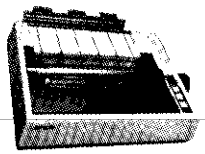
• 25 CPS (Optional 45 CPS) • Typewriter quality • Centronics parallel • RS 232 Serial (Optional) • Proportional spacing • Bidirectional • Programmable VFU • Self test • Diablo compatible • Friction feed (Optional tractors) • 136 printable columns. • Manufactured by C. ITOH.

VISTA V300 (C. ITOH) ..... (List $1895) **$ Call**

OUR BEST BUY!

EPSON MX80/MX70

Low-Priced Professional Print Quality



• 9 x 9 dot matrix • Lower case descenders • 80 CPS • Bidirectional, Logic seeking • 40, 66, 80, 132 columns per line • 64 special graphic characters: TRS-80 Compatible • Forms handling • Multi-pass printing • Adjustable tractors

EPSON MX80 ..... (List $645) **$Call**
EPSON MX 70 Dot graphics, 5 x 7 matrix ..... (List $450) **$Call**

ANACOM

Low Cost, High Speed, Wide Carriage

• 9 x 9 dot matrix • Lower case descenders • Wide carriage • Adjustable tractors to 16" • 150 CPS, Bidirectional, Logic Seeking

ANACOM 150 ..... (List $1350) **$ Call**

ANADEX

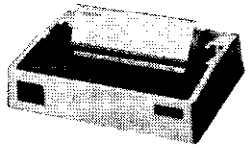
Dot Graphics, Wide Carriage

• 11 x 9 dot matrix; lower case descenders • Dot resolution graphics • Bi-directional, logic seeking • Up to 200 CPS • RS 232 Serial & Parallel • Forms control • X-ON/X-OFF • Up to 6 part copy.

ANADEX 9501 ..... (List $1650) **$1350**

OKIDATA MICROLINE SERIES

TRS-80 Graphics Compatibility, Friction Feed



• 9 x 7 dot matrix • 80 CPS • 80, 132 columns — 64 shapes for charts, graphs & diagrams • Double wide characters • 6/8 lines per inch • Up to 3 part copy • Friction & pin feed • 200 M character head warranty

OKIDATA M82 Bidirectional, Forms handling ..... (List $960) **$750**
OKIDATA M83 Wide carriage, 9 x 9 dot matrix ..... (List $1260) **$1050**

NEC SPINWRITER

High Speed Letter Quality

• 55 CPS • Typewriter quality • Bidirectional • Plotting • proportional spacing.

5510-5 RO, Serial, w/tractors ..... (List $2995) **$2625**
5530-5 RO, Parallel, w/tractors ..... (List $2970) **$2599**

TELEVIDEO CRT'S

AT DISCOUNT PRICES!

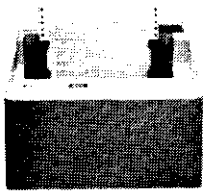
TVI 912C
TVI 920C
TVI 950

Please Call Toll Free
Prices are too low to advertise

QUANTITY PRICING AVAILABLE

IDS PAPER TIGERS

Dot Resolution Graphics, quality print, speed



• 7 wire printhead (445); 9 wire printhead (460) with lower case descenders • Over 150 CPS • bi-directional, logic seeking (460) • 8 character sizes; 80-132 columns • Adjustable tractors • High-resolution dot graphics • Proportional spacing & text justification (460).

IDS 445G 7 wire printhead, graphics ..... (List $895) **$ 750**
IDS 460G 9 wire printhead, graphics ..... (List $1394) **$1150**
IDS 560G 9 wire, wide carriage, graphics ..... (List $1794) **$1590**

PRINTERS

MALIBU 165 wide carriage, graphics, letter quality .. (List $2495) **$ 1975**
QUME 5/45 typewriter quality ..... (List $2905) **$ 2559**

INTERFACE EQUIPMENT

CCS APPLE PARALLEL interface & cable ..... **$ 150**
APPLE II - EPSON MX80 parallel interface board & cable ..... **$ 100**
MICROTRONICS Atari parallel interface ..... **$ 69**
ATARI 850 interface module, serial/parallel ..... **$ 199**
TRS-80 CABLES to keyboard or Exp. interface ..... **$ Call**
NOVATION D-CAT direct connect modem ..... **$ Call**

CALL FOR FREE CATALOG

(800) 854-8275

CA, AK, HI (714) 630-3322

At Orange Micro, we try to fit the right printer to your application. Call our printer specialists for free consultation.

TELEPHONE ORDERS: Mon.-Fri. 8:30 - 5:00
The Orange Micro Printer Store (Retail):
Mon.-Fri. 10:00 - 6:00, Sat. til 4:00

Phone orders WELCOME; same day shipment. Free use of VISA & MASTERCARD. Personal checks require 2 weeks to clear. Manufacturer's warranty included on all equipment. Prices subject to revision.

Orange Micro, Inc.
3150 E. La Palma, Suite I
Anaheim, CA 92806

HOW TO WRITE YOUR OWN PROGRAMS

Part 1 : Using Flowcharts To Outline a Solution

By James J. Dugan

Sitting down in front of your TI-99/4 and running packaged software may at times stimulate your desire to try out some programming of your own. If you were fortunate enough to have taken some courses in computer programming or have had some practical on-the-job training, you can probably just type up some lines and have the computer do what you want. However, if your formal training has not included this experience, you may soon find the frustration of not knowing where to start, too much to bear. Well, take heart! In this article, I am going to present some basic information on how you can begin programming on your own.

A Framework for Writing Programs

Computer programming is an exercise in reasoning and logic. Before programmers can develop software to do specific jobs, they must plan their attack on the individual elements that are inherent to those jobs or problems. This is where it is helpful to have in mind a general framework as to how to go about solving each problem.

This general framework could take a number of different forms. Most will, however, contain similar steps. These steps can be described as follows.

1. Define the Problem

Initially, it is necessary to have a good understanding of exactly what you want the program to do. If it is possible, try to express the problem in a simple thought or sentence stating the intended outcome of the programming effort. Defining the problem in this manner may not only save you time, but may also help focus your efforts.

1031 S. Patton Ct., Denver, CO 80219

2. Outline the Solution

This step is the primary purpose of this article. We'll get back to examine this step in more detail later.

3. Select the Algorithm

Many problems requiring a computer for solution depend on certain mathematical algorithms that are required in the calculation of the desired solution. For those who are puzzled by the word "algorithm," mathematicians and math teachers use this word to refer to the specific method of solving a certain kind of mathematical problem. For example, you may have been taught to subtract whole numbers by placing the larger number on top, the smaller on the bottom and to borrow when necessary. This is but one possible algorithm for subtraction. In general, you can either locate those algorithms that are necessary from published sources, or you may need to design your own. In either instance, the simpler the algorithm, the better.

4. Writing the Program

Many people believe the writing or "coding" of a program is what computer programming is all about. Actually, this step is just *one* in a series of steps. Prior planning (as detailed in steps 1-3, above) is absolutely essential before the actual writing of the program can begin. And inherent in the writing of the program must be a reasonable understanding of the computer language you will be using.

5. Debugging

Once you have typed the program into the computer, it is necessary to run it to determine if and what difficulties exist. You will seldom write an error-free program on the first draft. Trying to locate and correct those "bugs" can be frustrating. This is where some of the TI-99/4 built-in features help tremendously.

6. Validating the Program

In this step, you intentionally try to locate situations in which the program yields inaccurate or undefined solutions.

7. Documentation

It is a good idea to document or record the characteristics of the program for communication of the program's intent, its algorithms, and specifications. In its simplest form, documentation of what each variable represents is helpful. Incidentally, when buying a program, the author's documentation (or lack thereof) can often be a good indication of the quality of the program.

Outlining the Solution

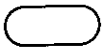
The development of an adequate outline is the most critical step in writing a program. Many of us novice dabblers in the art of programming seem to fail in developing an acceptable (let alone, adequate) outline. My intention here is to demonstrate to you some elementary outlining techniques—in the hopes that we, the dabblers, may be able to improve our lot in the somewhat puzzling world of bits, bytes, and bugs.

Flowcharting

There are a number of methods available for outlining a solution to a problem. Of those used in computer programming, basic flowcharting is one of the simplest and easiest.

To introduce you to the flowcharting method, let's first look at some of the symbols used.

1. START AND END SYMBOL



This symbol is used to indicate both the beginning and the end of the program.

2. INPUT–OUTPUT SYMBOL

The input-output symbol is used to indicate where the user of the program will need to supply a piece of data or where a calculation will be printed out for the user.

3. COMPUTATION OR ASSIGNMENT SYMBOL

This symbol is used to indicate where computations or assignments of values to variables will occur.

4. DECISION SYMBOL

The decision symbol is used to indicate where a "yes" or "no" or "true" or "false" decision point is located.

5. STOP SYMBOL

This symbol is used in some programs to indicate a termination point if this point is different from the end point of the program.

There are other symbols that can be used according to your needs. Also, remember that no rules exist to stop you from developing your own symbols.

Toward a Workable Technique

The outlining strategy that works best for me is to start off simple, and then increase the complexity of my outline until it does what I want it to do. My approach includes (1) writing a sentence that defines the problem I want to solve, (2) preparing an informal outline, (3) developing a more complex flowchart, and then (4) writing the program. To demonstrate how this approach leads you to developing a better program, let's take a look at some examples.

EXAMPLE 1

For our first example, let's write a program that will add two numbers together and print their sum. We will design the program so that we may input two numbers from the console. This is called an *interactive* program, in that the user must input the values to be assigned to the variables. Following the approach presented, we first define the problem:

Step 1. Definition of the Problem:

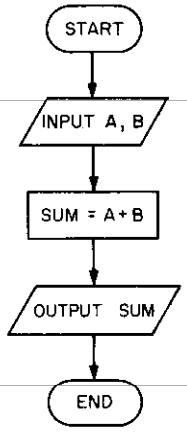
The program will take two numbers being input from the console, add them together and print the sum.

Step 2. Informal Outline:

1. Start
2. Input two numbers, A and B
3. Add A and B
4. Output the sum of the numbers

Step 3. Flowcharting:

Using the flowcharting symbols, the solution is further developed.



Explanation of the flowchart

In examining the flowchart, the "flow" is evident in the continuous line running from the initial start symbol to the final end symbol. The input symbol shows that two values are requested, with the first input value being assigned to the variable A, and the second to B. The addition of the two numbers and the assignment of their sum to a variable occurs inside the computation symbol. The value of the sum is then output and the program ends. The algorithm necessary for the solution is shown.

Now that the problem has been outlined, we proceed to write or code the program.

Step 4. Coding:

```
100 REM **ADDITION PROGRAM**
110 INPUT A,B
120 LET S=A+B
130 PRINT S
140 END
```

Explanation of the program.

The program shows how the original intent is followed.

Line 100 contains a REM statement, allowing us a means of identifying the program.

Line 110 allows the user to type in the two numbers to be added.

Line 120 assigns the value of A plus B to the variable S.

Line 130 prints the value of S.

Line 140 ends the program.

Since my primary intent here is to explain how an outline is developed and used, I will not explain the TI BASIC command statements, but assume that readers of this article have already read most of the TI *Beginner's BASIC*, the book that came with their computer.

EXAMPLE 2

For a more complex example, let's develop a program that will select and print the larger of two input values.

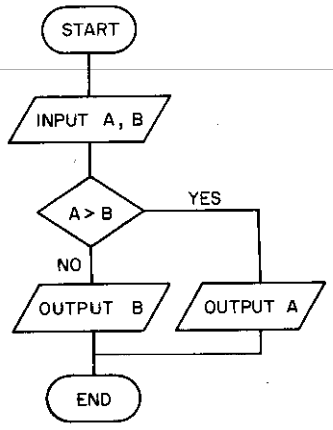
Step 1. Defining the Problem:

Given two numbers, the program will select the larger of the two and print it.

Step 2. Informal Outline:

1. Start.
2. Input two numbers, A and B from the console.
3. Compare number A with B. If A is larger than B, print A. If A is not larger than B, print B.
4. End.

Step 3. Flowcharting:



Explanation of the flowchart.

The flowchart begins with the start symbol. The two values are then input. In the decision box, a comparison of the value A with B takes place. If the statement A > B is true, the computer is instructed to bypass the output B box, and output the value of A. If the statement is false, the computer continues down the chart to output the value assigned to B. The program then ends.

Step 4. Coding:

```
100 REM  **PRINTS LARGER OF TWO
    NUMBERS**
110 INPUT A,B
120 IF A>B THEN 150
130 PRINT B
140 GOTO 160
150 PRINT A
160 END
```

Explanation of the program.

Line 100 is a REM statement used to identify the program.

Line 110 allows the user to input the two numbers to be compared.

Line 120 is the algorithm used to

compare the two numbers. If the statement A is greater (>) than B is true, the computer is then instructed to go to line number 150 and print A. If the statement in line 120 is false, the THEN condition does not hold, and the computer continues to the next line.

Line 130 prints the value of B, as it must be the larger.

Line 140 is used to direct the computer to go to line 160. Without this line, the computer would print the value of B, then the value of A. This is, of course, not what we wanted.

One difficulty exists with this program. If A and B are equal, the program will not be able to distinquish the two. (If this arises, B will be printed.) This difficulty could be corrected by allowing another step for this situation where value A equaling value B could be displayed.

EXAMPLE 3

Let's take a look at one more simple example. This time we'll try writing a program that will print out all the squares of the integers between 1 and 99, inclusive of the two boundaries.

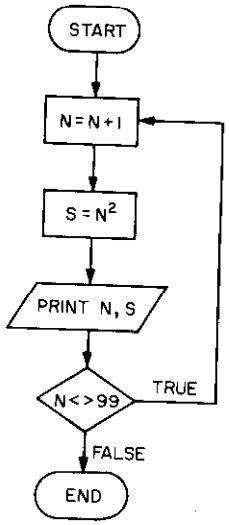
Step 1. Defining the Problem:

The program will make a list of all the squares of the integers between 1 and 99, inclusive.

Step 2. Informal Outline:

1. Start.
2. Let N be a variable whose initial value is 1.
3. Compute the value of N^2, and let the result be the value of S.
4. Print N and S on one line of the screen.
5. If the value of N is 99, then end the program. Otherwise, go to step 6.
6. Add 1 to the value of N and then go back to step 3.

Step 3. Flowcharting:



Explanation of the flowchart

After starting the program, the variable N is increased by 1. As the TI BASIC will automatically set the initial value of N to zero, using the statement N=N+1 will set the first value of N to 1. Next, the square is calculated. Both the integer and its square are then printed. The next step checks to see if N is equal to the upper boundary of 99. If N is equal to 99, the computer is instructed to end the program. If N is not equal to 99, the program loops back to add one to the value of N and continues.

Step 4. Coding:

```
100 REM **SQUARES**
110 LET N=N+1
120 LET S=N^2
130 PRINT N,S
140 IF N<>99 THEN 150 ELSE 110
150 END
```

Explanation of the program.

Line 100 is the REM statement.

Line 110 adds one to the variable N.

Line 120 computes the square.

Line 130 prints the integer N and its square S.

Line 140 determines if the value of N is 99. If N is equal to 99, the computer goes to line 150 and ends the program. If N is not equal to 99, the computer returns to line 110.

Line 150 ends the program.

Now that we have seen the use of the outlining technique in some rather elementary program examples, let's get serious and try something more challenging.

EXAMPLE 4

Let's try writing a program to test our recall of a series of digits. With each correct matching of the series of digits, We'll instruct the computer to add another digit to the series.

Step 1. Defining the Problem:

The program will display a series of digits of increasing length and ask the user to recall the correct order of the digits.

It might be helpful to place some limits on the program to further qualify what we want it to do. This can be done in the informal outline.

Step 2. Informal Outline:

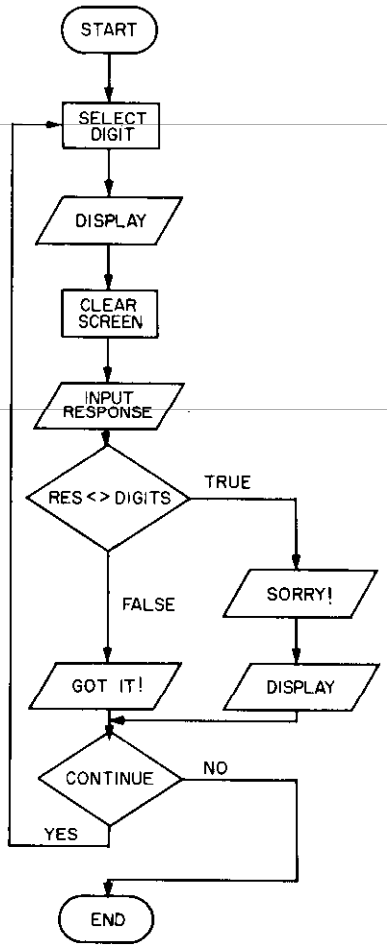
1. Start.
2. Have the computer select a random digit.
3. Display the series of digits for a short time.
4. Clear the screen.
5. Ask for a response from the user.
6. Compare the response to the series of digits.

7A. If the response is correct, congratulate and ask if the user wants to continue.

7B. If the response is incorrect, show the correct series of digits and ask if the user wants to continue.

8. If the user wants to continue, have the computer select another digit and add it to the end of the previous series.
9. If the user does not want to continue, end the program.

Step 3. Flowcharting:



Explanation of the flowchart.

After the program starts, a random digit is selected. The series of digits is displayed and the screen cleared. The user is then asked to respond. If the response is correct, the computer offers congratulations with a GOT IT! message, then asks if the user wants to continue. If the response is incorrect, the computer says SORRY!, then displays the correct response. The user is then asked if he wants to continue. If the answer is yes, the computer loops back to the random digit selection box, tacks on an extra digit to the string, and continues. If the answer is no, the program ends.

Step 4. Coding:

In coding the program, there are some specific features of TI BASIC that can

help us. We'll use the RANDOMIZE statement to pick the random digit, and the RND function to get a better selection of the digits. The necessary algorithm is developed as follows: Take the number generated by the RND function and multiply it by 10; then round the number using the INT function. This algorithm is expressed as INT(RND*10).

To display the series of digits and to allow the computer to compare a user's response to the original series, it might be easiest to translate the digits to a numeric string. We can perform the translation by using the STR$ function.

Explanation of the program.

Lines 100-170 are REM statements.

Line 180 is the RANDOMIZE statement.

Line 190 begins the selection of the random digit. With this statement, the computer will display a series of digits starting with a single digit and extending to an upper limit of 25 digits maximum.

Line 200 is the algorithm for selecting the random digit and assigning its value to the variable A.

Line 210 translates the digit selected to a numeric string. The line will also function in adding each digit selected to the end of the previous series of digits.

Line 220 clears the screen.

Lines 230-250 present the series of digits and tells you how much time you are allowed to study the series.

Line 260-270 time the digits being displayed. Going through the FOR . . . NEXT loop takes about five seconds.

Line 280 clears the screen.

Line 290 directs the computer to jump to line 250 The line is intended to get us out of the FOR I . . . NEXT I loop without disrupting it.

Line 300 continues the FOR I . . . NEXT I loop.

Line 310 ends the program.

Line 320-330 prompt the user to respond.

Line 340 compares the response to the series of digits. If the response is incorrect, the THEN condition directs the computer to line 380 which is the SORRY! comment. If the response is correct, the THEN condition does not hold, and the computer goes to the next line (line 350).

Line 350 clears the screen.

Line 360 congratulates the user.

Line 370 directs the computer to go to line 390, bypassing the SORRY! comment.

Line 390 asks if the user wants to continue.

Line 400-410 check to see if the user is interested in continuing.

Line 420 returns the computer back into the FOR I . . . NEXT I loop.

Line 430 ends the program.

FINAL COMMENTS

Once you've had a chance to use this approach—defining the problem, doing an informal outline, flowcharting, and then coding—in a project of your own, programming your computer will no longer be as forbidding and mysterious as you first thought.

Before attempting programs of your own, you may want to try a little exercise. Add the following features to the previous program:

(1) Allow the user to choose how much time the digits are displayed on the screen.

(2) If response is correct, play a 3-note chord.

(3) If response is incorrect, play one note of noise, and print a screen message that tells how many digits were contained in the largest number correctly guessed.

The solution, in the form of an expanded program will appear in the next issue.

```
100 REM  ******************
110 REM  ***               **
120 REM  ***NUMBER MATCH**
130 REM  ***               **
140 REM  ******************
150 REM
160 REM   BY JAMES DUGAN
170 REM  99'ER VERSION 5.81.1
180 RANDOMIZE
190 FOR I=1 TO 25
200 LET A=INT(RND*10)
210 LET MSG$=MSG$&STR$(A)
220 CALL CLEAR
230 PRINT "HERE IS THE NUMBER"::
240 PRINT MSG$::::
250 PRINT "YOU HAVE FIVE SECONDS":
    "TO STUDY IT"
260 FOR DELAY=1 TO 1500
270 NEXT DELAY
280 CALL CLEAR
290 GOSUB 320
300 NEXT I
310 GOTO 430
320 PRINT "TYPE THE NUMBER"
330 INPUT RES$
340 IF RES$<>MSG$ THEN 380
350 CALL CLEAR
360 PRINT "GOT IT!"
370 GOTO 390
380 PRINT "SORRY! THE NUMBER WAS:"
    ;MSG$
390 PRINT "DO YOU WANT TO CONTINUE
    ?    ""Y"", OR ""N""."
400 INPUT ANS$
410 IF ANS$<>"Y"THEN 430
420 RETURN
430 END
```

An External Keyboard for the TI-99/4

By Mike Kovacich

Although the unique keyboard on the TI-99/4 console has its definite advantages in an educational and game-playing environment, a more standard, terminal-like keyboard would be more suitable for data and word processing, communications, and program development. In this article, I will present a method of attaching an external keyboard to your 99/4 console. In future articles, I will explore several possible hardware configurations for this modification, and weigh the advantages and disadvantages of each, so that you can decide which method is most appropriate for you. *None* of the methods that I will employ will involve the disabling of the console keyboard. You'll always retain "dual control" of your computer, and will never have to sacrifice the overlay advantages and "friendliness" of the original.

In Figure 1, you can observe the eighteen connections to the keyboard. Ten of the lines—for explanation purposes I will call *scan* lines—originate from the 74LS156. Actually, there are five scan lines that are connected to one half of the keyboard, and then are wired in parallel to the other half of the keyboard. There are also four lines on each side of the keyboard that I will call *interrupt* lines. These lines are connected to the interrupt inputs of the TMS9901 I/O controller. I've numbered all eighteen lines at the keyboard, and have made a chart (Table 1) to distinguish between the scan and interrupt lines.

The basic operation of the TI-99/4 keyboard is quite simple. The TMS9901 outputs 3 bits of data to the inputs of a decoder (74LS156); the selected decoder output is then inverted and forced into a *low* state. As a result, only *one* of the scan lines is *low* at this time. All of the scan lines are normally *high* because of pull-up resistors, so only *one* scan line at a time is selected as a *low* level. The 3-bit data output from the TMS9901 is continuously changing—thus, the "scanning" effect.

When a keyswitch is closed, the corresponding row (interrupt line) is brought *low* when the 9901 selects the corresponding column (scan line).

An interrupt is generated and tells the microprocessor that a key was depressed. The microprocessor looks at what interrupt line caused the interrupt and which scan line was selected at the time of the interrupt. The internal software determines what value to give to that key.

Now that you basically know the operation of the keyboard, you are ready to connect an external keyboard to your TI-99/4.

Early in my planning stage, I decided to mount a 25-pin, D-type (DB/25) female connector to the back of my console (see Photo 3). The interrupt and scan lines were brought to this connector; so different external keyboard configurations could be tried without having to take apart the console over and over again. I also decided to connect the +5V,

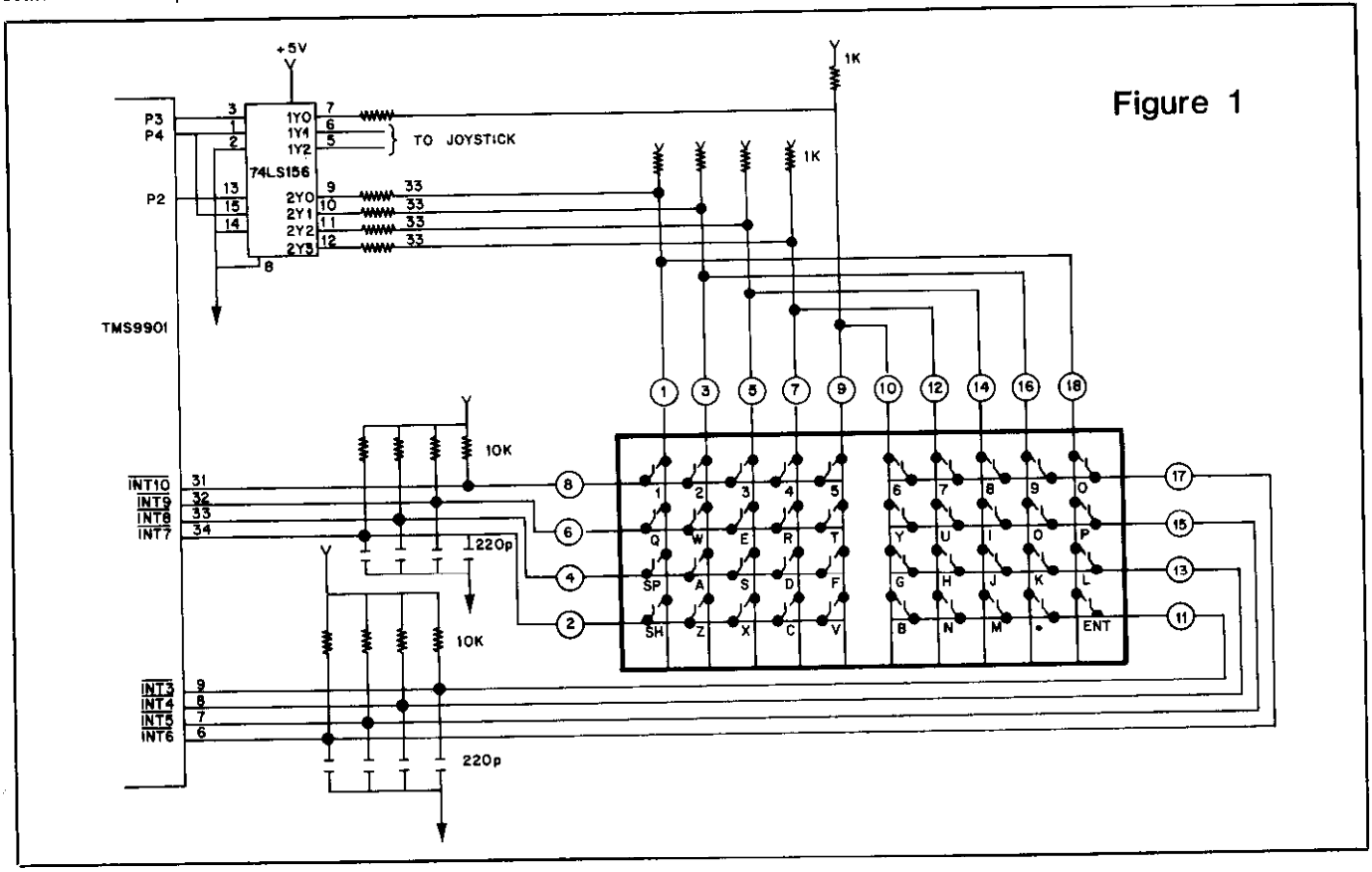
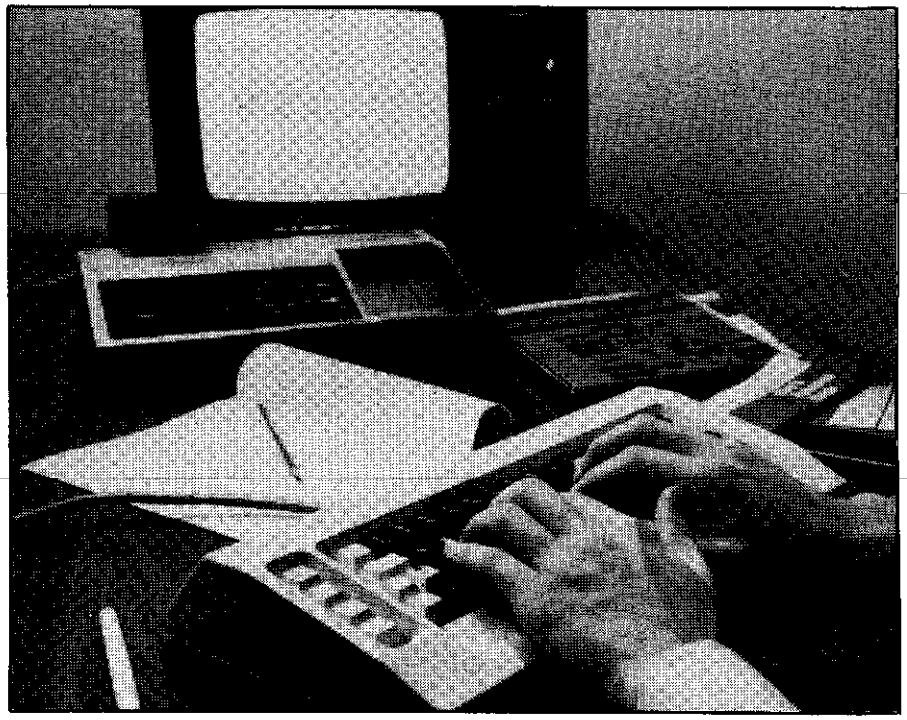


Figure 1

Photo 1



None of the methods involve the disabling of the console keyboard. You'll always retain dual control of your computer, and will never have to sacrifice the overlay advantages and friendliness of the original.

–5V, +12V, GND and external speaker to this connector; so an external power supply would not be necessary.

I have provided a table of the pin-outs I used, and suggest that you use the same for future compatibility. Besides the female DB/25 chassis connector, the only other parts needed to modify the console are an 18" length of 13-conductor color-coded ribbon cable, and four color-coded 18" lengths of #18 wire. If you want to mount the DB/25 connector on the back panel, the plastic housing will have to be physically modified. This means that a drill, hacksaw, some files, as well as some type of glue will all be necessary.

With all its cables removed (including power), place the unit with the keyboard face down. Put a towel underneath to prevent the front surface from being scratched. The bottom shell can be easily separated from the main body by removing the screws that hold it in place. It will be necessary to dismantle the power switch assembly to remove the bottom shell. Once the bottom is removed, study the power switch assembly, and practice putting it together and taking it apart a few times. If you have dared to go this far and your heart is thumping too fast, don't worry. A lot of design work has gone into the physical mounting and electrical shielding of the logic board; nothing you have done so far could damage the computer. (To play it safe, however, you might want to reconnect the power and video, and go through a small test procedure after each major step.) This is all the dismantling of the console you will have to do.

With the bottom off, all of the points that you will solder to are fully exposed. Please refer to Photo 2, and note where the wires from the connector are soldered to the logic board. It is difficult to see all of the solder points from the photograph; so I've drawn a pictorial diagram (Figure 2) to identify each solder point. As they say in all the pirate movies, "An X marks the spot." By the way, you'll only need to bring thirteen of the eighteen keyboard lines to the DB/25 connector. This is because five wires on the scan lines are duplicated. There's no need to wire them up as long as you remember to reduplicate them at the new keyboard. The numbers in Figure 2 identify the keyboard lines (see Figure 1 and Table 1).

As mentioned earlier, you'll have to modify the plastic housing if you want a secure connection. So get your hacksaw ready and refer to Figure 3. Measure a 3" piece of the leaf on the backside of the bottom shell (Figure 3a), and carefully cut it out. The DB/25 female connector is mounted on this piece of plastic. Drill and file the plastic piece until the connector properly fits in place. The ½" measurement in Figure 3b is critical because of the limited room inside the 99/4 console.

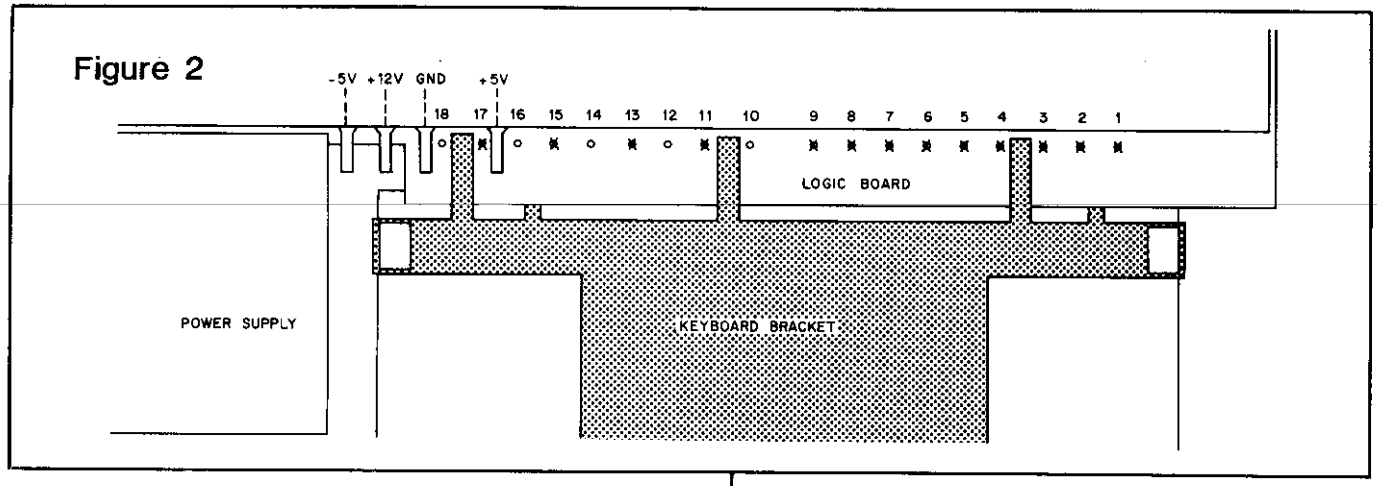
You may now proceed to solder the leads for the keyboard lines and power supply—first to the DB/25 connector, and then to the appropriate points on the logic board. It's a good idea to write down (on a separate piece of paper) what color wire is soldered to what pin on the connector, so you can easily tell them apart. Use heat-shrink tubing on every pin to eliminate any possible shorting at the connector end. Once the wires have been soldered to the connector using the pin-outs of Table 1, route the group of wires underneath the logic board as in Photo 2. Cut off excessive wires lengths and proceed to solder the wires to the logic board. Use a cable tie or tape to group the wires together for neatness.

Once again, check to see if you have made the right connections. If you're sure that no mistakes were made, proceed to glue the connector assembly to the console as in Figure 3c. I used a hot glue gun which works very well with plastic. Photo 3 shows the connector glued in place.

While you are waiting for the glue to harden, you can prepare the connecting cable assembly. This cable connects the external keyboard to the connector that was just mounted on the back of the 99/4 console (see Photo 4). A color-coded multi-conductor cable (or ribbon cable) and a male DB/25 connector are needed for this assembly. The cable must have a minimum of fifteen conductors. The pin-out must be identical to the pin-out of the chassis mounted connector. Write down on paper the color of wire that was soldered to each pin (as you did with the ribbon cable earlier). This will make it very easy to identify the coded wires at the opposite end of the cable—the end that goes to the external keyboard (as in Photo 1).

By the time you finish wiring up the cable, the glue should be quite hard. Make sure there is no excessive glue around the edges of the plastic. It's a good idea to lift the console and shake loose any piece of solder or wire that might have fallen inside the console. The bottom shell can now be put back on the console. Insert the power switch lever into place, and fasten the bottom with the screws.

Figure 2



Re-connect the power and video, and test the operation of the computer. All the keys should work as before. If by chance, some keys do not function properly, power down immediately; re-open the console, and check your solder connections. Also, look for solder bridges near the points where you soldered. Otherwise, if everything works fine, you have completed a major step and deserve a cup of coffee.

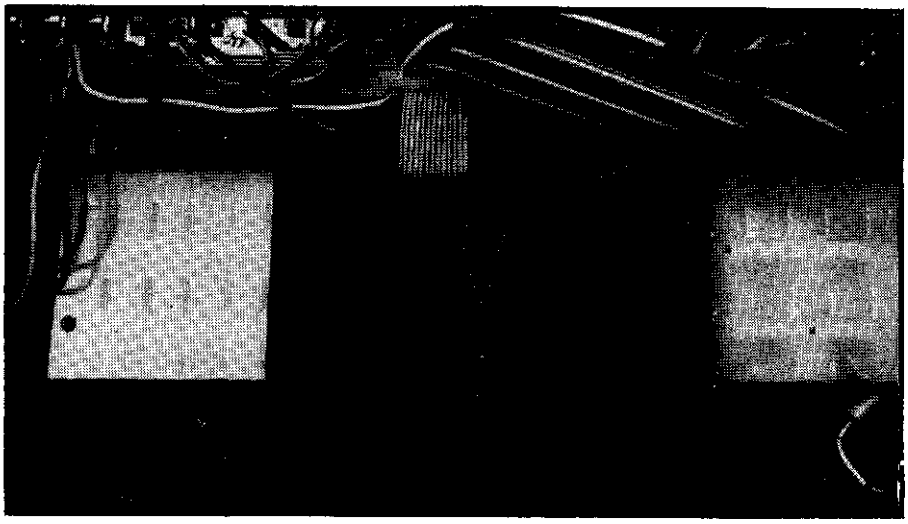
Before you plug in the connecting cable with the male DB/25 at one end, make sure all the power supply lines (at the open end of the cable) are isolated from the keyboard lines. To test the cable assembly and your console modification, refer to Figure 1 and the sheet of paper you used to identify the color-coded wire in the cable assembly. Plug in the male DB/25 connector into the console and turn on the computer's power. Take keyboard scan line 1 and interrupt line 8, and short them together. The character "1" should appear on the display. If you short scan line 3 and interrupt line 8, the character "2" will be displayed. If more than one character appears on the display, it's because you are not making a solid contact. But if no characters appear, then you might have identified the wires incorrectly. Trace back until you find your error. There is a complete list of characters with their coordinates in Table 2. You will have to match these coordinates with whatever keyboard you wish to use.

Choose a keyboard that has easy access to all the traces on both sides of the printed circuit board, since you will probably have to remove the keyswitches and cut all the interfering traces. Once all the trace cuts have been made, replace the keyswitches and wire-up the keyboard matrix the same as the console keyboard in Figure 1.

Keyboard Lines		DB/25 Pin-out
Scan	Interrupt	
1	-----	1
-----	2	2
3	-----	3
-----	4	4
5	-----	5
-----	6	6
7	-----	7
-----	8	8
9	-----	9
10	-----	-----
-----	11	10
12	-----	-----
-----	13	11
14	-----	-----
-----	15	12
16	-----	-----
-----	17	13
18	-----	-----
Power Supplies		
-5V		22
+12V		23
+5V		24
GND		25

Table 1. Keyboard Connector Pin-out

Photo 2



If you're working with a standard keyboard, the SHIFT keys can be connected in parallel. The RETURN key can be wired as the ENTER key. The set of keytops you have, will probably not match the console keyboard(especially the top row). There will also be quite a few unused keys such as the one containing the question mark and slash. My purpose in this first article is to show you how to interface a *minimum-configuration* external keyboard. You'll want to give considerable thought to your choice of what particular keyboard to use, since the characteristics of the keyboard will ultimately affect how much you actually gain from the enhancement.

You can also add remote controls for your game programs. For example, the *Football* Command Module uses four keys per player. Once the time limit and team names have been entered, player A uses keys "1" and "2" to select and enter his plays; player B uses keys "8" and "9". Both players use the "G" key to start their plays, and the "T" key for time-outs. Figure 4a shows one possible configuration of the controls. Seven wires would be required for each control (see Figure 4b), and would be plugged directly into the external keyboard connector. Instead of both players awkwardly trying to access the console keyboard at the same time (and getting sore eyes from being too close to the monitor or TV), they can sit comfortably in their own chairs and get more enjoyment out of the game.

It's important to note that both the external keyboard and the remote controls do *not* interfere at all with the normal operation of the console keyboard. And although the external keyboard presented in this article leaves a lot to be desired, the connector we mounted on the back of the TI-99/4 console will, in fact, allow more sophisticated keyboard designs to be easily implemented.

Key-Scan-Int.	Key-Scan-Int.
1 – 1 – 8	SP – 1 – 4
2 – 3 – 8	A – 3 – 4
3 – 5 – 8	S – 5 – 4
4 – 7 – 8	D – 7 – 4
5 – 9 – 8	F – 9 – 4
6 – 1 – 17	G – 1 – 13
7 – 3 – 17	H – 3 – 13
8 – 5 – 17	J – 5 – 13
9 – 7 – 17	K – 7 – 13
0 – 9 – 17	L – 9 – 13
Q – 1 – 6	SH – 1 – 2
W – 3 – 6	Z – 3 – 2
E – 5 – 6	X – 5 – 2
R – 7 – 6	C – 7 – 2
T – 9 – 6	V – 9 – 2
Y – 1 – 15	B – 1 – 11
U – 3 – 15	N – 3 – 11
I – 5 – 15	M – 5 – 11
O – 7 – 15	– 7 – 11
P – 9 – 15	ENT – 9 – 11

Table 2. Table of Key Co-ordinates

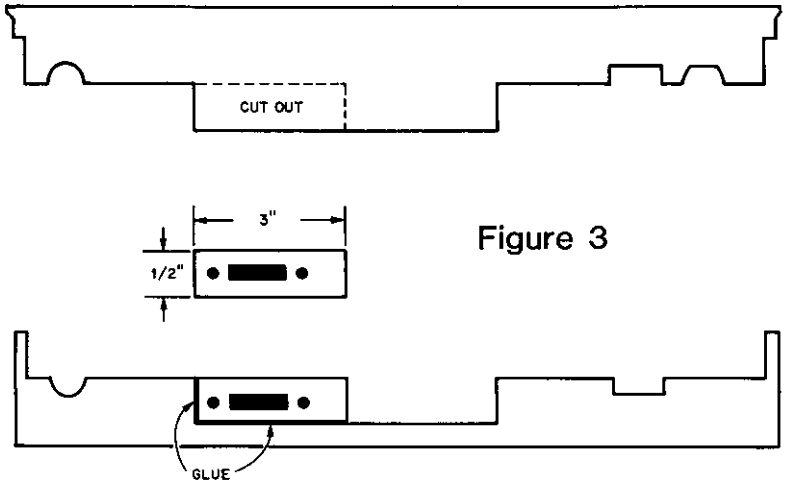
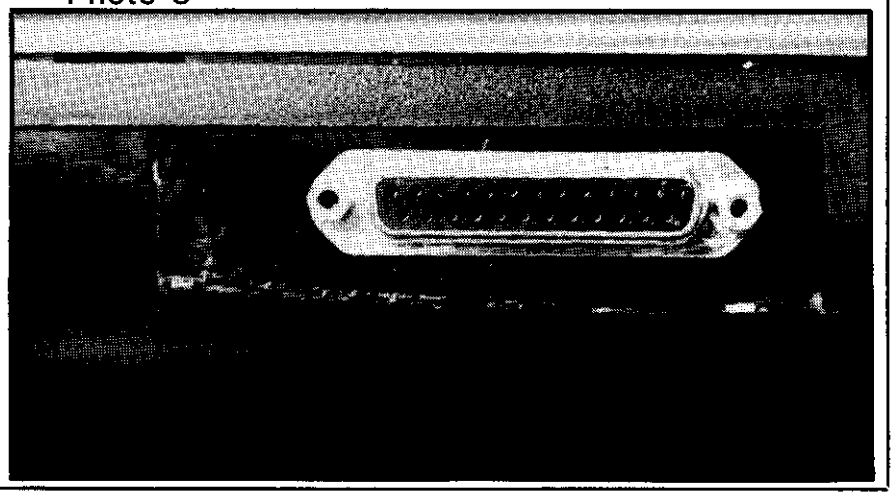


Figure 3

Photo 3



In the next issue, I will show you how to add single-key functions to your external keyboard. You will be able to move the cursor left or right, delete or insert characters, erase lines, or use the edit functions all by individual keystroke. I will also discuss the use of a microprocessor-controlled external keyboard which I am currently designing. By using a microprocessor, the standard keyboard we connect to our console can have extra enhancements such as auto-repeat and pre-programmed string keys. Watch for all this in future issues.

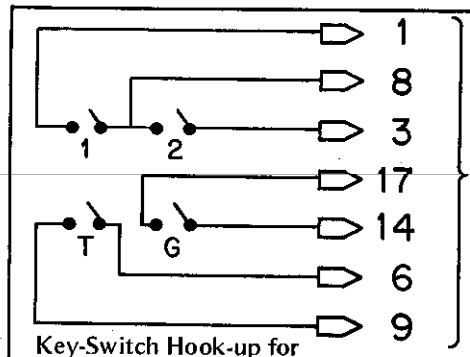
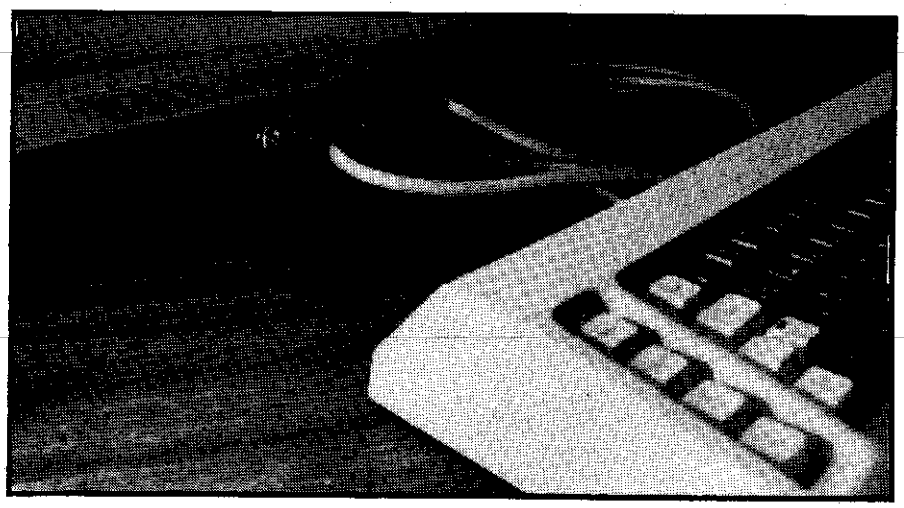
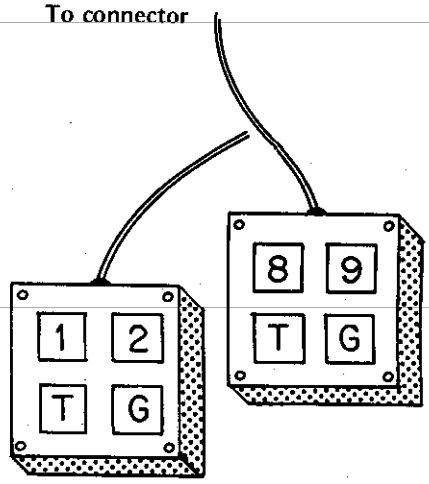


Figure 4

Key-Switch Hook-up for one of the Controls

Photo 4





Remote Controls for Football

MORE GOOD NEWS

FROM MARINCHIP SYSTEMS

OFF-THE-SHELF 16-BIT HARDWARE AND SOFTWARE

16-BIT HARDWARE

Why wait when you want a state of the art 16-bit microcomputer? At Marinchip Systems, we believe in being responsive to your needs. So when you want a system that delivers, call Marinchip Systems, because we have it in stock.

* S-100 CPU board
* TMS9900 microprocessor from Texas Instruments
* Support for 8-bit and 16-bit memories and peripherals
* Software included with CPU
* CPU price - $700.
* 64K byte memory with 16-bit access, bank-select for multi-user systems - $1050.
* Quad SIO board with four programmable asynchronous or sychronous ports - $350.
* Complete systems (except CRT) with 1 Megabyte floppy disc storage - $5500.
* Hard disc available.

CBASIC is a trademark of Compiler Systems

16-BIT SOFTWARE

Because we're a total computer company, we also carry our series of the most complete and flexible software systems available anywhere. So when you want the software that delivers, call Marinchip Systems, because we have it in stock.

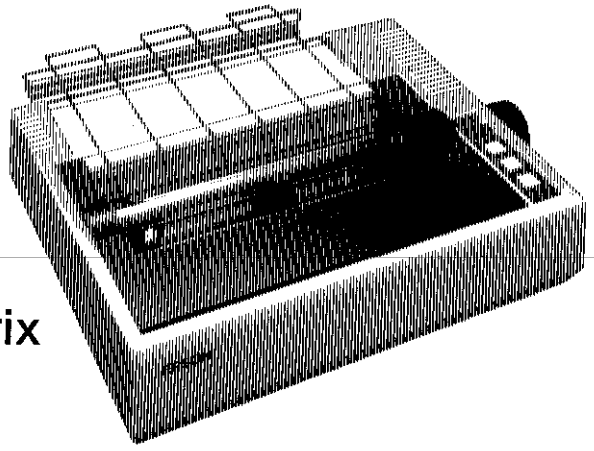
* Single-user Disc Executive - free with CPU
* BASIC interpreter, relocatable assembler and linker, debug monitor, utilities - free with CPU
* EDIT: Context editor with the same features found on UNIVAC, CDC, DEC systems - free with CPU
* WORD: Document formatter with justification, page numbers, user-specified headings and footings, macro expansion, copy from disc, and more - free with CPU
* WINDOW: Simplest, most powerful screen editor you can buy - $250.
* Extended Commercial BASIC: interpreter with 16-digit precision, Print Using, random access disc files - $120.
* PASCAL: Brinch Hansen's Sequential Pascal - $150.
* Applications: The Osborne General Ledger, Accounts Payable/Receivable, Payroll, full source in QBASIC - $150. each
* NOS: Multi-user operating system with byte-addressable device-independent files, hierarchical file system, read/write/execute protection, print spooling, background batch, upward compatible from Disc Executive - $250.
* QBASIC: Extension of CBASIC 2 TM that generates fast machine code for the 9900. New and unique options include fast binary I/O, separate compilation of functions, assembly-language functions - $220.
* Documentation: CPU, free software package, PASCAL, NOS - $40 - applicable to purchase; QBASIC - $20; Applications - $25 each

Marinchip Systems

Computer Power and Human Reason
16 St. Jude Road
Mill Valley, CA 94941
415/383-1545

Advertising Design: Asterisk Associates

EPSON MX-80



Dot Matrix Printer

By G. R. Michaels

Is it possible to have a "love affair" with a printer? The way many Epson MX-80 owners have talked with affection about their "magnificent machines," made me wonder . . . Then someone told me about an MX-80 user's group. "A user's group for a *printer*? Who are you trying to kid, fella . . . "

As it turned out, the "Epsoniac" was speaking the truth; A user's group did indeed exist! My curiosity was piqued. I just had to try an MX-80 for myself.

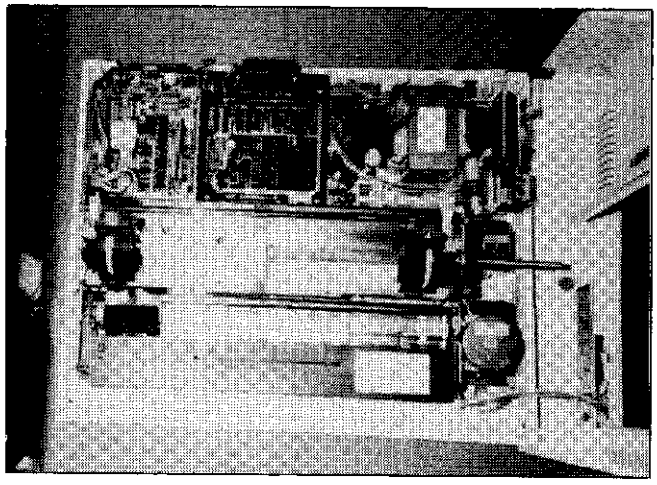
Using any printer, other than the TI Thermal Printer, with the TI-99/4 requires that it be compatible with the RS-232C industry standard. To connect the MX-80, I therefore needed the separate Epson RS-232C Serial Interface card. (Without this circuit board, the MX-80 can connect directly only to computers with a Centronics-style 8-bit parallel port). The RS-232C serial interface is the most commonly used interface in computer systems; all data bits are lined up sequentially and sent along the bus communication lines as a long string of information.

Installing the interface is easy: Remove four screws on the bottom of the case, and pull off the roller knob from its shaft; this allows you to lift off and partially separate the upper case from the printer mechanism (with the cables still connected). The serial interface board simply plugs into the connector on top of the control circuit board, and is held down with four screws in the corner mounting poles. Incidentally, the bottom circuit board has two dip switches that allow you to choose desired control modes. About the only two factory-set options that I might someday change are the ones that allow special symbols for use in France, Germany, and England, and the change-over from the built-in block graphics character set to the Japanese Katakana alphabet. (You just never know when you'll have to dash off a letter in Katakana . . .)

The RS-232C serial interface board that is mounted on top of this control board (the large, dark square area near the top of the photograph) also has a dip switch that allows users to customize the printer to their particular specifications. By throwing a combination of very tiny switches you can choose the transmission bit rate (from 75 to 9600 Baud), the word length (7-bit or 8-bit), and whether a parity check is to be enabled (and if so, a choice of odd or even). Although this dip switch does, in fact, give a user great flexibility in configuring the printer to changing needs, reaching it does necessitate removing four screws, the roller knob, and the top cover; so I'd recommend planning your particular applications ahead of time, and setting the switches accordingly.

Judging from the amount of mail this magazine has received concerning both "getting the graphics to work on the MX-80" and "getting the necessary ASCII codes greater than 159 out of the TI-99/4," I should point out that this particular dip switch is the key. If having the graphics character-set immediately available (without having to open the case) is more important than typing in a few extra characters (**.DA=8**) in *each* **OPEN** and **LIST** command, then set SW1-2 to the OFF position. This sets the printer for recognizing an 8-bit word length, as opposed to the standard (default parameter on the TI RS-232 Interface) 7-bit length.

With the top case removed, you also have access to the printing head (see dark rectangle in lower left section of photograph). Why would anyone other than a service technician want access to this? Because when needed (after 50-100 million characters), the head is designed to be removed and replaced by *any* user—without tools, and in less than a minute! And the best part is that a new MX-80



print head sells for only about $30. Quite obviously, this feature can save a lot of expense and down time.

O.K., so the MX-80 is built with service and reliability in mind. But can this smaller-than-a briefcase, 12-pound package actually live up to its highly touted reputation for outstanding print quality? Although the self-test mode demonstrated an extremely sharp 9x9 dot matrix character set and some block graphics, "word processing quality" it definitely was not. There was only one way to actually find out if the printer was capable of demanding text applications: I had to connect the MX-80 to a port on my RS-232 interface with a male/female DB-25 cable (not included with the printer), and throw some of those "software switches" that printer manufacturers have been recently building into their smart (containing microprocessors) machines.

If you haven't had the opportunity to work with one of these new-generation printers, you're in for a pleasant surprise. By sending some selected **CHR$(...)** codes (with one- to three- digit numbers inside the parenthesis), you can put most of these printers through their paces, and watch them expand and condense character sets, over-strike individual characters, vary their line spacings, etc.

The Epson MX-80 proved to be no exception. It was extremely easy to "run it through its gears" and make it do what I wanted it to do. And what a set of gears! 80-CPS bidirectional printing with logical seeking of shortest lines; full 96-character ASCII with lowercase descenders; forms handling with adjustable tractors; programmable tabs, line, and form feeds; 4 printing modes that control the dot density (through minute paper advances and line repeats, shifting right and double striking, and a combination of the two); plus 4 different pitches (5, 10, 8.25, and 16.5 characters per inch). And although the normal printing pitch of 10 CPI allows only a maximum of 80 characters per line, in compressed mode you can get a full 132 character line on your 8½ inch sheet.

If you have to put your finger on the *one* feature that distinguishes this printer from the rest of the pack, it would have to be its superb "correspondence quality" printing when in the multi-strike and multi-pass mode—truly unique on a dot matrix printer in this price range. The well-formed characters with appropriate lowercase descenders are suitable for all word-processing applications so long as you're not attempting to disguise the printed pages as typewriter-written correspondence or documents. It comes close (and can probably pass quite a few people without notice), but it's definitely not the same as fully-formed characters produced by the much more expensive, direct-impression daisy or thimble printers.

The only features that I found conspicuously absent are such amenities as proportional spacing, underlining, larger buffers (the MX-80 has only one line), friction feed for single sheets or roll paper, and switch-selectable options *outside* the case. But keep in mind that every time an engineering team starts adding options, the price starts to climb. So marketing strategy necessitates that each printer manufacturer package as many desirable features as possible for a target price. Here, the Epson design and marketing team deserves a round of applause: They have created in the MX-80 an affordable printer that is capable of holding its own with many more expensive machines. It rates quite high on my MFPB (More-Features-Per-Buck) scale.

Note: By the time you read this, Epson will have introduced the MX-80FT, with friction feed in addition to tractor feed, as well as a bit-plot graphics option (in ROM) that can be added to existing MX-80s. In the next issue, I will show you a non-Epson adapter for adding single-sheet and roll feed to the MX-80, and explore bit-plot printer graphics.

ASCII, Bits, & Printer Graphics:

A Brief Tutorial

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	NUL		SP	0	@	P	`	p	NUL							
1	0001		DC1	!	1	A	Q	a	q		DC1						
2	0010		DC2	"	2	B	R	b	r		DC2						
3	0011		DC3	#	3	C	S	c	s		DC3						
4	0100		DC4	$	4	D	T	d	t		DC4						
5	0101			%	5	E	U	e	u								
6	0110			&	6	F	V	f	v								
7	0111	BEL		'	7	G	W	g	w	BEL							
8	1000		CAN	(	8	H	X	h	x		CAN						
9	1001	HT		)	9	I	Y	i	y	HT							
A	1010	LF		*	:	J	Z	j	z	LF							
B	1011	VT	ESC	+	;	K	[	k	{	VT	ESC						
C	1100	FF		,	<	L	\	l	|	FF						
D	1101	CR		-	=	M	]	m	}	CR							
E	1110	SO		.	>	N	^	n	~	SO							
F	1111	SI		/	?	O	_	o	DEL	SI							

Take a look at the Coding Table shown here. It happens to be for the Epson MX-80 printer, but it could, however, be for most printers with just a few modifications. In column 0, row 0, you'll find the box containing the NUL character (00000000) which is given the ASCII code 0(zero). Continue down the column until BEL. The bit pattern for that is 00000111, and gets the ASCII code 7. In a similar manner, moving down the columns and across the rows from left to right, all the alphanumeric, control, and graphic characters get number codes. See if you can verify that ASCII 65=A, and produces bit pattern 01000001. Notice that the left-most bit in the first 128 characters (the first 8 columns) is always a zero, and is not needed to distinguish among any of these characters. The ASCII standard is therefore a 7-bit code.

Now, let's look at the Texas Instruments RS-232 Interface that attaches to your TI-99/4. It has a normal default mode of 7 bits. This means that it will only send 7 of the 8 bits to the printer unless you specifically tell it that you want 8 bits by tacking on **.DA=8** to each of your **OPEN** or **LIST** statements. For Example:

```
OPEN #1:"RS232.DA=8"
LIST "RS232/2.BA=9600.DA=8"
```

As long as you stay below ASCII 128(NUL) you're O.K.; but as soon as you go above, the 8th bit is the only thing that differentiates between characters that are exactly 128 codes numbers apart: All characters above 128 have a 1 for the 8th (left-most) bit. (Of course, for this particular coding table, you won't notice the difference until you go above code 159.) For example, find code 161 (in column 10, row 1) that produces a small solid block positioned in the upper left corner of a theoretical 2x3 matrix.

Compare its bit pattern of 10100001 with that for ASCII 33 (the exclamation mark) that uses 00100001. The only difference is in the left-most bit. There is no way for the printer to know that you wanted the graphics character (code 161) without receiving (and being configured to receive by its proper dip switch setting) the 8th bit; in the 7-bit default mode, it will always print out the exclamation mark (!) whenever the BASIC command **PRINT CHR$ (161)** is invoked.

So, if you want to access a printer's resident graphics set (like the MX-80's) keep in mind that the codes for the characters will usually reside above ASCII 127. You'll then need two things: (1) to make sure the 8th bit is sent, and (2) to make sure the printer is configured to recognize it.

99'er MAGAZINE
FOR USERS OF TI-99/4
AND OTHER TMS9900-BASED
PERSONAL COMPUTER SYSTEMS
TM
EMERALD VALLEY PUBLISHING CO.
P.O. BOX 5537
EUGENE. OREGON 97405

Programming Printer Graphics

By W.K. Balthrop

The special graphics character sets that are built into some printers can be extremely useful. In the business world, for example, applications might include the production of charts and graphs, the printing of business forms, and even the design of a letterhead.

The following short program demonstrates how DATA statements are used to format selected graphics characters to produce a letterhead. The DATA statements here are for use with the Epson MX-80 printer, but can be easily modified to accommodate any printer with similar graphics capabilities. Keep in mind that this is a "shell" program; you can plan the DATA statements to direct the printer to produce virtually any design or pattern (within limits of the resident block graphics set). The actual graphic design (the letterhead) in this example is unimportant; but understanding how to plan and implement it is crucial.

DATA statements are read sequentially from left to right, using the READ statement. The Epson MX-80 printer uses numerical codes 160 to 223 (ASCII 32 to 95 with a 1 for the 8th bit), to generate graphics characters already defined within the printer. Each graphics character is made up of one to six squares within a 2x3 matrix as indicated below.

1	2
4	8
16	32

(The numbers within the squares are not important if you have a coding table in front of you. They represent a particular manufacturer's coding of the matrix print head. For example, numerical code 165 would produce the fifth character in the set, and would cause wires number 1 and 4 to fire; if we want the 21st character, wires 1, 4 and 16 would be fired.)

The key part of the program lies in lines 550-590. This controls what will happen when a DATA statement is read. If the first DATA cell is a number greater than 100, that character will be printed. If the first DATA cell is a number greater than 0 but less than 100, the program will read the second DATA cell, and then print it the number of times specified in the first DATA cell. For example, if the first DATA element is 8, and the next is 160 (a blank space), the computer will print a blank space 8 times. This helps lessen the amount of required DATA when it is necessary to repeat the same character several times. If the first DATA cell read is equal to 0, a Carriage Return will be executed.

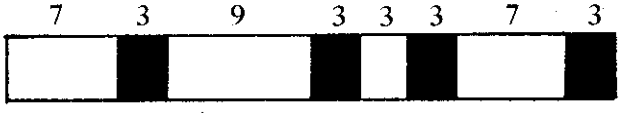
Regular text can be printed on the same line with the graphics. This is done by inserting a negative number in the DATA statement. The value of the negative number designates which message is to be printed out. This can also be used to change the printer's type style, if your printer has that capability. For example, if you want to print a graphics pattern on the left, and a printed message on the right, you would place the negative number just before the zero that causes the Carriage Return. Line 590 controls which message gets printed using the statement:

```
ON ABS(A) GOSUB xxx,xxx,xxx,xxx,xxx,xxx,xxx
```

Every time a character is printed—whether graphics, text, or a control character—it should always be followed by a semicolon. This will insure that the next printed character will be on the same line, until the Carriage Return is executed. The only exception to this is in line 610 where I wanted the Carriage Return to be executed.

The following is an example of DATA and the graphics line it creates. This line can be found in the 8th print line of the letterhead.

```
310 DATA 7,160,3,223,9,160,3,223,3,160,3,223,7,
    160,3,223,-4,0
```



In this graphics line, I first needed to put seven blank spaces between the first character position and the first graphics

characters. This was done with the first two DATA cells. When the program first **READ**s A, its value is 7. Because this value is less than 100 and greater than 0, the program will **READ B**, the next DATA cell. The value of B is 160. ASCII (160) is a blank character on the MX-80, so the program will now **PRINT B** (blank), **A** (7) times. On the next cycle the value stored in **A** will be 3, and the value stored in **B** will be 223. This will cause **B** (ASCII for a solid 2x3 block) to be printed 3 times. This process is continued until a value less than or equal to zero is encountered.

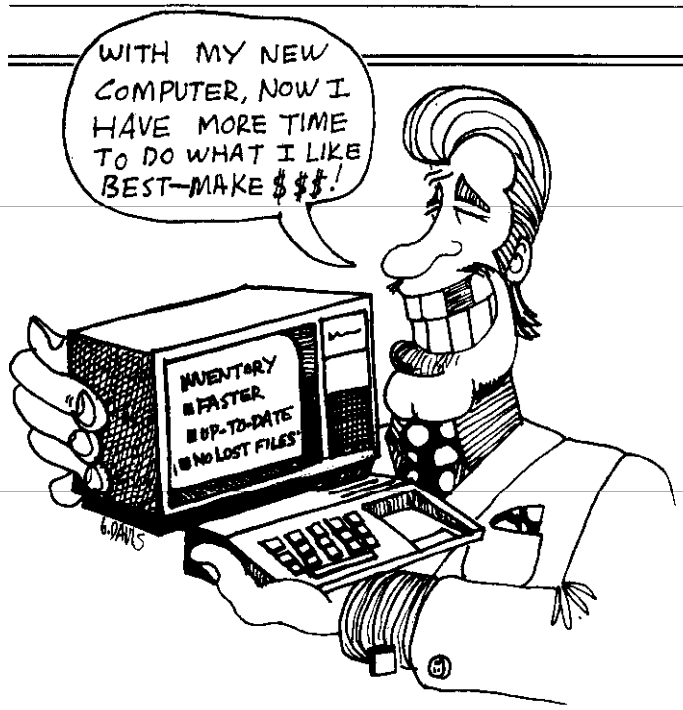
If the value in A is a negative number, the program will branch off to a subroutine which will **PRINT** a text message. In the above example, the value –4 caused the message "FOR USERS OF TI-99/4" to be printed. These subroutines are extremely versatile; you can change the type style, print a message as I have done, continue the program to do calculations, or run program lines.

Note: To conserve space in the magazine listing, I have used many separate DATA statements. All the data necessary for one entire print line could have been contained in a *single* DATA statement. You may, in fact, prefer to do it this way, since it makes the program a little easier to debug.

PROGRAM STRUCTURE FOR LETTERHEAD DESIGN

Line Nos.	
190-510	Contains DATA formatted to print the 99'er Magazine letterhead.
520	OPENs a line to the RS-232 interface for output to the printer.
530	Sets the printer for "Double Strike" mode.
540	Sets the printer for "Emphasized" mode.
550	READs the DATA statement and stores the result in A.
560	Tests A; if A is greater than 100, then PRINT the character stored in A.
570	Tests A; if A is greater than 0, and less than 100, then READ B; Print B, A times.
580	Tests A; if A is equal to zero, then do a Carriage Return. This marks the end of a line.
590	A equals a negative number at this point; ABS(A) controls the branching of subroutines for special tasks e.g., PRINT text.
610-620	Subroutine to execute the Carriage Return.
630-640	Subroutine to PRINT the value in A.
650-690	Subroutine to READ B, and print B, A times.
700-830	Subroutine to print normal text instead of graphics.
840-860	End-of-print message on the screen; END program.

```
100 REM  ****************
110 REM  **              **
120 REM  **   99'ER      **
130 REM  ** LETTERHEAD   **
140 REM  **              **
150 REM  ****************
160 REM
170 REM  BY W.K. BALTHROP
180 REM  99'ER VERSION 5.81.1
190 DATA 10,223,2,160,10,223
200 DATA 160,223,221,0,2,223,6,160,2
210 DATA 223,2,160,2,223,6,160,2,223
220 DATA 160,162,223,0
230 DATA 2,223,6,160,2,223,2,160
240 DATA 2,223,6,160,2,223,0,2,223
250 DATA 6,160,2,223,2,160,2,223,6,160,2,223
260 DATA 3,160,8,223,2,160,2,223
270 DATA 2,160,216,2,223,0,10,223,2,160
280 DATA 10,223,3,160,2,223,4,160
290 DATA 2,223,2,160
300 DATA 2,223,192,222,167,2,163,0
310 DATA 7,160,3,223,9,160,3,223,3,160
320 DATA 2,223,4,160,2,223
330 DATA 2,160,3,223,161
340 DATA 0,7,160,3,223,9,160,3,223
350 DATA 3,160,8,223,2,160,3,223,0
360 DATA 7,160,3,223,9,160,3,223,3,160
370 DATA 3,223,7,160,3,223,-4,0,7,160,3,223
380 DATA 9,160,3,223,3,160,3,223,7,160,3,223
390 DATA -5,0,7,160,3,223,9,160,3,223,3,160
400 DATA 8,223,2,160,3,223,-6,0,-7,0
410 DATA 25,160,183,180,182,181,160,192
420 DATA 166,196,2,160,183,2,163,165,160
430 DATA 192,166,196,2,160,2,163,211
440 DATA 165,160,162,203,163,160,183,196
450 DATA 160,181,160,183,2,163,165,0,25,160
460 DATA 181,162,160,181,160,189,172,172,181
470 DATA 160,213,208,210,181,160,189,2,172
480 DATA 181,160,220,211,208,176,160,192,218
490 DATA 208,160,181,162,196
500 DATA 181,160,215,211,209,180
510 DATA 0,0,-1,0,-2,0,-3,0,-8
520 OPEN #1:"RS232/2.BA=9600.DA=8"
530 PRINT #1:CHR$(27);"G"
540 PRINT #1:CHR$(27);"E"
550 READ A
560 IF A>100 THEN 630
570 IF A>0 THEN 650
580 IF A=0 THEN 610
590 ON ABS(A)GOSUB 700,720,740,760,780,800,820,840
600 GOTO 550
610 PRINT #1
620 GOTO 550
630 PRINT #1:CHR$(A);
640 GOTO 550
650 READ B
660 FOR X=1 TO A
670 PRINT #1:CHR$(B);
680 NEXT X
690 GOTO 550
700 PRINT #1:TAB(30);"EMERALD VALLEY PUBLISHING CO.";
710 RETURN
720 PRINT #1:TAB(38);"P.O. BOX 5537";
730 RETURN
740 PRINT #1:TAB(34);"EUGENE, OREGON  97405";
750 RETURN
760 PRINT #1:TAB(43);"FOR USERS OF TI-99/4";
770 RETURN
780 PRINT #1:TAB(43);"AND OTHER TMS9900-BASED";
790 RETURN
800 PRINT #1:TAB(43);"PERSONAL COMPUTER SYSTEMS";
810 RETURN
820 PRINT #1:TAB(63);"TM";
830 RETURN
840 PRINT "ALL DONE WITH LETTERHEAD"
850 CLOSE #1
860 END
```



Getting Down to Business

By George Struble

You do not need to be reminded that microcomputers are having *more* than a micro impact on business. If you are reading this, it is because you would like some of that impact to benefit you. In a series of articles, we will explore some of those benefits and show you how to incorporate them in your business or professional work. Some articles will be at least partly cautionary —written to try to keep you out of trouble. On occasion, I will review a piece of business-oriented software of potential use to you. We'll also examine some of your experiences (if you'll be kind enough to submit them) in developing applications for the TI-99/4. But don't expect only *success* stories. After all, *failures* can be most instructive too . . .

Planning Use vs. Integrated Use

It is important to distinguish between two major and very different categories of business and professional use of the computer. The first I will call **planning**. This category includes a lot of activities that are helpful to businesses and professional people. Applications in this category tend to be analytical or evaluative. They need not be done on a regular basis, but often can be a dramatic help in charting future direction and improving the profitability of a business. Some applications require rather little in the way of input data, and are essentially projections; others analyze whatever body of historical data that might be available. Some common examples are the following :

- Comparisons of ROI (Return On Investment) for the various options.
- Interest calculations (e.g., effective interest rates on installment loans).
- Profitability analyses for comparing charges and costs of providing various services.
- Lease vs. purchase analyses.

The second category of use is what I call **integrated** use. This category includes a lot of functions that support a business on a minute-to-minute or day-to-day basis. These are for example :

- Maintenance of inventory records
- Preparation of invoices, orders, service contracts, bills, etc.
- Accounts payable and accounts receivable
- Maintenance of customer or mailing lists
- Payroll records
- General ledger and other accounting records.

The potential benefits to your business of applications like these are enormous. But then, so are the risks! Before you allow your business to become dependent on a microcomputer (or any other computer) and set of computer programs, there are a number of steps you must take to safeguard it against the small and large catastrophes that could be (at the least) a major setback for you. This is not to *discourage* you from integrated uses, but rather, to *encourage* you to be very careful about implementing them. In our next issue we will recommend some steps that should help you protect yourself against Murphy's Law—"If anything can go wrong, it will!" —as it *may* apply (and indeed, *has* applied—more frequently than most would care to admit) to integrated computer applications.

A good place for you to start using the power of your microcomputer is in **planning** applications. They don't require extensive systems of programs or comprehensive detailed business records. They don't need to be done at any given moment, at peril of disaster to your business. And you don't have to chase down some itinerant programmer or software house to update your program upon change of, say, some federal tax formula —again, at peril of disaster. Furthermore, you can implement some planning applications yourself, without extra software, disk drives, extensive data files, or a lot of time.

George Struble, a professor of computer and information science at the University of Oregon, is author of **Business Information Processing with Basic,** *Addison-Wesley Publishing Co., 1980.*

Projections : A Planning Use

Perhaps you've heard the story of the wealthy Indian maharajah who was challenged to a chess match by a shrewd foreign merchant. The merchant put up one hundred gold coins as his part of the wager, but only asked for rice if the maharajah lost the game—one grain of rice on the first square of the chessboard, two grains on the second square, four on the third, eight on the fourth, and so on. The maharajah was amused and somewhat skeptical that the merchant would only ask for so little as a few grains of rice, but nevertheless, accepted the challenge. Naturally—or there would be no point to the story—the maharajah lost. And as the prize was being paid, the full impact became shockingly clear: So much rice did not exist in the world! And even if it did, the immense wealth of the maharajah could have only paid for a tiny fraction of it . . .

You are not often confronted with this type of wager. But you do have opportunities to evaluate. A computer can help you to project events into the future, vary the assumptions, and tabulate the projected results. Using a computer program, you can analyze a much more complex situation than you would be willing to do with just a pencil, calculator, and paper. You can change your assumptions and let the computer recalculate and reprint the projections, and thus gain much more understanding of the consequences of various contingencies as you play what is essentially a game of "What if . . ." As an extra benefit, consider the effect the necessity of making clear and explicit assumptions usable by the computer may force you to think more clearly and objectively than you might have done otherwise. (I wonder whether baseball clubs would pay as much for some of their benchwarmers and stars if they evaluated the consequences and contingencies objectively.)

A Program Outline

Perhaps the best thing about a projection program is that it is not hard to write yourself in BASIC. The fundamental tool is a two-dimensional array. If you thought anything connected with arrays was necessarily complex and difficult, please read on. You'll soon discover that an array application can be a lot easier than you imagined.

An array is nothing more than a table in computer storage; a two-dimensional array has rows and columns. We must assign meaning to each, and write our program to honor those meanings. In a projection program, I let each column represent a year (or month?). If the problem requires, I let the numbers in the first column represent initial values, investments, or costs; the numbers in the last column I use to represent residual values, or perhaps totals over all years in the projections. Each row represents a significant quantity that we want to project over the time span.

		1 1981	2 1982	3 1983	4 1984
1	Rents				
2	Vacancy loss				
3	Gross revenue				
4	Property tax				
5	Insurance				
6	Interest				
7	Maintenance				
8	Management				
9	Depreciation				
10	Gross expenses				
11	Net income				

Figure 1. Use of a Table for a Rental Projection

Figure 1 shows a simple projection of a rental operation. There are four columns in the array; they represent 1981, 1982, 1983, and 1984. Each row represents a quantity necessary to the projection of rental results. The array can be declared in BASIC by

```
60      DIM T(11,4)
```

A Basic program can refer to any number in the array: for example, to refer to the maintenance expense in 1982 we refer to T(7,2).

The best thing about a projection program is that it is not hard to write yourself in BASIC . . . you can implement some planning . . . without extra software, disk drives, extensive data files, or a lot of time.

The outline of a program is shown in Fig. 2. Let us examine each of the steps of the outline and show how it would be accomplished in a BASIC program. The first step is accomplished simply by a bunch of LET statements. If rental income for 1981 is projected to be $20,000, with a vacancy rate of 5%, property tax of $3200, insurance of $700, etc., the first several BASIC statements would be

```
1010    LET T(1,1)=20000
1020    LET T(2,1)=.05*T(1,1)
1030    LET T(3,1)=T(1,1)-T(2,1)
1040    LET T(4,1)=3200
1050    LET T(5,1)=700
```

These illustrate several ways of assigning values :

- directly as a given number (as in statements 1010, 1040, 1050)
- as a multiple of another number (as in statement 1020)
- as sum or difference of other numbers (as in statement 1030)

Your application will make it clear how to assign each of your values.

1. Set initial (1981) values
2. FOR each additional year
   compute the projected values
3. FOR each row of the table
   PRINT a row of the table

Figure 2. Outline of a Projection Program

The second step of the program is probably the most complex. The idea is to march across the table, usually deriving each number from the one to its left—that is, the corresponding entry for the previous year. However, some of these entries, too, will be multiples, sums, or differences of other numbers in the same column. We can use the BASIC statement FOR to good advantage here; it easily specifies a repetition for each year. In our rental example, these statements could be :

```
2000    FOR J=2 TO 4
2010    LET T(1,J)=T(1,J-1)*1.08
2020    LET T(2,J)=.05*T(1,J)
2030    LET T(3,J)=T(1,J)-T(2,J)
2040    LET T(4,J)=T(4,J-1)*1.06
2050    LET T(5,J)=T(5,J-1)*1.10
  . . .
2200    NEXT J
```

These statements reflect assumptions :

- Rental income increases at an 8% inflation rate.
- Vacancy continues at 5%.
- Property taxes increase at only (!) a 6% inflation rate.
- Insurance costs increase at a 10% inflation rate.

If you are not sure how all this works, take out your pencil and for J with a value of 2, play computer and fill in numbers in the table yourself as the computer would.

Note how all the assumptions are built into the program; each one can be changed at will. You should, in fact, change several, and re-RUN the program several times in order to see the effect of each of your assumptions. This is sometimes called **sensitivity analysis**, but don't let big words scare you.

Also, you may use the full capabilities of BASIC for any special situations. For example, we might project that in the third year the property will be annexed to the city and taxes will go up 30% instead of 6%. We could replace statement 2040 by

```
2040    IF J=3 THEN 2047
2043    LET T(4,J)=T(4,J-1)*1.06
2044    GOTO 2050
2047    LET T(4,J)=T(4,J-1)*1.30
```

Also, suppose that in the same year we expect to have to put on a new roof for $8000; this is a maintenance expense, but one in addition to the regular budgeted maintenance. And unlike the taxes, the extra maintenance does not continue into 1984. We may use another form of the multiplication here :

```
2070    IF J=3 THEN 2077
2073    LET T(7,J)=T(7,1)*
        1.08 ^ (J-1)
2074    GOTO 2080
2077    LET T(7,J)=T(7,1)*
        1.08 ^ 2+8000
```

In the last step we display the table. The print-out can be prettied up with column headings, a description of each row, and other such features. A bare-bones approach is sufficient, really, and could look like this :

```
3000    FOR K=1 TO 11
3010    PRINT K,T(K,1),T(K,2),
        T(K,3),T(K,4)
3020    NEXT K
```

This segment prints the four numbers of each row of the table on one line, so the table appears on paper just the way we have been thinking about it; each row of numbers is preceded by the row number (K), which at least helps you to identify and keep track of your output.

Figure 3 gathers these program segments into one skeleton. With this as a guideline, you should now be able to sit down and develop your own useful projection programs—applied to sales, production, commissions, or whatever else you need.

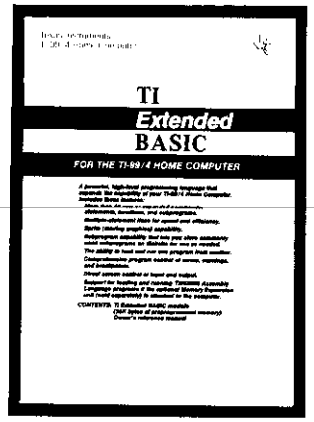
Contributions or Ideas?

Please send in your ideas for what you like to see in this series. If you have a project you are pround of, tell us about it; we always like to share good news. If you have a project that has run into a lot of trouble, write about that too. I may have some suggestions for you, or at least be able to use your experience to help others avoid similar problems.

If you have developed a piece of business applications software that you would like to share with the world (for a fee?), please tell me about it; I might like to review it if the software seems to be of general usefulness. And above all, write if I have made errors . . . I try hard for accuracy, but Murphy's Law hits writers too!

```
10 REM  SKELETON OF A PROJECTION PROGRAM
20 REM  ROWS OF T REPRESENT INCOME OR EXPENSE ITEMS
30 REM  COLUMNS OF T REPRESENT YEARS
60 DIM T(11,4)
1000 REM  STEP 1. INITIAL VALUES
1010 LET T(1,1)=20000
1020 LET T(2,1)=.05*T(1,1)
1030 LET T(3,1)=T(1,1)-T(2,1)
1040 LET T(4,1)=3200
1050 LET T(5,1)=700
1990 REM  STEP 2. PROJECT TO FUTURE YEARS
2000 FOR J=2 TO 4
2010 LET T(1,J)=T(1,J-1)*1.08
2020 LET T(2,J)=.05*T(1,J)
2030 LET T(3,J)=T(1,J)-T(2,J)
2040 LET T(4,J)=T(4,J-1)*1.06
2050 LET T(5,J)=T(5,J-1)*1.10
2200 NEXT J
2990 REM   STEP 3. PRINT THE RESULTS
3000 FOR K=1 TO 11
3010 PRINT K,T(K,1),T(K,2),T(K,3),T(K,4)
3020 NEXT K
9990 END
```

Figure 3



HOW E-X-T-E-N-D-E-D IS IS EXTENDED BASIC?

By Gary M. Kaplan

Nothing has caused as much excitement and anticipation in the TI-99/4 community as the announcement (what now seems like an eternity ago) that an Extended BASIC would be forthcoming. Well, now that the new programing language is about to be gobbled up by hungry Home Computer users, the question on everyone's mind is, naturally enough, "Was it worth waiting for?"

For the answer to this, and to help put the new software in proper perspective, we first should examine TI's claim (in the introduction to the reference manual): "Texas Instruments Extended BASIC . . . has the features expected from a high level language plus additional features not available in many other languages, including those designed for use with large, expensive computers." The key words here are "expected" and "not available." Features such as DISPLAY AT, ACCEPT AT, PRINT . . . USING, IMAGE, ON ERROR, multiple statement lines, expanded IF-THEN-ELSE statements, PEEK, Boolean operators, and assembly language subroutine calls are indeed "expected." Unfortunately, they were expected in the *ordinary* TI BASIC, since they're *standard* features of various Microsoft BASICs found in other machines. But just as plain, old, ordinary TI BASIC has its share of surprises that *aren't* commonly found in other BASICs (e.g., CALL SAY, RESEQUENCE, *complete* EDIT, TRACE, and BREAK utilities, plus its marvelously simple character definition and color assignment facilities), TI Extended BASIC too has its own unique bag of tricks *not found* on other machines. And this bag of tricks includes some mighty impressive feats of computing magic.

But before we get into these *extended* features, let's examine some of the obvious changes from TI-BASIC. First, there's the matter of a slight reduction in usable RAM. The maximum program size of Extended BASIC is 864 bytes smaller than TI BASIC. Although this represents only about a 6% reduction, *any* reduction in user memory is significant if it prevents certain applications from being RUN. And, in fact, as little as 500 bytes is frequently the critical amount of extra memory needed. (Witness the several programs in this issue that cannot be loaded or RUN with the disk controller's power turned on—even with the CALL FILES(1) command that frees all but about 500 bytes for the disk system.) So programmers without the 32K RAM expansion, should try wherever possible to make up the loss with Extended BASIC's built-in memory saving features: multiple statement lines (with more allowable characters per line), expanded IF-THEN-ELSE statements, multiple variable assignments, trailer comments that immediately follow statements (instead of separate REMs), repetition of strings with the RPT$ function, and the use of MIN and MAX functions.

The loss of user-definable characters in the character sets 15 and 16 is another departure from the TI BASIC standard. These custom characters are no longer available to programmers since the memory area is needed to keep track of sprites. Therefore, a TI BASIC program that doesn't use these character sets is supposed to RUN in Extended BASIC [Watch the next issue for an analysis of what *is* and *isn't* interchangeable.] in most circumstances—unless, of course, you've done something that will obviously cause trouble such as accidentally using a TI Extended BASIC keyword as a variable in your TI BASIC program (e.g., DIGIT, ERASE, ERROR, IMAGE, MERGE, MAX, MIN, SIZE, WARNING, etc.)

Now, let's take a peek (no pun intended) into the "bag of tricks" I mentioned earlier. We'll cover only the highlights in this article; subsequent articles will take an in-depth look at the different features, and show you powerful and unique ways of using them.

A good place to start is with Extended BASIC's exciting new graphics capabilities. Nine new subprograms (plus 2 redesigned ones) provide the ability to create and thoroughly control the shape, color, and motion of smoothly moving, high-resolution graphics. These are the true *sprites*—graphics that can be displayed and moved at any of 49,152 positions (192 rows x 256 columns) rather than the 768 positions (24 rows x 32 columns) CALLed by the VCHAR and HCHAR statements of TI BASIC. But that's only the beginning. Sprites can be set in motion with simple X and Y velocity components, and will continue their motion without further program control; they can grow and shrink at will, be relocated or "hidden", and even pass over and blot out fixed objects and other sprites to give the illusion of depth and 3-D animation. [This is a function of the three-dozen stacked image planes of the Home Computer's video display processor chip—a unique graphics display that will be explained more fully in the next issue.]

Although games aficionados and educators have every right to be overjoyed with the new sprites capability, TI-99/4 users who are more interested in business, scientific and professional applications should be drawn to other Extended BASIC features. First on the list is the impressive subprogram capability. Several options exist for communicating values (and entire arrays) between main and subprograms. There's also built-in protection to prevent a subprogram's *local* variables from affecting the *main* variables. Additionally, commonly used subprograms may be SAVEd on a sepa-

rate disk, and later MERGEd. This will allow programmers to build up an library of "universal" subprograms that can be called upon to supply the appropriate modules for new programming tasks—without time-consuming re-coding and debugging.

If this new subprogram flexibility is not enough for your most demanding tasks, how about "program chaining" —where one program can load and RUN another program from a disk. This means that multi-part programs of almost *unlimited* size can now RUN on the TI-99/4 by breaking them into pieces and letting each segment RUN the next. And at any point in this chain, a "menu" may be inserted, allowing the user to choose with a *single keystroke* the particular program to be RUN. Imagine the possibilities!

Those of you with a speech synthesizer, or thinking of purchasing one, will be happy to learn that Extended BASIC includes a speech editor. You will no longer need the separate Command Module (with a retail price of about $45). What's more, with the combination of CALL SPGET, the capability of subroutine MERGEs, and the data for the code patterns (that TI supplies in the appendix of the reference manual), you can now easily add the suffixes ING, S, and ED to the roots of words in the resident vocabulary. And if TI ever supplies users with their master file of coded speech patterns and rules for combining them, it would be possible to create your own new words. As of now, they only provide the cryptic, "Because making new words is a complex process, it is not discussed in this manual." [See the speech synthesis contest in the next issue, for a glimpse of what these code patterns look like—Ed.]

Incidentally, this cabability of having the computer say what *you* want it to say rather than being limited to a fixed vocabulary will, in fact, be implemented through a related approach. I'm referring to the "text-to-speech" capability of the forthcoming *Terminal Emulator II* Command Module which is programmable in TI BASIC. Since only one Command Module at a time can be attached to the TI-99/4, text-to-speech [see related article in this issue —Ed.] cannot be used with the *Extended BASIC* Command Module. I think it's safe to say, however, that we can expect to see TI solve this problem in the near future.

The final two features I'm going to cover in this overview provide a fair degree of software protection and open the door to additional language capabilities. Consequently, these are the particular features that may have the most profound impact on the entire TI-99/4 community—ultimately determining the quality and quantity of most of the commercial software produced for this machine.

Extended BASIC programs can be SAVEd in a PROTECTed f rm to guard against software piracy. This irreversible feature only allows a program to be RUN or loaded into memory with an OLD command. A program thus PROTECTed cannot be LISTed, EDITed, or SAVEd. If the program was originally SAVEd and PROTECTed on a disk, you still must use the protect feature of the *Disk Manager* Command Module to completely "lock up" the software by preventing it from being copied as well.

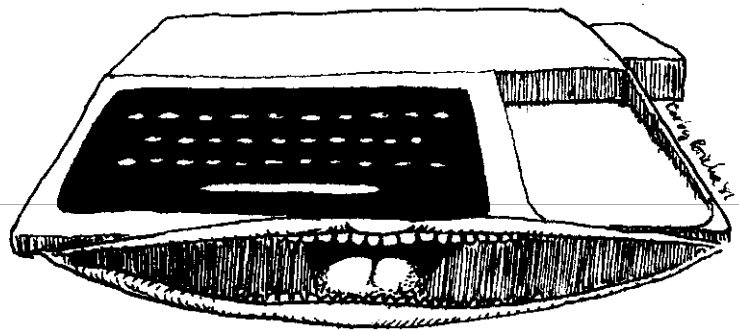
Extended BASIC has the capability to CALL and RUN assembly language programs if the 32K RAM expansion peripheral is attached to the computer. Since assembly language has a much faster execution speed than BASIC, many applications programs that are unfeasible to write in either TI BASIC or TI Extended BASIC (and Extended BASIC is not significantly faster than its predecessor) can now be written in TMS9900 assembly language, LOADed into the expansion memory peripheral and RUN on a TI-99/4. This paves the way for some fairly sophisticated applications programs that now can be targeted for the TI-99/4 users. [See related assembly language article in this issue.]

Even though a TI-99/4 with Extended BASIC and the memory expansion can CALL and RUN assembly language programs and subroutines, it *presently* cannot be used to *write* them. [Except for its "P-coded" version equipped with the *UCSD Pascal Development System* as explained in this issue's related articles on third-party software development, and UCSD Pascal/Version 4.0] Alternately, it would have been helpful if TI had given Extended BASIC users the capability to POKE values into memory locations, but this feature was not implemented. Nevertheless, the door is now opened for a TMS9900 assembler (which TI will soon be marketing) that can be loaded into the expansion 32K RAM peripheral, and called up through Extended BASIC. Besides the *obvious* use of an assembler—being able to write programs or subroutines in assembly language—it does, in fact open up other exciting possibilities: "More exotic" languages can be specially written in TMS9900 assembly for TI-99/4 implementations. (FORTH and LISP hackers take note . . .)

The bottom line is more software tools for developers, and more economic incentive for them to produce valuable programs that can be protected against most piracy. This means that the TI-99/4 user community will be seeing a lot more useful software enter the market. Being able to run this software should more than justify the $100(retail) price for this filled-to-capacity 36Kbyte *TI Extended BASIC* Command Module with accompanying 224-page reference manual. Therefore, the answer to the title's rhetorical question, "How Extended Is Extended BASIC?" is apparently, "Extended enough . . . "

Get Ready Sprites —

Here I Come . . .



TEXT-TO-SPEECH ON THE HOME COMPUTER

By Gary M. Kaplan

Go ahead—shout something at your home computer . . . but don't be surprised if it answers you back! Welcome to the exciting world of talking computers—a world in which synthetic speech will very soon cease being a novelty, and will instead become instrumental in the everyday interactions between humans and machines.

If you have a Texas Instruments Home Computer, you're one jump ahead of everyone else in taking advantage of this revolutionary communications tool. All you need additionally is the Speech Synthesizer peripheral, and the plug-in *Terminal Emulator II* Command Module. Text that you type on the console keyboard will be converted to synthetic speech, and "spoken" through your TV set or monitor. There's no fixed vocabulary to constrain you, and personal phrases can be called up under program control through the TI BASIC computer language.

But this is only the beginning . . . If you connect TI's RS-232 interface and a modem to this configuration, you can have access (through your telephone) to the electronic mail, database, entertainment, and computing facilities of The SOURCE and its soon-to-be-available offspring, TEXNET.* The TEXNET service will allow TI-99/4 users to access all the menu selections from its parent information utility (plus some additional) *with the enhancements of text-to-speech, sound effects, music, and color graphics!* Imagine a weather report with a color graphic representation of a bright sun being blotted out by ominous looking rain clouds, while *"Stormy Weather"* is being played in the background, and the temperature, wind, humidity and other vital statistics are flashed on the screen and recited to you by your speech synthesizer—an exciting prospect at the least.

Linear Predictive Coding (LPC)

When Texas Instruments made the first single-chip speech synthesizer in 1978, its original application was in their Speak & Spell learning aid. The chip, a TMS 5100, is essentially an electronic model of the human vocal tract (a mathematical model implemented as a filter network) that reproduces speech through a technique known as *Linear Predictive Coding (LPC).* There have been other approaches to speech storage —methods employing digitalized speech and pulse-coded modulation codecs—but these result in data rates that are too high (64,000-100,000 bits per second). And the higher the data rate, the fewer words of speech the available memory can hold.

The value of TI's LPC technique is its modest memory requirement; it provides speech quality nearly comparable to these other methods, at a much lower data rate (1,200 bits per second). For example, a speech reproduction of the words "Texas Instruments" requires approximately 90 times as many bits using digitized speech techniques as it requires with LPC.

What is the secret to LPC's economy of storage? The "P" in the middle that stands for "Predictive." Here's how it works: A speech waveform is originally sampled and encoded. This data is used to calculate the coefficients of the linear equations of the digital filter network that will control the "shape" of this synthetic vocal tract. When excitation noise (a chirp function and white noise generator) is applied to this filter network, the circuitry produces a simulation of the resonant effects of the mouth and nasal cavities.

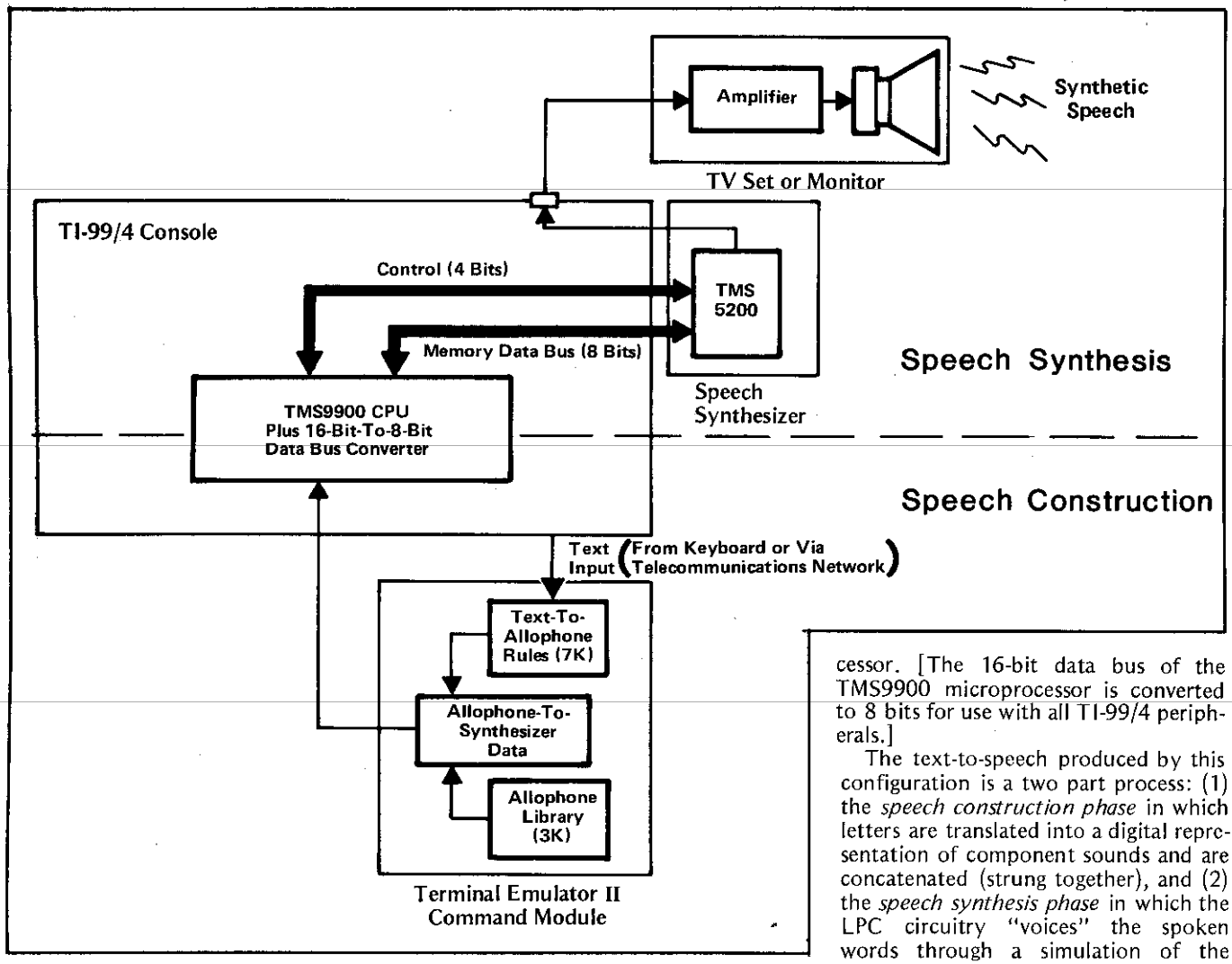
Allophones

This is fine if a user doesn't mind being confined to a pre-stored vocabulary. It is, however, under-utilizing the synthesizer's capability to produce *any* spoken word *on demand* as long as it has the appropriate input data. This is where TI's recently unveiled allophone stringing technique comes into play. TI linguists have chosen 128 separate sounds called "allophones" that can be linked together to sound out any word in the English language. Allophones are variations of a particular "phoneme" (the smallest unit of speech that can distinguish one utterance from another) that are modified by the environment in which they occur. For example, the aspirated (followed by a puff of air) "p" in "pin" and the non-aspirated "p" in "spin" are allophones of the phoneme "p." These allophones represent the sound more accurately than the phoneme.

A total of 128 allophones are grouped in a library occupying only 3 kilobytes of memory storage. Each allophone is identified with a numerical code (indicating the parameters for setting the filter characteristics in the LPC synthesizer). When a word is entered into the computer, its ASCII (American Standard Code for Information Interchange) representation is identified; the computer's CPU then searches through a set of rules (contained in 7 kilobytes of memory storage) to pick out the appropriate allophones and string them together in the proper sequence ("concatenation") to represent the keyed-in words.

The rules (about 650 presently) over-

* Service Mark of Texas Instruments



come most of the many pronunciation exceptions and irregularities in the English language (able to correctly select both phonemes and allophones over 90% of the time). However, speech scientists have found it impossible to achieve 100% accuracy in a text-to-speech system of this type since there are too many silent letters and other incongruities that humans perceive, but that the computer cannot discern. To get around this problem, some words must be typed into the computer phonetically or entered allophone-by-allophone.

If this was all the text-to-speech software did, the quality of the speech would sound monotonous and unnatural. TI therefore provides its software with the ability to produce more lifelike inflection: users can add stresses to certain syllables, and required pitch patterns to particular points in a sentence. Questions then sound like questions, and commands like commands!

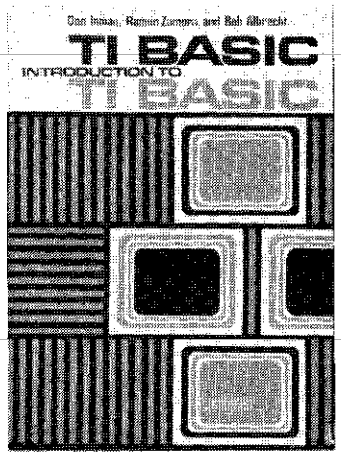
Text-to-Speech with the TI-99/4

The chip used in the Speech Synthesizer peripheral that attaches to the TI-99/4 is a TMS 5200—a second generation of the TMS 5100 (used in the Speak & Spell). It has the following added features : (1) "Speak External" input which allows the chip to accept speech data from a source other than a Speech ROM (read-only memory), (2) an internal buffer to store chunks of data (freeing the CPU for other tasks), and (3) a memory data bus allowing it to work with any standard 8-bit microprocessor. [The 16-bit data bus of the TMS9900 microprocessor is converted to 8 bits for use with all TI-99/4 peripherals.]

The text-to-speech produced by this configuration is a two part process: (1) the *speech construction phase* in which letters are translated into a digital representation of component sounds and are concatenated (strung together), and (2) the *speech synthesis phase* in which the LPC circuitry "voices" the spoken words through a simulation of the mouth and nasal cavities. As seen in the accompanying diagram, speech construction is handed by the software resident in the *Terminal Emulator II* Command Module, and speech synthesis by the TMS 5200 chip within the separate speech peripheral.

Notice the reactions of people hearing text-to-speech for the first time (introduced at the January Consumer Electronics Show).

99'er BOOKSTORE



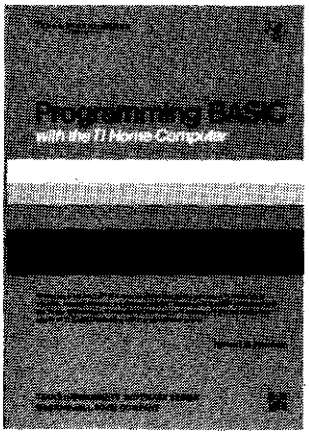
INTRODUCTION TO TI BASIC

By D. Inman, R. Zamora, and R. Albrecht.

This comprehensive work, written by three of the foremost microcomputing programming experts in the country, will teach you all about computers and BASIC for use with the Texas Instruments Home Computer. Even if you've never worked with a computer, you can now teach yourself how to use, program and enjoy the TI Home Computer with this entertaining, and easy-to-read work. The authors have carefully constructed this introduction so that you will soon be writing BASIC programs and exploiting all of the excellent features of the TI machines. Its 14 chapters and Appendices cover all of the essential programming statements and machine features.

CONTENTS: Gateway to Adventure. Do It Now: Sound and Color Graphics. Simple Programming. Looping Sound and Color. More Programming Power. Beginning Simulation. More Program Control Statements. Using Data Files. One Dimensional Arrays. Two Dimensions and Beyond. Color, Graphics, Sound, and Animation. More Strings. Editing. Subroutines and Your Personal Library.

paper, **$10.95**
1980, 320 pages, 7⅛ x 9¼

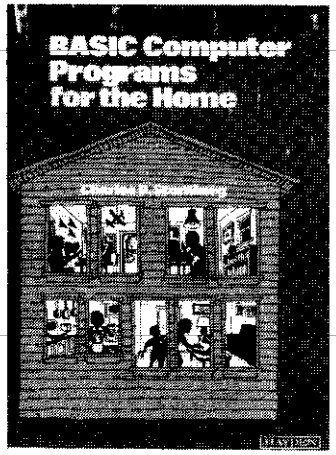


PROGRAMMING BASIC WITH THE TI HOME COMPUTER

By Herbert D. Peckham.

A tutorial guide that helps you learn TI BASIC in a friendly, relaxed manner. It goes beyond *Beginner's BASIC* furnished with the TI-99/4, and introduces the full range of TI BASIC features including color graphics and sound. Its 11 chapters are written in a complete-the-blanks, programmed instruction format.

paper, $10.95
1979, 306 pages, 6 x 9



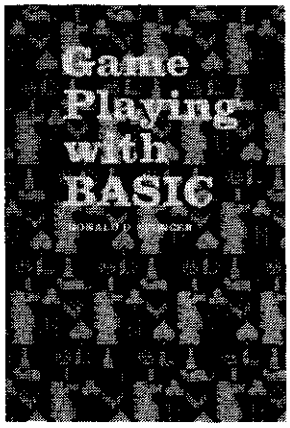
BASIC COMPUTER PROGRAMS FOR THE HOME

By Charles D. Sternberg.

The only book named in an update article in the Personal Business section of *Business Week* (June 23, 1980)!

An invaluable book at a great price, it contains over 75 practical home application programs that will be helpful to the novice or experienced owner in increasing the usefulness of any home computer. Each program is documented with a description of its functions and operation, a listing in BASIC, a symbol table, sample data, and one or more samples. Programs included are: Home Financial Programs; Automobile Related Programs; Kitchen Helpmates; Scheduling Programs for Home Use; List Programs for Every Purpose; Miscellaneous Programs for the Home; Tutorial Programs for Home Use; Conversion Program; and Hobbyist's Diaries.

paper, **$9.95**
1979, 336 pages, 7⅛ x 9¼



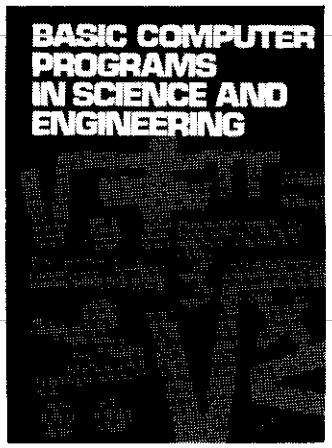
GAME PLAYING WITH BASIC

By Donald D. Spencer, Abacus Computer Corporation.

Enjoy the challenge of competition with your computer. Amuse yourself with such games and puzzles as 3-D Tic-Tac-Toe, Nim, Roulette, Magic Squares, the 15 Puzzle, Baccarat, Knight's Magic Tour, and many others. The writing is nontechnical, allowing almost anyone to understand computerized game playing. The book includes the rules of each game, how each game works, illustrative flowcharts, diagrams, and the output produced by each program. The last chapter contains 26 games for reader solution.

paper, **$9.55**
1977, 176 pages, 6 x 9, illus.

99'er BOOKSTORE



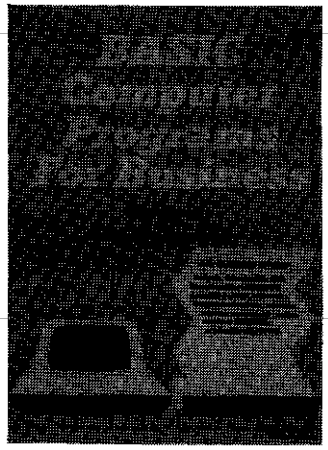
BASIC COMPUTER PROGRAMS IN SCIENCE AND ENGINEERING

By Jules H. Gilder.

Save time and money with this collection of 114 ready-to-run BASIC programs for the hobbyist and engineer. There are programs to do such statistical operations as means, standard deviation averages, curve-fitting, and interpolation. There are programs that design antennas, filters, attenuators, matching networks, plotting, and histogram programs. There is even a justified typing program that can be used in typesetting. All programs in the book have been tested and are fairly universal; so you should have no difficulty running them on your system. You won't find anywhere a more comprehensive collection of usable, ready-to-run BASIC programs!

See also software section.

paper, **$9.95**
1980, 160 pages, 6 x 9, illus.



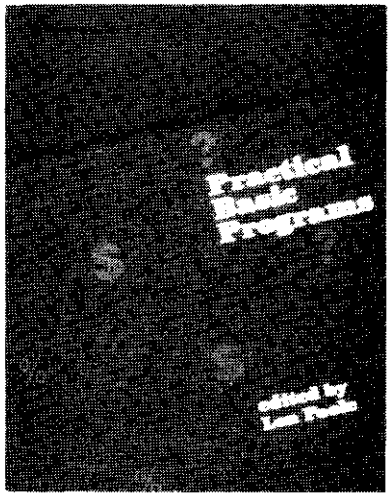
BASIC COMPUTER PROGRAMS FOR BUSINESS: VOL. 1

By Charles D. Sternberg.

A must for small businesses utilizing micros as well as for entrepreneurs, these two volumes provide a wealth of practical business applications. Each program is documented with a description of its functions and operation, a listing in BASIC, a symbol table, sample data, and one or more samples.

Volume 1 contains over 35 programs covering: budgets, depreciation, cash flow, property comparisons, accounts payable, order entry, warehouse locations, inventory turnover analysis, job routing, resource allocation, production scheduling, etc.

volume 1, paper, **$10.95 (t)**
1980, 384 pages, 7 x 9¼



PRACTICAL BASIC PROGRAMS

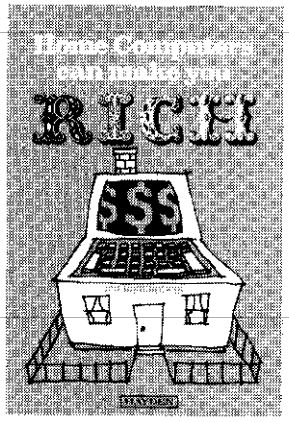
Edited by Lon Poole

Here is a new collection of 40 programs you can easily key in and use on most microcomputers. Each program does something useful. *Practical BASIC Programs* is especially useful in small business applications. It solves problems in finance, management decision, mathematics and statistics. It requires no prior programming knowledge. Each program is thoroughly documented. The book contains sample runs, practical problems, BASIC source listings, and an easy to follow narrative to help you realize the potential uses of each program. This book is a valuable reference for anyone who needs a wide range of useful programs: income averaging, present value of a tax shield, lease/buy decision, financial statement ratio analysis, checkbook reconciliation, home budgeting, nonlinear breakeven analysis, Program Evaluation and Review Technique (PERT), statistics, data forecasting divergence, musical transposition, Bayesian decision analysis, etc.

paper $15.99
1980, 200 pages, 8½ x 11

Use the order card in the back of this magazine, or itemize your order on a separate piece of paper and mail to: 99'er Magazine/Book Dept., 2715 Terrace View Drive, Eugene, OR 97405. Be sure to include check or detailed credit card information. No. C.O.D. orders accepted. Add $1.50 postage & handling for 1 book, $2.00 for 2 books, or $2.50 for 3 or more books. Please allow 4-6 weeks for delivery. If there is a question regarding your order please write to Customer Service at the above address. PRICES SUBJECT TO CHANGE WITHOUT NOTICE.

99'er BOOKSTORE



HOME COMPUTERS CAN MAKE YOU RICH

By Joe Weisbecker.

Here's a valuable text for every home computer owner and nonowner interested in spare-time income opportunities. You'll be introduced to the microcomputer industry, and the types of people involved in it. You'll find out how to learn more about this new industry. Discussed are basic principles of making money, freelance writing, programming, consulting, inventing, computer-made products, investing, and much more. The goal of this book is to stimulate computer designers and microcomputer companies to direct more effort to the home computer market.

CONTENTS: The Microcomputer Industry. What You Need to Know About Making Money. Resources You Can Use. Choosing Your Hardware. Writing for Money. Creating and Selling Programs. Services for Sale. Use Your Imagination. Invent Your Way to Success. Making Your Money Grow. Working at Home.

5177-8, paper, **$6.50**
1980, 128 pages, 6 x 9

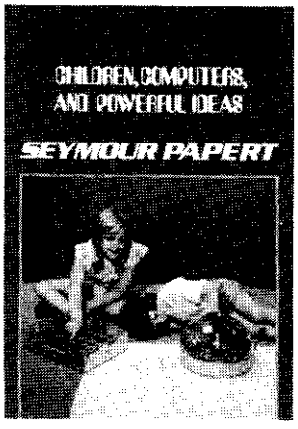
BEAT THE ODDS: MICROCOMPUTER SIMULATIONS OF CASINO GAMES

By Hans Sagan.

Here's an extremely useful programming guide that provides realistic simulations of five popular Casino games: Trente-et-Quarante (Thirty and Forty), Roulette, Chemin-de-Fer, Craps, and Blackjack. Each of the five chapters has the same structure. It begins with a computer run, displaying facets of the programs, followed by an explanation of the objectives and the physical execution of the game. Acceptable bets and how to place them are discussed and systems and/or strategies laid out. Finally, the computer program is developed and various modifications of the program are detailed.

All programs are written in BASIC and heavily REM'd for readability and conversion. A comprehensive bibliography, a glossary of French gambling terms and phrases, and hints on the discrepancies between BASIC dialects are included, as well as a summary of maxims of probability theory.

5181-6, paper, **$7.95(t)**
1980, 128 pages, 6 x 9



MINDSTORMS: CHILDREN, COMPUTERS AND POWERFUL IDEAS

By Seymour Papert

The definitive work on the philosophy behind LOGO. Excerpted in the May/June issue of this magazine.

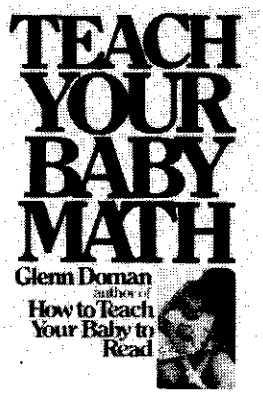
hardcover, $12.95
1980, 230 pages, 6 x 9

TEACH YOUR BABY MATH

By Glenn Doman

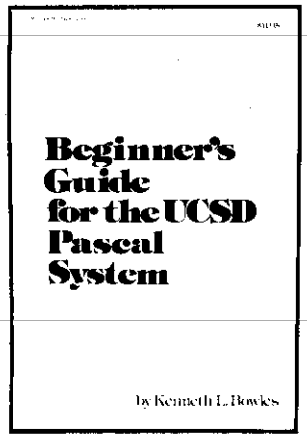
The book upon which the *Tiny Math I* program (in the May/June issue of this magazine) is based.

hardcover, $8.95
1969, 110 pages, 6 x 9



Use the order card in the back of this magazine, or itemize your order on a separate piece of paper and mail to: 99'er Magazine/Book Dept., 2715 Terrace View Drive, Eugene, OR 97405. Be sure to include check or detailed credit card information. No. C.O.D. orders accepted. Add $1.50 postage & handling for 1 book, $2.00 for 2 books, or $2.50 for 3 or more books. Please allow 4-6 weeks for delivery. If there is a question regarding your order please write to Customer Service at the above address. PRICES SUBJECT TO CHANGE WITHOUT NOTICE.

99'er BOOKSTORE

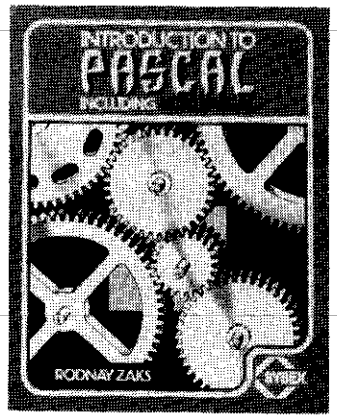


BEGINNER'S GUIDE FOR THE UCSD PASCAL SYSTEM

By Kenneth Bowles

This highly informative book is written by the originator of the UCSD Pascal System. It is designed as an orientation guide for learning to use the UCSD Pascal System, and features tutorial examples of programming tasks in the form of self-study quiz programs. Once familiar with the system you will find the guide an invaluable reference tool for creating advanced applications.

paper $11.95
1980, 204 pages, 6 x 9

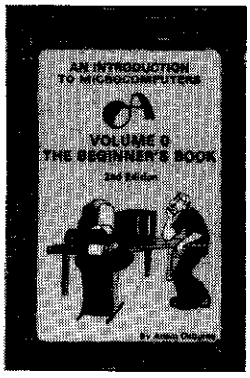


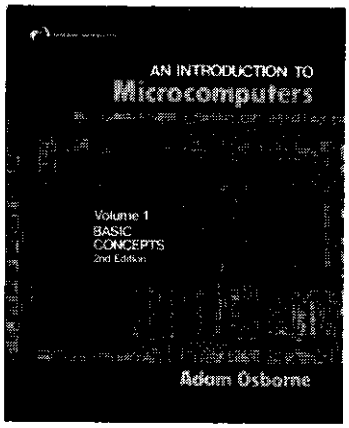
INTRODUCTION TO PASCAL (INCLUDING UCSD PASCAL)

By Rodnay Zaks

This is the first book on Pascal that can be used by persons who have never programmed before, but more generally it is a simple and comprehensive introduction to standard and UCSD Pascal for anyone—beginner to experienced programmer—who wants to learn the language rapidly. The logical progression and graduated exercises—designed to provide practice as well as test skill and comprehension—enable the reader to begin writing simple programs almost immediately. This book presents all concepts and techniques in a clear and simple style, making it accessible to beginners and useful to experienced programmers. All Pascal features are covered in detail, from basic definitions to complex data structures. An extensive appendix section presents a listing of all symbols, keywoods and rules of syntax for programming in Pascal, providing a concise summary and important reference tool.

paper $14.95
1981, 440 pages, 7 x 9





AN INTRODUCTION TO MICROCOMPUTERS — VOLUME O: THE BEGINNER'S BOOK

By Adam Osborne

Here's the book to start with if you know nothing about microcomputers but wish to learn about them. With the help of numerous illustrations and a wonderfully lighthearted text, *Volume O* will help give you a sound understanding of the basics of microcomputing. You'll learn about the microcomputer's construction, terminology, internal logic, and application. If you have plans to program microcomputers, or if you must make decisions related to microcomputer applications, *The Beginner's Book* will provide the terminology and general concepts you'll need. This volume also provides an excellent background for the beginner wanting to go on to *An Introduction to Microcomputers: Volume 1 — Basic Concepts.*

paper, $7.95
1979, 240 pages, 5 x 8

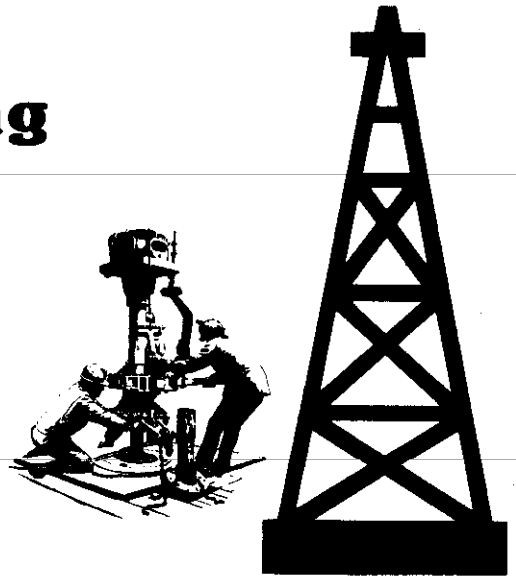
AN INTRODUCTION TO MICROCOMPUTERS — VOLUME 1 — BASIC CONCEPTS

By Adam Osborne

Using concepts that are common to all microprocessor systems, *Volume 1* develops a detailed picture of what a microcomputer can do, how it does what it does, and how the capabilities of microcomputers can best be applied. *Basic Concepts* presents the fundamental logic framework upon which microcomputer systems are built, so that the reader can evaluate the applicability of microcomputers to any practical problem. This new revised edition incorporates all recent microprocessor developments. Concepts are discussed in terms of modern hardware configurations, and examples of common applications are drawn from today's most popular devices. For example, the logic instructions and programming concepts of the new 16-bit microprocessors are discussed in detail, and current logic distribution configurations are used throughout the text with numerous illustrations and examples. Programming mnemonics conform to the newly proposed IEEE standard. This is the first book in print to use them.

paper, $12.99
1980, 320 pages, 7 x 9

Wildcatting



By W. K. Balthrop

Let me tell you the story about a man named Jed,
A poor mountaineer--barely kept his family fed,
And then one day while shootin' at some food,
Up from the ground came a'bubblin' crude . . .
Oil, that is--"Black Gold" . . . "Texas Tea."

The Beverly Hillbillies
(A popular TV show of the late 1950s)

Ever since the debut of *The Beverly Hillbillies* in the late 1950s, one of the American public's favorite "get-rich-quick" fantasies has been to strike oil in one's own backyard . . . and like Jed, become an "instant millionaire." The last decade's oil price hikes, along with the recent heavy promotion of oil and gas lotteries have undoubtedly added to the mystique and lure of "wildcatting" —prospecting for oil in an untapped or questionable area. And Hollywood movies *haven't* exactly portrayed wildcatters as calm, level-headed, conservative businessmen who are out to make a reasonable profit. Rather, we're shown images of a lusty, brawling gambling lot who are time and again willing to risk all they have (and can borrow!) on a crapshoot—little more than a hunch that the next well sunk won't come up dry . . .

What is it *really* that brings success and wealth in this romantic endeavor? Luck? Courage? Good business sense? Or perhaps, a combination of all three? Well, now you can find out for yourself—in the comfort of your own home, with nothing more to risk than the price of a new simulation program for your Home Computer.

Wildcatting, from Image Computer Products, is a game of strategy for up to four players. In no time at all, any of the players may, in fact, "strike it rich," or "come up dry" once too often—joining the ranks of history's defeated gamblers . . .

Operation

To play *Wildcatting*, you'll need a *color* TV or *color* monitor connected to your TI-99/4. Unlike many other computer "business simulations" that display only text or simple graphs, this program includes a simulation of an oil field as a three-dimensional model, with colors representing the height. Each time the program is run, a different map is constructed and stored in the computer's memory. Drilling costs, the predicted probability of striking oil, and the amount of oil a well produces are all calculated for each location.

Each player takes a turn (representing a week) picking a location, taking a geological survey, and deciding whether or not to invest money drilling for oil. The survey reveals the probability of a strike, drilling cost per meter, and weekly taxes. If drilling is attempted, the drilling cost, depth of well, and player's net worth are displayed every 50 meters. A cross-sectional view of the actual drilling progress is also displayed on the screen and controlled by depressing the space bar. A player can stop drilling at any time when he or she feels that the cost is getting too expensive; The well may then be sold for a loss. Any of the player's other wells that start to dry up (and produce less income than the weekly taxes) may also be sold at that time. After each player's turn, a color position marker is left on the oil field. These squares indicate the probability of striking oil at that location. By studying the color patterns that begin to emerge after a few turns, it becomes possible to make predictions on where the exact center of the oil field (with 100% probability of striking oil) will be. Players have ten weeks to either amass their fortune or go broke.

Performance and Engrossment

Although the economics are purely fictional, the costs for drilling and the outputs per week provide a well-balanced and realistic game. Simple, straightforward instructions make the game easy to learn. The controlling keys are well documented with screen instructions: A player makes decisions with a simple "Y" or "N", selects wells to be sold by using the up and down arrow keys, and accomplishes all other functions with the space bar. Despite the game's simple structure and ease of play, it is nevertheless challenging and difficult to win. Even in an area with a high probability of striking oil, the well may, in fact, be dry!

The only minor operational fault I could find is display-dependent. It is, however, not the direct fault of the software developer. At the time the game was originally designed, the TI-99/4 was sold together with the color video

monitor; the RF modulator (for hooking up the computer to a regular TV set) wasn't available. When used as a computer display, some TVs can't fit all 32 columns on the screen, and wind up chopping off one or two columns from each edge. So if a programmer is working only with the monitor and plans the display to utilize the very outer columns for important information, it can be a problem for users with certain models of TVs. Since in *Wildcatting* you use a blinking arrow on the extreme left to indicate the well you decide to sell, it's quite possible that on certain TVs you won't be able to see this symbol, and might accidentally sell the wrong well! Fortunately, the map itself is not affected.

[Since most TI-99/4s are now being sold without monitors, *99'er Magazine* suggests that all software developers take into account the wide range in performance of different TV sets, and design their program screen displays accordingly—Ed.]

If *Wildcatting* merely had the features that I've detailed so far, I would have to classify it as "an interesting game that performs well, and is easy to learn and play." I wouldn't necessarily feel that it was particularly engrossing or absorbing—in the sense of losing sleep over . . . or forgetting to eat. But the people at Image Computer Products did, in fact, put in one additional feature that caused the program to jump up a few notches on my "engrossment scale": Each player gets a chance to actually "handle the drill" (controlled by depressing the space bar), hear the whirring and grinding noise of the bit, and watch in fascination as the shaft plunges deeper and deeper into the earth. When you finally make a strike, the abrupt audio and visual effects that simulate the "gusher" practically jolt you out of your seat (at least it nearly did the first time . . .). It's quite a realistic effect —one that will keep you anxiously awaiting for your turn to "handle the drill" to come around again.

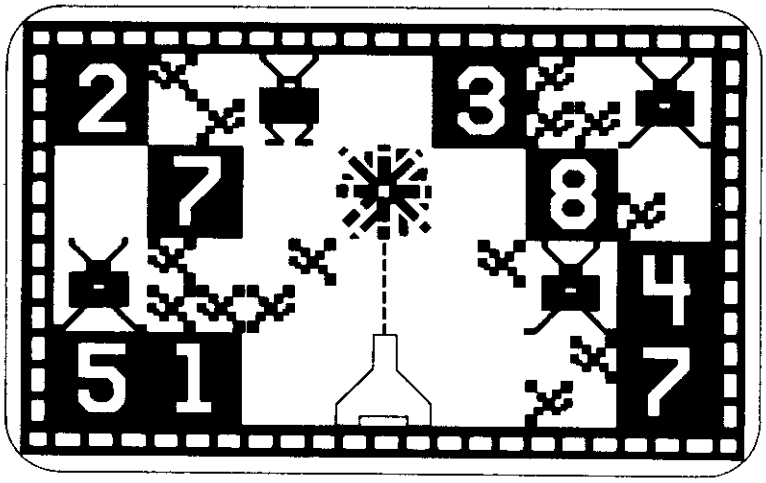
Documentation and Packaging

The people at Image have done well in making sure that the product lives up to their company name. The cassette (within its protective plastic case) is packaged in an attractive, 4-color, book-size box; it can conveniently be stored in a bookshelf next to your TI Command Modules. The 6-page typeset instruction booklet explains every step of play in well defined, easily-referred-to sections—making it quick and easy to look up some forgotten detail. The one minor complaint I have concerns the design of the storage box. I'd like to see a latch on the rear panel of the cardboard box to prevent the cassette and case from accidentally falling out when the box is pulled from the shelf or carried.

All in all, I feel that *Wildcatting* is an effective software product that many TI-99/4 users will enjoy playing repeatedly. Potential third-party software developers would do well to study the way this product has been implemented.

Wildcatting
TI BASIC; 16K
$14.95 (Cassette)
Image Computer Products, Inc.
Northbrook, Illinois

The Attack*



They're coming from another world—from deep space! Suddenly you're surrounded by an infestation of spores. You must destroy the spores before an alien is formed. The aliens are here. Their numbers are growing. And they're coming for you! Your mission . . . Destroy the aliens before they destroy you. The world is counting on you. Good luck, Commander.

By W. K. Balthrop

You'll need more than luck, Commander—nothing less than impeccable will power—if you hope to get any serious work done after you've made the mistake of unboxing this single-player video game to just "check it out and make sure it works . . . " You'll soon discover the *true* significance of the game's name. No, I don't mean an "attack" from some bowlegged TV sets with an affinity for the Charleston who try and pass themselves off as aliens. What I do mean is an "attack" on your time—complete with hypnotic music and engrossing, fast-paced, edge-of-your-seat action that can temporarily erase from your mind a few of life's *minor* details such as family, eating, and sleeping . . . A word of warning: If you use the TI Wired Remote Controllers, don't play the game within one hour of having to do something that involves the critical usage of your hands—e.g., the finger dexterity needed for brain surgery or for a flamenco guitar performance. After an exciting bout of *The Attack* with its inevitable frenzied clutching at the controls, your hands will need a rest, Commander.

Learning to play the game by reading the brief instruction booklet shouldn't take more than a few minutes. But learning to play the game *well* is another matter; it requires very quick reflexes, manual dexterity, and total concentration. I just don't think that very many players will have to

worry about remembering to add both scores when the score display rolls over at the maximum 500,000 points and restarts at 000000!

The game operates on four skill levels. The level chosen determines the number of incubators, concentration of spores, and the speed of attacking aliens. Colored numbers appear inside the incubators and start counting down. When an incubator reaches zero, two or three spores, or an alien hatch. The hatching life-form depends on the skill level chosen, and the stardate the player is presently in. Spores can't attack, but must be destroyed before four of them cluster into a square and form a large, hungry red alien who pursues your ship. It's then you or him, Commander . . . either blast the menacing creature to smithereens, or be gobbled up by the inter-galactic gourmand. If the latter occurs, your initial ship supply of ten is depleted by one, and the fight continues in another section of the galaxy. But if, on the other hand, you're fortunate enough to possess the skill of Luke Skywalker, and destroy all the aliens and all but ten spores, you will receive an extra ship with which to continue your attack in another threatened sector.

You use the four directional arrow keys to control your ship's movement, and either the SHIFT or ENTER key to fire the laser. Alternately, one of the TI Wired Remote Controllers can be used instead of the keyboard. Caution: I don't suggest you use the SHIFT key because of its proximity to the Z key—i.e., an accidental SHIFT Z (easily done in the "heat of battle") will terminate the present game and take you BACK to the level-of-difficulty screen. Therefore, If you don't want your potential record-setting game to be cut short, use the ENTER key (or the wired Remote Controllers) to fire.

This brings up the question of whether or not the Wired Remote Controllers—more commonly known as "joysticks"—provide an advantage. Although this device allows players to keep their eyes concentrated on the screen action without having to occassionally check their finger placement on the keyboard (an often fatal tactical error), the response to commands is markedly slower and less precise. It is definitely more enjoyable and realistic to use a joystick, yet in all but the lowest level of play, I'm afraid you'll eventually yank out the connecting cable of the TI joystick in frustration, and resort to tapping away at the keyboard. [We at *99'er Magazine* have noticed a significant improvement in joystick action by using an Atari joystick that is connected to the TI-99/4 with an adapter manufactured by Denali Data—Ed.] A very skillfull Commander (one who can rub his stomach, pat his head, wink, wiggle both ears, and alternately snap his fingers—all simultaneously) will be able to forgo joysticks altogether, keep his eyes permanently glued to the screen, and rid the cosmos of these malignant Milton Bradley-created menaces . . .

One final tip: Before choosing your skill level and rushing off to do battle, take the time to watch several preprogrammed sample "games" that run continuously once the Command Module is selected. Pay no attention to the actions taken by the "dumb" (randomly firing) Commander. Rather, watch closely how the spores move and cluster to form aliens, and how the aliens move around spores and attack the ship. Try and train your eyes to take in the "big picture" all at once, so that you can anticipate where the next attack will come from.

So go to it, Commander. Show your friends and neighbors that the TI-99/4 now has a game that rivals the best of the video arcades. And think of all those quarters you'll be saving . . .

D-E-BUG

FORUM

In this column, readers are invited to share their discoveries of bugs and/or fixes with the rest of us . . .

Subject:	Missing Dates and Device Default Error in "Checkbook Manager" Software
Fix Status:	Fixable by User; Method Follows
Submitted By:	W.K. Balthrop & G. R. Michaels c/o *99'er Magazine*

We have discovered two bugs in TI's recently released, diskette-based *Checkbook Manager* software package. Both bugs are user-fixable, and shouldn't take more than about 15 minutes to correct.

CHECKBOOK PROGRAM

The first bug is in the CHECKBOOK program. An incorrect variable assignment causes erroneous data to be placed in the date column. You should, therefore, make this first fix prior to starting a data file, or the many hours spent in file creation will have to be repeated.

The procedure we recommend for this, and all diskette-based program changes, is to (1) make a back-up copy of the disk, (2) make and test out the "fix" on this back-up copy, (3) transfer the corrected program to the original disk, (4) put away the original disk for safekeeping, and (5) use the back-up disk as your "work disk." Then, if this program work disk is ever damaged, you'll still have the original disk from which to make a new one. Don't forget to also make a copy of any data disk that you use with a program. If you want the maximum protection from disk damage and its accompanying file loss, data disks have to be re-copied after *each* file update session. (And with *Checkbook Manager* it is your *data disk* that is more likely to be damaged due to the program's frequent reading from and writing to the resident files.)

Assuming you now have a back-up copy of *Checkbook Manager* (if you need instruction in copying disks, see pages 22-24 in the *Disk Memory System* users manual), you are now ready to make the first fix:

1. Select TI-BASIC
2. Insert the *back-up* disk in Drive #1.
3. Type in **OLD DSK1.CHECKBOOK** and then press **ENTER.**
4. When the cursor reappears, either type in **4502** and press **SHIFT** ↓ or type in **EDIT 4502** (either keystroke sequence will put you into the EDIT mode).
5. Line 4502 should appear. If the statement reads **D$=Q$** it is already correct, so just press **ENTER.** If the statement reads **D$=A$** you must use the right-arrow key to position the cursor over the letter **A** and change it to letter **Q.** Now press **ENTER.** Wait until the cursor reappears, type in **LIST 4502** and check that it now reads **D$=Q$**
6. Type in **SAVE DSK1.CHECKBOOK** AND PRESS **ENTER.**
7. **RUN** the program with some "dummy" data according to the TI documentation. You should now see the dates correctly displayed. When you're satisfied that the revised program is working correctly, copy it onto the original TI disk.
8. Put the original TI program disk in a safe place, label your new work disk (with the addition **99'er Rev. 5.81.1**), and use it to manage your computer checkbook.

ACCTSUM PROGRAM

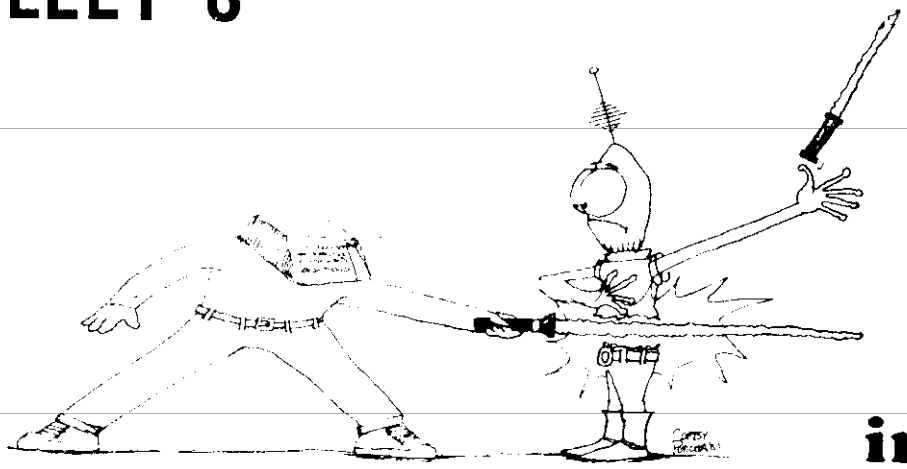
The second fix is in the ACCTSUM program. It is only necessary if you want to use an RS232C-compatible printer. If you will only be using the TI Thermal Printer, the program will work even though it contains the bug (because it automatically defaults to the "TP" device name).

We won't go through the entire step-by-step procedure as we did with the previous bug, since this "fix" requires several code changes. Follow the same general test and transfer procedure.

1. In line 210, move the closing " to the space immediately after **NAME?**
2. Insert new line **396 GOSUB 800**
3. Delete line 410
4. Change line 490 to read
   490 PRINT "DEFAULT""TP""THERMAL PRINTER"
5. Insert new line **501 IF PRINT$>" " THEN 510**
6. Insert new line **502 PRINT$="TP"**

The corrected program should now (as it was originally intended) allow you to type in your device name (e.g., **RS232/2.BA=9600**) and send the printed ACCTSUM output to your external printer.

KELLEY'S KORNER



Space & Combat Games in TI BASIC

Forget all the educational and technical stuff you've been reading in the rest of this magazine. Sure, it's been interesting and informative . . . but you need some fun too! Right? Relax then. You're in *my* territory now: Kelley's Korner—*the* place for great graphic games and sensational simulations.

Call me Kelley. I'll be your guide here, and will lead you through a lush landscape of action, adventure, and challenge. On this first trip we'll allow our imaginations to soar, and let our home computers pit us against devious foes bent on our destruction.

I've chosen three confrontations to test your mettle. You're in for a smorgasbord of hypnotic involvement, relentless pressure, and constant decision-making . . . and most of all—**FUN!**

But lest I be criticized by others on the staff of this magazine for "not teaching you anything of value," I can only suggest to these critics that they look very closely at my carefully chosen itinerary for this first issue's excursion into the world of action games. If they do so, they'll find—and may be even learn from—an extensive array (no pun intended) of fully documented game programming techniques that come in quite handy in a TI BASIC environment. Enough said. The enemy is at hand. There's no time to waste . . .

Enemy Attack

By Charles Ehninger

The problem with most action games written in TI BASIC is their speed. Because they are so slow, the author must limit himself to no more than three moving objects (as in pong-type games). More than three of these "sprites," and the player soon falls asleep. This is the hammer with which my Assembly programmer friends beat me over the head every time we have coffee together. I am a BASIC programmer, I love challenges, and I don't want any more lumps on my head. So I accepted, and here is my reply: An action game with *many* "sprites" that is guaranteed to keep you awake until 3 o'clock in the morning.

My first task was to design the game. No more space or intergalactic nonsense for me! I much prefer war games now, so I borrowed an idea from World War II. To be conclusive, my game needed to have several moving objects: I included seven aircraft, 9 missiles and forty-five blinking, changing and disappearing stars. Is the game exciting? You'll be the judge . . . but first let me tell you that I had trouble finishing the program because I found myself playing the game—trying to repel wave after wave of invaders—until 3 or 4 o'clock in the morning!

In this game, you are flying a reconnaissance mission when your radar alerts you to an enemy attack. The enemy is flying three squadrons of six aircraft each, led by the "rookies." Your radio is out, so you can't alert your base. You must fight all by yourself, because if you let a single airplane get behind you (or you let them destroy you), your home base will be lost. Fortunately, your plane is equipped with a triple-missile guidance system. That is, it can monitor (visably on the screen) the flight of up to three missiles simultaneously, whereas the enemy is equipped with single-missile systems. To launch a missile, you must press the ENTER key. (After launching three missiles, the ENTER key will be deactivated until one of your missiles disappears from the screen.) You may also change your location to intercept the enemy. You may move up or down by pressing the corresponding arrow key, or you may remain in one place by depressing any other key. Once the motion is selected, it will remain active until changed. Whatever your command, hold the key down until the "acknowledge tone" is emitted. The tone indicates that your command has been received.

ENEMY ATTACK : List of Variables

1. Numeric Arrays:

Name	Elements	Purpose
C	6x8 (48)	There are six row vectors, each assigned to one of six attacking air craft. the elements are: 1 - Row location of craft and its associated missile. 2 - Columnar location of aircraft. 3 - ASCII code of character assigned to the aircraft. 4 - Indicator which determines whether or not aircraft is "alive."(0=off, 1=on) 5 - Indicator to show a "missile in progress" (0=off, 1=on) 6 - Column location of missile. 7 - Holds background character to replace the present location of aircraft. 8 - Holds background character to replace the present location of missile.
D	3x4 (12)	There are three rows of vectors, one for each "friendly" missile. The elements are: 1 - Row location of missile. 2 - Column location of missile. 3 - ASCII configuration of missile character. 4 - Background character to replace the present location of the missile.
P	3	Foreground color number for each group of attacking aircraft.
R	4	Foreground colors for star group #1.
S	4	Foreground colors for star group #2.

2. Numeric variables:

Name	Purpose
A	Row location of friendly craft.
B	Normally 0; set to 1 if either the friendly aircraft is shot down or an enemy airplane gets through.
E	Incremental motion of friendly aircraft (1=down; −1=up)
F	Row location of explosion.
G	Columnar location of explosion.
H	General purpose (FOR-NEXT loops).
I	General purpose (FOR-NEXT loops).
J	General purpose random number holder.
L	Holds background character to replace the present location of friendly craft.
V	Return variable for input command.
W	Status variable for input command.
EN	Number of enemy aircraft remaining.
FR	Enemy missile launch indicator (0=none; 1=launch)
GM	Used to cycle through the full battle 3 times.

Display Characters:

Number	Assignment:
96	"Friendly" aircraft.
97	White explosion cloud.
104	"Enemy" aircraft.
105	Red, yellow or blue cloud.
112	"Fixed" stars.
113	Yellow cloud.
120	"Friendly" missile.
121	Red cloud.
122	"Enemy" missile.

EXPLANATION OF THE PROGRAM

Line Nos.	
100-390	Program initialization: Character assignments and color designations.
400-690	Display of game title and introductory music.
700-760	Switch star colors at random.
770-1540	**Defense**: The defensive cycle consists of three major routines: Input command (770-1000), missile advance (1010-1320), and aircraft motion (1330-1540). *Input command:* The player controls the firing of a missile (ENTER key (13)), upward motion of the aircraft (UP-arrow (69)), downward motion (DOWN-arrow (88)) or he may command the airplane to remain in present position (any other key). *Missile advance:* Lines 1050-1320 are executed three times, once for each missile. Line 1060 is a quick bypass for unfired missiles. *Aircraft motion:* The aircraft is repositioned by incrementing A with the contents of E. Note that E may be any integer between −1 and +1. This determines whether the craft ascends, decends or remains in place.
1550-2170	**Offense**: The offensive cycle also includes three phases: Missile firing (lines 1550-1690), missile advance (1700-1890) and aircraft advance (1900-2170). *Missile firing:* Since each craft may sustain only one active missile, line 1610 checks to determine if there is a missile already in progress. If not, a second test is made in line 1620 to insure that the craft is still "alive." Finally, line 1630 prevents the firing of missiles when the plane is at or beyond display column 30. Line 1640 generates a random number within ever-narrowing ranges and line 1650 bypasses missile firing if the number is greater than zero. If a missile is to be fired, indicator FR is turned on. *Missile advance:* Active missiles are advanced one position and any missile-to-missile or missile-to-craft conflict is resolved. *Aircraft advance:* If a missile has just been fired (line 1900) or a randomly-generated number is positive (line 1910-1920) aircraft motion is suspended; otherwise, the enemy craft advances one position.
2180-2260	Explosion simulation. An explosion cloud is displayed at location F,G.
2270-2540	Initial scenario. Sky and friendly craft are positioned.
2550-2710	Initialization of enemy aircraft.
2720-2770	Program root. This is the game "clock".
2780-2800	"YOU WIN" message.
2810-2820	"YOU LOSE" message.
2830-2890	"REPEAT?" and loop or end.

```
100 REM     ******************
110 REM     * ENEMY ATTACK *
120 REM     ******************
121 REM  BY CHARLES EHNINGER
122 REM  99'ER VERSION 5.81.1
130 OPTION BASE 1
140 DIM C(6,8),D(3,4),P(3),R(4),S(4)
150 CALL CLEAR
160 CALL SCREEN(2)
170 FOR I=1 TO 3
180 READ P(I)
190 NEXT I
200 FOR I=1 TO 4
210 READ R(I),S(I)
220 NEXT I
230 DATA 8,11,7
240 DATA 1,16,5,16,16,5,16,1
250 CALL CHAR(96,"303039FF39303000")
260 CALL CHAR(97,"12343EFC7F2F4A90")
270 CALL CHAR(136,"08")
280 CALL CHAR(128,"08")
290 CALL CHAR(104,"0C0C9CFF9C0C0C")
```

continued

Enemy Attack, continued from p. 41

```
300 CALL CHAR(105,"12343EFC7F2F4A90")
310 CALL CHAR(112,"1818")
320 CALL CHAR(113,"12343EFC7F2F4A90")
330 CALL CHAR(120,"00000EFC0E")
340 CALL CHAR(121,"12343EFC7F2F4A90")
350 CALL CHAR(122,"0000703F70")
360 CALL COLOR(9,16,1)
370 CALL COLOR(10,2,1)
380 CALL COLOR(11,12,1)
390 CALL COLOR(12,7,1)
400 REM **********
410 REM * TITLE *
420 REM **********
430 DISPLAY TAB(10);"ENEMY ATTACK!"::::::::::::
440 CALL HCHAR(6,7,104,21)
450 CALL VCHAR(7,27,120,13)
460 CALL HCHAR(20,7,96,21)
470 CALL VCHAR(7,7,122,13)
480 CALL SCREEN(8)
490 FOR F=6 TO 20 STEP 14
500 FOR G=8 TO 26 STEP 8
510 GOSUB 2210
520 NEXT G
530 FOR I=1 TO 500
540 NEXT I
550 GOSUB 580
560 NEXT F
570 GOTO 2300
580 CALL SOUND(300,440,0,-4,0)
590 CALL SOUND(150,523,0,-4,0)
600 CALL SOUND(450,523,0,-4,0)
610 CALL SOUND(150,466,0,-4,0)
620 CALL SOUND(150,440,0,-4,0)
630 CALL SOUND(150,392,0,-4,0)
640 CALL SOUND(450,440,0,-4,0)
650 CALL SOUND(450,466,0,-4,0)
660 CALL SOUND(450,494,0,-4,0)
670 CALL SOUND(450,523,0,-4,0)
680 RETURN
690 GOTO 2300
700 REM **********
710 REM * STARS *
720 REM **********
730 F=INT(4*RND)+1
740 CALL COLOR(13,R(F),1)
750 CALL COLOR(14,S(F),1)
760 RETURN
770 REM ***********
780 REM * DEFEND *
790 REM ***********
800 CALL KEY(0,V,W)
810 IF W=0 THEN 1040
820 IF V<>13 THEN 930
830 CALL SOUND(-400,440,0,880,0,-4,0)
840 FOR I=1 TO 3
850 IF D(I,3)<>0 THEN 910
860 D(I,1)=A
870 D(I,2)=31
880 D(I,3)=1
890 D(I,4)=96
900 GOTO 1040
910 NEXT I
920 GOTO 1040
930 CALL SOUND(-400,110,0,139,0,-8,0)
940 IF V<>69 THEN 970
950 E=-1
960 GOTO 1040
970 IF V<>88 THEN 1000
980 E=1
990 GOTO 1040
1000 E=0
1010 REM *************
1020 REM * DEF FIRE *
1030 REM *************
1040 GOSUB 730
1050 FOR I=1 TO 3
1060 IF D(I,3)=0 THEN 1320
1070 CALL HCHAR(D(I,1),D(I,2),D(I,4))
1080 IF D(I,2)=1 THEN 1270
1090 D(I,2)=D(I,2)-1
1100 CALL GCHAR(D(I,1),D(I,2),D(I,4))
1110 IF (D(I,4)=32)+(D(I,4)=112)+(D(I,4)>127)<>0
     THEN 1300
1120 IF D(I,4)=120 THEN 1270
1130 F=D(I,1)
1140 G=D(I,2)
1150 GOSUB 2210
1160 FOR H=1 TO 6
1170 IF C(H,1)<>D(I,1)THEN 1260
1180 IF D(I,4)<>122 THEN 1220
1190 C(H,5)=0
1200 C(H,6)=0
1210 GOTO 1270
1220 C(H,2)=0
1230 C(H,4)=0
1240 EN=EN-1
1250 GOTO 1270
1260 NEXT H
1270 D(I,2)=0
1280 D(I,3)=0
1290 GOTO 1320
1300 CALL HCHAR(D(I,1),D(I,2),120)
1310 CALL SOUND(-100,110,8,-6,4)
1320 NEXT I
1330 REM *************
1340 REM * DEF MOVE *
1350 REM *************
1360 IF E=0 THEN 1540
1370 CALL HCHAR(A,31,L)
1380 A=A+E
1390 IF A THEN 1420
1400 A=1
1410 E=0
1420 IF A<25 THEN 1450
1430 A=24
1440 E=0
1450 CALL GCHAR(A,31,L)
1460 IF (L=32)+(L=112)+(L>127)<>0 THEN 1520
1470 F=A
1480 G=31
1490 GOSUB 2210
1500 B=1
1510 GOTO 1540
1520 CALL HCHAR(A,31,96)
1530 CALL SOUND(50,440,8,-4,0)
1540 RETURN
1550 REM **********
1560 REM * ENEMY *
1570 REM **********
1580 GOSUB 730
1590 FOR I=1 TO 6
1600 FR=0
1610 IF C(I,5)<>0 THEN 1700
1620 IF C(I,4)=0 THEN 2160
1630 IF C(I,2)>30 THEN 1900
1640 J=INT(GM*EN*RND)
1650 IF J THEN 1900
1660 C(I,5)=1
1670 C(I,6)=C(I,2)
1680 C(I,8)=C(I,3)
1690 FR=1
1700 CALL HCHAR(C(I,1),C(I,6),C(I,8))
1710 C(I,6)=C(I,6)+1
1720 IF C(I,6)=32 THEN 1850
1730 CALL GCHAR(C(I,1),C(I,6),C(I,8))
1740 IF (C(I,8)=32)+(C(I,8)=112)+(C(I,8)>127)<>0
     THEN 1880
1750 F=C(I,1)
1760 G=C(I,6)
1770 GOSUB 2210
1780 IF C(I,8)=96 THEN 2020
1790 FOR H=1 TO 3
1800 IF D(H,2)<>C(I,6)THEN 1840
1810 D(H,2)=0
1820 D(H,3)=0
1830 GOTO 1850
1840 NEXT H
1850 C(I,5)=0
1860 C(I,6)=0
1870 GOTO 1910
1880 CALL HCHAR(C(I,1),C(I,6),122)
1890 CALL SOUND(-200,165,16,-5,12)
1900 IF FR THEN 2160
1910 J=INT(GM*EN*RND)
1920 IF J THEN 2160
1930 CALL HCHAR(C(I,1),C(I,2),C(I,7))
1940 C(I,2)=C(I,2)+1
1950 IF C(I,2)>32 THEN 2020
1960 CALL GCHAR(C(I,1),C(I,2),C(I,7))
1970 IF (C(I,7)=32)+(C(I,7)=112)+(C(I,7)>127)<>0
     THEN 2140
1980 F=C(I,1)
1990 G=C(I,2)
2000 GOSUB 2210
2010 IF C(I,7)<>96 THEN 2040
2020 B=1
2030 GOTO 2170
```

continued on p. 45

Invasion from Space

By W. K. Balthrop

After producing my own TI BASIC version of the familiar "Space Invaders" game, it was immediately obvious to me why the arcade version enjoyed its great popularity—*SPEED*. Since my game had over *ninety* moving objects, it was easy to become bored waiting around while they all moved. I strongly felt that a computer game should be challenging enough to hold a player's interest *regardless* of its interaction speed, so it was back to the planning stage . . .

The solution I came up with was fairly straightforward, but added a new level of strategic sophistication to the basic "you-blast-me-I-blast-you" game plan. Boredom immediately became a thing of the past. The first change was to equip the aliens with multiple warhead missiles. At a given elevation, each missile will split into three warheads—each traveling in a different direction. To compensate for this obvious breach in the galactic SALT agreement, I have equipped your interceptor jet with the capability of firing its laser in three directions at any of the sixty alien ships. You can also destroy warheads that are in a direct flight path with your jet, or hide behind any of the three barriers. But don't sit still forever—this temporary protection won't last long once the multiple warheads start striking!

The main difficulty in writing *Invasion from Space* was in keeping it simple for the player while producing the fastest program possible. By keeping data (status and locations) for the alien ships and missiles in arrays, I was able to cycle through each of them in FOR-NEXT loops. Next in importance was to break all functions into subroutines, so they could be accessed at any time in the main control loop. Four main subroutines make up the program:

1. The main control loop: Updates enemy positions, branches off to the other three subroutines, and reads the keyboard.

2. The defensive routine: Moves the jet left or right; fires jet missiles, and checks for hits.

3. The offensive routine: Selects an alien ship to fire a missile; controls the missile's direction, multiple warhead split, and checks for any hits.

4. The scoring display: Adds or subtracts points from your score; branches to end of game messages.

EXPLANATION OF THE PROGRAM

Line Nos.	
190-600	Initialization: Set up variables, character definition, and color assignment.
610-800	Print instruction page.
810-1090	Display: playing screen.
1100-1610	Main program control loop.
1100-1390	Move alien ships left or right.
1400-1450	Branch to subroutines.
1460-1500	Check enemy position for edge of screen; reverse direction.
1510-1610	Read keyboard; branch to jet control subroutines.
1620-2110	Defensive subroutines.
1620-1700	Move-jet-left subroutine.
1710-1790	Move-jet-right subroutine.
1800-2050	Calculate new score when enemy is hit: 5 , 10 , 25 , or 50 points.
2060-2110	Barrier hit by friendly missile: score=score −5 points.
2120-3030	Offensive subroutines.
2120-2380	Select alien ship to fire missile.
2390-2780	Fire enemy missile and advance position.
2790-2920	Barrier hit by enemy missile.
2930-3030	Player's jet hit by alien missile.
3040-3510	End-of-game messages; Next game, or end.
3520-3620	Re-initialize variables for the next game.
3630-3780	Display score.
3790-4540	Defensive subroutines.
3790-4020	Fire users missile in one of three directions.
4030-4130	Check for type of hit by users missile.
4290-4540	Enemy missile is destroyed by friendly missile Score=score+5.
4550-4590	End of program message.

```
90 REM      **********************
95 REM      *                    *
100 REM     *INVASION FROM SPACE*
105 REM     *                    *
106 REM     **********************
110 REM   BY W.K. BALTHROP
120 REM   99'ER VERSION 5.81.1
130 REM
140 REM
150 REM    DESTROY ALL OF THE
160 REM     ENEMY SHIPS BEFORE
170 REM      THEY DESTROY YOU!
180 REM     ---INITIALIZATION--
190 RANDOMIZE
200 DIM A(2,12)
210 DIM B(2,12)
220 DIM C(2,36)
230 DIM M(6,30)
240 FOR X=1 TO 12
250 A(2,X)=(X*2)+4
260 A(1,X)=3
270 B(2,X)=(X*2)+4
280 B(1,X)=5
290 C(2,X)=(X*2)+4
300 C(2,X+12)=(X*2)+4
310 C(2,X+24)=(X*2)+4
320 C(1,X)=7
330 C(1,X+12)=9
340 C(1,X+24)=11
350 NEXT X
360 P1=24
370 P2=16
380 CALL CLEAR
390 CALL SCREEN(2)
400 CALL COLOR(10,10,1)
410 CALL COLOR(9,6,1)
420 CALL COLOR(11,4,1)
430 CALL COLOR(12,7,1)
440 FOR X=2 TO 8
450 CALL COLOR(X,16,1)
460 NEXT X
470 CALL CHAR(96,"0008143E6B491C3E")
480 CALL CHAR(104,"18187EFFFF7E2442")
490 CALL CHAR(112,"FFFFFFFFFFFFFFFF")
500 CALL CHAR(113,"00000000081C3FF")
510 CALL CHAR(114,"80B0C0C0F0F8FCFF")
520 CALL CHAR(115,"01010303071F7FFF")
530 CALL CHAR(120,"0808080808080808")
540 CALL CHAR(121,"8040201008040201")
550 CALL CHAR(122,"0102040810204080")
560 CALL CHAR(105,"7E7E9999BD9918")
570 CALL CHAR(106,"7E3C183C7E7E3C18")
580 CALL CHAR(107,"1008100810083C18")
590 CALL CHAR(108,"00020408506070")
600 CALL CHAR(109,"004020100A060E")
610 REM    -PRINT FIRST PAGE-
620 PRINT "****INVASION FROM SPACE*****'
630 PRINT :::
640 PRINT "  YOU MUST DESTROY THE ENEMY'
650 PRINT
660 PRINT "  BEFORE THEY DESTROY YOU"
670 PRINT
680 PRINT CHR$(107);" =5 PTS."
690 PRINT
700 PRINT CHR$(104);" =10 PTS."
710 PRINT
720 PRINT CHR$(105);" =25 PTS."
730 PRINT
740 PRINT CHR$(106);" =50 PTS."
750 PRINT
760 PRINT "S= MOVE LEFT"
770 PRINT "D= MOVE RIGHT"
780 PRINT "W= FIRE CANNON LEFT"
790 PRINT "E= FIRE CANNON UP"
800 PRINT "R= FIRE CANNON RIGHT"
810 REM    SET UP PLAYING
820 REM    SCREEN
```

continued on p. 44

Invasion from Space, continued from p. 43

```
830 PRINT "PRESS ANY KEY TO START"
840 CALL KEY(0,K,S)
850 IF S=0 THEN 840
860 CALL CLEAR
870 FOR X=1 TO 12
880 CALL VCHAR(A(1,X),A(2,X),106)
890 CALL VCHAR(B(1,X),B(2,X),105)
900 CALL VCHAR(C(1,X),C(2,X),104)
910 CALL VCHAR(C(1,X+12),C(2,X+12),104)
920 CALL VCHAR(C(1,X+24),C(2,X+24),104)
930 NEXT X
940 CALL HCHAR(20,5,112,3)
950 CALL HCHAR(21,4,112,5)
960 CALL HCHAR(22,4,112,5)
970 CALL HCHAR(20,15,112,3)
980 CALL HCHAR(21,14,112,5)
990 CALL HCHAR(22,14,112,5)
1000 CALL HCHAR(20,25,112,3)
1010 CALL HCHAR(21,24,112,5)
1020 CALL HCHAR(22,24,112,5)
1030 CALL VCHAR(24,16,96)
1040 CALL VCHAR(1,3,83)
1050 CALL VCHAR(1,4,67)
1060 CALL VCHAR(1,5,79)
1070 CALL VCHAR(1,6,82)
1080 CALL VCHAR(1,7,69)
1090 D=-1
1100 REM    MOVE ENEMY LEFT
1110 REM    AND RIGHT
1120 IF A(2,1)=2 THEN 1150
1130 IF A(2,12)=31 THEN 1170
1140 GOTO 1180
1150 D=1
1160 GOTO 1180
1170 D=-1
1180 W=1
1190 X=1
1200 A(2,X)=A(2,X)+D
1210 B(2,X)=B(2,X)+D
1220 C(2,X)=C(2,X)+D
1230 C(2,X+12)=C(2,X+12)+D
1240 C(2,X+24)=C(2,X+24)+D
1250 IF A(1,X)=0 THEN 1280
1260 CALL VCHAR(A(1,X),A(2,X),106)
1270 CALL VCHAR(A(1,X),A(2,X)-D,32)
1280 IF B(1,X)=0 THEN 1310
1290 CALL VCHAR(B(1,X),B(2,X),105)
1300 CALL VCHAR(B(1,X),B(2,X)-D,32)
1310 IF C(1,X)=0 THEN 1340
1320 CALL VCHAR(C(1,X),C(2,X),104)
1330 CALL VCHAR(C(1,X),C(2,X)-D,32)
1340 IF C(1,X+12)=0 THEN 1370
1350 CALL VCHAR(C(1,X+12),C(2,X+12),104)
1360 CALL VCHAR(C(1,X+12),C(2,X+12)-D,32)
1370 IF C(1,X+24)=0 THEN 1420
1380 CALL VCHAR(C(1,X+24),C(2,X+24),104)
1390 CALL VCHAR(C(1,X+24),C(2,X+24)-D,32)
1400 REM  BRANCH TO
1410 REM  SUBROUTINES
1420 GOSUB 1530
1430 GOSUB 3630
1440 GOSUB 2150
1450 GOSUB 3630
1460 REM    CHECK ENEMY POS.
1470 W=W+1
1480 X=X+1
1490 IF POINTS>1090 THEN 3280
1500 IF W<13 THEN 1200 ELSE 1120
1510 REM    SUB-ROUTINE
1520 REM    OPERATOR INPUTS
1530 CALL KEY(0,K,S)
1540 IF S<>0 THEN 1560
1550 RETURN
1560 IF K=83 THEN 1630
1570 IF K=68 THEN 1720
1580 IF K=69 THEN 4260
1590 IF K=87 THEN 4180
1600 IF K=82 THEN 4220
1610 RETURN
1620 REM    MOVE LEFT
1630 P2=P2-1
1640 IF P2=1 THEN 1660
1650 GOTO 1670
1660 P2=P2+1
1670 CALL VCHAR(P1,P2+1,32)
1680 CALL VCHAR(P1,P2,96)
1690 CALL SOUND(100,220,2)
1700 RETURN
1710 REM    MOVE RIGHT
1720 P2=P2+1
1730 IF P2=32 THEN 1750
1740 GOTO 1760
1750 P2=P2-1
1760 CALL VCHAR(P1,P2-1,32)
1770 CALL VCHAR(P1,P2,96)
1780 CALL SOUND(100,220,2)
1790 RETURN
1800 REM    CALCULATION
1810 REM     ENEMY DESTROYED
1820 POINTS=POINTS+10
1830 CALL SOUND(200,110,2,-7,2)
1840 FOR Y=1 TO 36
1850 IF C(1,Y)<>Z THEN 1880
1860 IF C(2,Y)<>Z1 THEN 1880
1870 C(1,Y)=0
1880 NEXT Y
1890 RETURN
1900 POINTS=POINTS+25
1910 CALL SOUND(200,110,2,-7,2)
1920 FOR Y=1 TO 12
1930 IF B(1,Y)<>Z THEN 1960
1940 IF B(2,Y)<>Z1 THEN 1960
1950 B(1,Y)=0
1960 NEXT Y
1970 RETURN
1980 POINTS=POINTS+50
1990 CALL SOUND(200,110,2,-7,2)
2000 FOR Y=1 TO 12
2010 IF A(1,Y)<>Z THEN 2040
2020 IF A(2,Y)<>Z1 THEN 2040
2030 A(1,Y)=0
2040 NEXT Y
2050 RETURN
2060 REM    FORTRESS DESTROYED
2070 REM    BY OPERATOR
2080 CALL SOUND(100,220,2,-6,2)
2090 CALL SOUND(200,110,2,-6,2)
2100 POINTS=POINTS-5
2110 RETURN
2120 REM    CALCULATION
2130 REM    SELECT SHIP TO
2140 REM    DROP THE MISSLE
2150 M2=M2+1
2160 M1=INT(RND*12)+1
2170 IF C(1,M1+24)=0 THEN 2210
2180 M(1,M2)=C(1,M1+24)+1
2190 M(2,M2)=C(2,M1+24)
2200 GOTO 2400
2210 IF C(1,M1+12)=0 THEN 2250
2220 M(1,M2)=C(1,M1+12)+1
2230 M(2,M2)=C(2,M1+12)
2240 GOTO 2400
2250 IF C(1,M1)=0 THEN 2290
2260 M(1,M2)=C(1,M1)+1
2270 M(2,M2)=C(2,M1)
2280 GOTO 2400
2290 IF B(1,M1)=0 THEN 2330
2300 M(1,M2)=B(1,M1)+1
2310 M(2,M2)=B(2,M1)
2320 GOTO 2400
2330 IF A(1,M1)=0 THEN 2360
2340 M(1,M2)=A(1,M1)+1
2350 M(2,M2)=A(2,M1)
2360 W1=W1+1
2370 IF W1=60 THEN 2990
2380 GOTO 2160
2390 REM    DROP MISSLE
2400 FOR X2=1 TO 29
2410 FOR X3=1 TO 5 STEP 2
2420 IF M(X3,X2)=0 THEN 3360
2430 IF X3>1 THEN 2490
2440 IF M(1,X2)<>18 THEN 2490
2450 M(3,X2)=M(1,X2)
2460 M(4,X2)=M(2,X2)
2470 M(5,X2)=M(1,X2)
2480 M(6,X2)=M(2,X2)
2490 IF M(X3,X2)=24 THEN 2520
2500 IF M(X3+1,X2)<2 THEN 2520
2510 IF M(X3+1,X2)<31 THEN 2570
2520 CALL GCHAR(M(X3,X2),M(X3+1,X2),CH)
2530 IF CH=96 THEN 2950
2540 CALL VCHAR(M(X3,X2),M(X3+1,X2),32)
2550 M(X3,X2)=0
2560 GOTO 3360
2570 ON INT(X3/2+1)GOTO 2580,2610,2640
2580 CALL GCHAR(M(X3,X2)+1,M(X3+1,X2),CH)
```

continued on p. 45

Enemy Attack, continued from p. 42

```
2040 FOR H=1 TO 3
2050 IF D(H,2)<>C(I,2)THEN 2090
2060 D(H,2)=0
2070 D(H,3)=0
2080 GOTO 2100
2090 NEXT H
2100 C(I,2)=0
2110 C(I,4)=0
2120 EN=EN-1
2130 GOTO 2160
2140 CALL HCHAR(C(I,1),C(I,2),C(I,3))
2150 CALL SOUND(-100,659,0,-8,0)
2160 NEXT I
2170 RETURN
2180 REM ********
2190 REM * BANG *
2200 REM ********
2210 FOR H=121 TO 97 STEP -8
2220 CALL HCHAR(F,G,H)
2230 CALL SOUND(-1000,110,0,165,0,-7,0)
2240 NEXT H
2250 CALL HCHAR(F,G,32)
2260 RETURN
2270 REM **********
2280 REM * START *
2290 REM **********
2300 CALL CLEAR
2310 CALL SCREEN(2)
2320 GM=3
2330 GOSUB 730
2340 B=0
2350 A=12
2360 L=32
2370 FOR I=1 TO 3
2380 D(I,3)=0
2390 D(I,4)=32
2400 NEXT I
2410 RANDOMIZE
2420 FOR I=1 TO 60
2430 F=INT(24*RND)+1
2440 G=INT(32*RND)+1
2450 CALL SOUND(-50,880,8)
2460 IF I>15 THEN 2490
2470 CALL HCHAR(F,G,112)
2480 GOTO 2530
2490 IF INT((F+G)/2)=(F+G)/2 THEN 2520
2500 CALL HCHAR(F,G,128)
2510 GOTO 2530
2520 CALL HCHAR(F,G,136)
2530 NEXT I
2540 CALL HCHAR(A,31,96)
2550 CALL COLOR(10,P(GM),1)
2560 FOR I=1 TO 6
2570 J=INT(24*RND)+1
2580 FOR H=1 TO I-1
2590 IF J=C(H,1)THEN 2570
2600 NEXT H
2610 C(I,1)=J
2620 C(I,2)=1
2630 C(I,3)=104
2640 C(I,4)=1
2650 C(I,5)=0
2660 C(I,6)=1
2670 C(I,7)=32
2680 C(I,8)=104
2690 CALL HCHAR(C(I,1),C(I,2),C(I,3))
2700 CALL SOUND(50,1319,0)
2710 NEXT I
2720 EN=6
2730 GOSUB 800
2740 GOSUB 1580
2750 IF B=1 THEN 2820
2760 IF EN THEN 2730
2770 GM=GM-1
2780 IF GM THEN 2550
2790 CALL CLEAR
2800 DISPLAY TAB(10);"OUR HERO!":::"YOU HAVE
     SAVED YOUR COUNTRY"::::::::::
2810 GOTO 2840
2820 CALL CLEAR
2830 DISPLAY TAB(5);"SEE WHAT YOU'VE DONE?":::
     "YOU LET THE ENEMY DESTROY":"YOUR
     COUNTRY!":::::::::::::
2840 DISPLAY "WANNA TRY AGAIN";
2850 CALL SCREEN(8)
2860 INPUT K$
2870 IF SEG$(K$,1,1)="Y"THEN 2300
2880 CALL CLEAR
2890 END
```

Invasion from Space, continued from p. 44

```
2590 X4=0
2600 GOTO 2660
2610 CALL GCHAR(M(X3,X2)+1,M(X3+1,X2)-1,CH)
2620 X4=-1
2630 GOTO 2660
2640 CALL GCHAR(M(X3,X2)+1,M(X3+1,X2)+1,CH)
2650 X4=1
2660 IF CH=112 THEN 2820
2670 IF CH=96 THEN 2950
2680 M(X3,X2)=M(X3,X2)+1
2690 M(X3+1,X2)=M(X3+1,X2)+X4
2700 CALL VCHAR(M(X3,X2)-1,M(X3+1,X2)-X4,32)
2710 CALL SOUND(50,440,10)
2720 ON INT(X3/2+1)GOTO 2730,2750,2770
2730 CALL VCHAR(M(1,X2),M(2,X2),107)
2740 GOTO 3360
2750 CALL VCHAR(M(3,X2),M(4,X2),108)
2760 GOTO 3360
2770 CALL VCHAR(M(5,X2),M(6,X2),109)
2780 GOTO 3360
2790 REM     CALCULATION
2800 REM     FORTRESS HIT BY
2810 REM     MISSLE
2820 CALL VCHAR(M(X3,X2),M(X3+1,X2),32)
2830 ON INT(X3/2+1)GOTO 2840,2860,2880
2840 CALL VCHAR(M(1,X2)+1,M(2,X2),113)
2850 GOTO 2890
2860 CALL VCHAR(M(3,X2)+1,M(4,X2)-1,114)
2870 GOTO 2890
2880 CALL VCHAR(M(5,X2)+1,M(6,X2)+1,115)
2890 POINTS=POINTS-5
2900 CALL SOUND(200,110,2,330,5,-5,2)
2910 M(X3,X2)=0
2920 GOTO 3370
2930 REM     OPERATOR HIT BY
2940 REM     MISSLE
2950 CALL CLEAR
2960 CALL SOUND(1000,110,2,220,2,330,2,-4,2)
2970 REM     SELECT END OF
2980 REM     GAME MESSAGE
2990 IF POINTS<400 THEN 3060
3000 IF POINTS<700 THEN 3100
3010 IF POINTS<900 THEN 3150
3020 IF POINTS<1000 THEN 3210
3030 GOTO 3280
3040 REM     PRINT END OF GAME
3050 REM     MESSAGE
3060 PRINT "YOU BLEW IT. THE ALIENS HAVE"
3070 PRINT "TAKEN OVER THE EARTH"
3080 PRINT "YOUR SCORE IS:";POINTS
3090 GOTO 3410
3100 PRINT "YOU HAVE LOST THE WAR WITH"
3110 PRINT "THE ALIENS.YOU MUST MAKE TAX"
3120 PRINT "PAYMENTS TO THEM"
3130 PRINT "YOUR SCORE IS:";POINTS
3140 GOTO 3410
3150 PRINT "BOTH THE EARTH, AND THE"
3160 PRINT "ALIENS HAVE HAD GREAT LOSSES"
3170 PRINT "IT WILL TAKE MANY YEARS TO"
3180 PRINT "REBUILD EARTH"
3190 PRINT "YOUR SCORE IS:";POINTS
3200 GOTO 3410
3210 PRINT "YOU FOUGHT THEM OFF BUT"
3220 PRINT "SUFFERED MANY LOSSES"
3230 PRINT "THE EARTH WILL RISE TO POWER"
3240 PRINT "AGAIN SOME DAY"
3250 PRINT "YOUR SCORE IS:";POINTS
3260 CALL SOUND(200,440,2,740,2)
```

continued on p. 54

Anti-Aircraft Gun

By Mark Moseley

Despite the fact that action games programmed in a high-level language such as TI BASIC run much slower than in low-level languages such as 9900 assembly language or GPL (the programming language of TI's Command Modules), it is indeed possible to create reasonably fast "real-time" games if you observe a few rules:

1. Keep the number of moving objects to a minimum.
2. Keep all unnecessary statements out of loops used to move objects.
3. Use only *one* character to define objects you want to move fast.
4. Increment the positions by *two* spaces each loop. This makes the movement slightly jerky, but contributes greatly to the illusion of speed.

I've followed these rules in writing *Anti-Aircraft Gun*. The basic idea of the game is simple: you must shoot down an attacking plane with your missile launcher before it blasts you twice with its laser. The plane attacks at random heights from both the left and right sides. Its speed and frequency of laser fire are dependent on the skill level you choose. Your missile launcher can move along the ground, and even hide behind a barrier; but when it fires a missile, it is committed to its last position until the missed shot passes off the screen. You'll have to move around as much as possible because the plane "remembers" your last position with its higher probability of firing the laser near that position. And don't expect too much protection from the barrier: After five laser blasts (or less, if you launch missiles through it), it will be disintegrated and leave you exposed.

PROGRAM STRUCTURE FOR ANTI-AIRCRAFT GUN
EXPLANATION OF THE PROGRAM

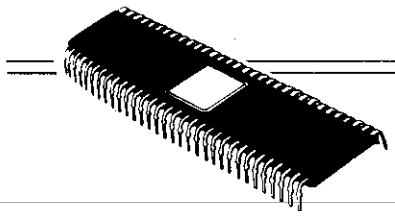
Line Nos.	
100-670	Instructions.
680-810	Set up levels of difficulty.
820-870	Set up variables to make plane fire more as difficulty increases.
880-1110	Character definitions and color assignments.
1120-1170	Initial displaying of tank.
1180-1220	Display ground.
1230-1260	Calculate plane's height.
1270-1380	Determine the direction of the plane.
1390-1450	Read keyboard, and branch to subroutines.
1460-1530	Fire tank's rocket.
1540-1610	Move plane.
1620-1650	Decide whether plane will fire or not.
1660-1760	Fire plane's laser; check for hits.
1770-1780	Check for a hit on the plane by the tank's rocket.
1790-1890	Check for plane at the edge of the screen.
1900-2030	Determine new direction for the plane.
2040-2090	If plane hits the tank at the same time the tank hits the plane, the tank wins.
2100-2180	Determine new direction for the plane.
2190-2330	Calculate score.
2340-2420	Print score.
2430-2460	Plane is destroyed by the tank.
2470-2620	Calculate if a free game was won; starts over.
2630-2750	Move tank left.
2760-2880	Move tank right.
2900	END.

```
100 REM  **********************
110 REM  *                    *
120 REM  * ANTI-AIRCRAFT GUN *
130 REM  *                    *
140 REM  **********************
150 REM   BY MARK MOSELEY
160 REM   99'ER VERSION 5.81.1
170 CALL CLEAR
180 CALL SCREEN(8)
190 REM  * INTRODUCTION *
200 PRINT TAB(3);"PRESS I FOR INSTRUCTIONS"
210 PRINT TAB(8);"AND N FOR NONE"::::::::::
220 CALL KEY(0,LP,PL)
230 IF PL=0 THEN 220
240 IF LP=78 THEN 680
250 PRINT TAB(5);"* ANTI-AIRCRAFT GUN *"
260 PRINT
270 PRINT "THE OBJECT OF ANTI-AIRCRAFT"
280 PRINT "IS TO DESTROY AS MANY PLANES"
290 PRINT "AS POSSIBLE. TO FIRE YOUR "
300 PRINT "MISSLE, PRESS Q. TO MOVE "
310 PRINT "LEFT, PRESS S, AND FOR RIGHT"
320 PRINT "PRESS D."
330 PRINT
340 PRINT TAB(9);"* PLANES: *"
350 PRINT
360 PRINT "THE PLANE FIRES A NEWLY-"
370 PRINT "DEVELOPED LASER. THE PLANE"
380 PRINT "MAY COME FROM LEFT OR RIGHT."
390 PRINT "WARNING! IT HAS A RADAR THAT"
400 PRINT "REMEMBERS YOUR LAST FIRING"
410 PRINT "POSITON AND TRIES TO GET YOU"
420 PRINT "THERE,SO BETTER MOVE AROUND."
430 PRINT :::
440 PRINT "PRESS ANYTHING TO CONTINUE"
450 PRINT
460 CALL KEY(0,KL,LK)
470 IF LK=0 THEN 460
480 PRINT TAB(8);"* BARRIER: *"
490 PRINT
500 PRINT "THE LASER CAN'T PENETRATE "
510 PRINT "THE BARRIER, BUT THE BARRIER"
520 PRINT "CAN SUSTAIN ONLY 5 DIRECT "
530 PRINT "HITS. YOU CAN FIRE WHEN BE-"
540 PRINT "HIND IT, BUT THIS WILL ONLY"
550 PRINT "SHORTEN ITS LIFE."
560 PRINT
570 PRINT TAB(8);"* SCORING: *"
580 PRINT
590 PRINT "PLANES ARE SCORED ACCORDING"
600 PRINT "TO HEIGHT: 5 FOR LOWEST, 20"
610 PRINT "FOR HIGHEST. IF YOUR SCORE "
620 PRINT "IS OVER 100 THEN YOU GET A "
630 PRINT "FREE BONUS GAME."
640 PRINT ::::::
650 PRINT "PRESS ANYTHING TO CONTINUE"
660 CALL KEY(0,KL,LK)
670 IF LK=0 THEN 660
680 CALL CLEAR
690 CALL COLOR(2,2,8)
700 PRINT "INPUT LEVEL OF DIFFICULTY:"
710 PRINT
720 PRINT "(1) PRO"
730 PRINT
740 PRINT "(2) INTERMEDIATE"
750 PRINT
760 PRINT "(3) NOVICE"
770 PRINT
780 PRINT "(4) BEGINNER"
790 PRINT ::::::
800 CALL KEY(0,DIF,XS)
810 IF XS=0 THEN 800
820 REM   PLANE FIRING MORE
    AS DIFFICULTY INCREASES
830 DIFF=DIF-38
840 M$="0"
850 AA=1
860 ZZ=3
870 CALL CLEAR
880 REM   DEFINE CHARACTERS
890 REM    BA$=BARRIER, L$=  LASER, R$=ROCKET,
    P$ & PR$= PLANE, TL$ & TR$=TANK,
D$=  DESTRUCTION
900 BA$="FFFFFFFF"
910 D$="2A04412289045220"
920 L$="1818181818181818"
930 R$="18181818183C7E"
940 P$="603098FFFF983060"
950 PR$="060C19FFFF190C06"
960 TL$="071F7FFFFF7F3F0F"
970 TR$="E0F8FEFFFFFEFCE0"
980 CALL CHAR(120,D$)
```

continued

Anti-Aircraft Gun, continued. . .

```
990 CALL CHAR(144,L$)
1000 CALL CHAR(97,R$)
1010 CALL CHAR(112,TL$)
1020 CALL CHAR(113,TR$)
1030 CALL CHAR(101,BA$)
1040 CALL CLEAR
1050 BA=0
1060 CALL COLOR(2,16,2)
1070 CALL COLOR(16,2,2)
1080 CALL COLOR(15,12,8)
1090 CALL COLOR(13,3,3)
1100 CALL COLOR(12,7,8)
1110 T=15
1120 REM   INITIAL PRINTING   OF TANK
1130 CALL HCHAR(24,T,112)
1140 CALL HCHAR(24,T+1,42)
1150 CALL HCHAR(24,T+2,113)
1160 CALL HCHAR(23,T+1,152)
1170 CALL HCHAR(22,15,101,3)
1180 REM  PRINT GROUND(GREEN)
1190 CALL HCHAR(23,1,128,15)
1200 CALL HCHAR(23,17,128,15)
1210 CALL HCHAR(24,1,128,14)
1220 CALL HCHAR(24,18,128,14)
1230 REM  A=HEIGHT OF PLANE
1240 RANDOMIZE
1250 A=2*INT(9*RND)+1
1260 IF A=1 THEN 1240
1270 REM  DETERMINE PLANE'S  DIRECTION
1280 RANDOMIZE
1290 B=INT(2*RND)+1
1300 ON B GOTO 1310,1350
1310 B=30
1320 CALL CHAR(96,PR$)
1330 DIR=-2
1340 GOTO 1380
1350 DIR=2
1360 CALL CHAR(96,P$)
1370 B=2
1380 Y=21
1390 REM  * BEGIN LOOP *
1400 CALL KEY(0,K,S)
1410 REM  81 FIRES ROCKET,     83 AND 68 MOVE TANK
1420 IF K=81 THEN 1450
1430 IF K=83 THEN 2640
1440 IF K=68 THEN 2770 ELSE 1550
1450 IF Y=21 THEN 1480 ELSE 1470
1460 REM  FIRE ROCKET
1470 CALL VCHAR(Y+2,T+1,32)
1480 CALL VCHAR(Y,T+1,97)
1490 TT=T+1
1500 Y=Y-2
1510 IF Y=-1 THEN 1520 ELSE 1550
1520 CALL VCHAR(1,T+1,32)
1530 GOTO 1380
1540 REM  MOVE PLANE
1550 IF DIR=-2 THEN 1570
1560 IF B<3 THEN 1590
1570 CALL VCHAR(A,B-DIR,32)
1580 IF B<2 THEN 1900
1590 CALL VCHAR(A,B,96)
1600 RANDOMIZE
1610 Q=INT(15*RND)+1
1620 REM  WILL PLANE FIRE      LASER?
1630 TT=TT-2+INT(Q/2)
1640 IF B=TT THEN 1670
1650 IF Q>DIFF THEN 1670 ELSE 1820
1660 REM  PLANE WILL FIRE      LASER
1670 IF BA=5 THEN 1760 ELSE 1680
1680 IF B=16 THEN 1690 ELSE 1760
1690 BA=BA+1
1700 IF BA<>5 THEN 1720
1710 CALL HCHAR(22,15,32,3)
1720 CALL VCHAR(A+1,B,144,21-A)
1730 CALL VCHAR(A+1,B,32,21-A)
1740 CALL SOUND(500,-5,2)
1750 GOTO 1850
1760 CALL VCHAR(A+1,B,144,23-A)
1770 IF B<>T+1 THEN 1790
1780 IF A<>Y+2 THEN 2440
1790 CALL VCHAR(A+1,B,32,22-A)
1800 CALL VCHAR(23,B,128,2)
1810 CALL SOUND(500,-5,2)
1820 IF B>30 THEN 1840 ELSE 1830
1830 IF B<3 THEN 1840 ELSE 1850
1840 CALL VCHAR(A,B,32)
1850 IF A<>Y+2 THEN 1870
1860 IF B=T+1 THEN 2050
1870 B=B+DIR
1880 IF B>32 THEN 1900 ELSE 1890
1890 IF K=81 THEN 1420 ELSE 1400
1900 RANDOMIZE
1910 B=INT(2*RND)+1
1920 REM  DETERMINE PLANE'S   DIRECTION
1930 ON B GOTO 1940,1980
1940 B=30
1950 CALL CHAR(96,PR$)
1960 DIR=-2
1970 GOTO 2000
1980 DIR=2
1990 CALL CHAR(96,P$)
2000 RANDOMIZE
2010 A=2*INT(9*RND)+1
2020 IF A=1 THEN 2000
2030 GOTO 1890
2040 REM  TANK HITS PLANE
2050 CALL SOUND(1000,-5,2)
2060 REM  A TIE-TANK WINS-     REPRINT TANK
2070 CALL HCHAR(24,T+1,42)
2080 CALL HCHAR(23,T+1,152)
2090 CALL HCHAR(A,B,120)
2100 REM  DETERMINE PLANE'S   DIRECTION
2110 B=INT(2*RND)+1
2120 ON B GOTO 2130,2170
2130 B=30
2140 DIR=-2
2150 CALL CHAR(96,PR$)
2160 GOTO 2200
2170 DIR=2
2180 CALL CHAR(96,P$)
2190 REM  * SCORING *
2200 IF A>13 THEN 2210 ELSE 2230
2210 SC=SC+5
2220 GOTO 2300
2230 IF A>7 THEN 2240 ELSE 2260
2240 SC=SC+10
2250 GOTO 2300
2260 IF A<>5 THEN 2290
2270 SC=SC+15
2280 GOTO 2300
2290 SC=SC+20
2300 M$=STR$(SC)
2310 RANDOMIZE
2320 A=2*INT(9*RND)+1
2330 IF A=1 THEN 2310
2340 REM  PRINT SCORE
2350 FOR I=1 TO LEN(M$)
2360 CV=ASC(SEG$(M$,I,1))
2370 CALL HCHAR(AA,ZZ+I,CV)
2380 NEXT I
2390 GOTO 1380
2400 FOR I=1 TO 1000
2410 NEXT I
2420 GOTO 1040
2430 REM  PLANE HIT TANK
2440 CALL SOUND(500,-6,2)
2450 CALL CLEAR
2460 SX=SX+1
2470 REM  IF 2 GAMES PLAYED, START OVER
2480 IF SX<2 THEN 1040
2490 PRINT TAB(7);"   GAME OVER    "
2500 PRINT TAB(7);"YOUR SCORE IS";SC
2510 FOR I=1 TO 10
2520 PRINT
2530 NEXT I
2540 SX=0
2550 IF SC<100 THEN 2600
2560 PRINT TAB(3);"* YOU GET A FREE GAME *"
2570 REM  * A FREE GAME! *
2580 SC=0
2590 GOTO 2400
2600 SC=0
2610 REM  GO BACK TO START
2620 GOTO 680
2630 REM  MOVE TANK LEFT
2640 IF T<3 THEN 2650 ELSE 2680
2650 CALL SOUND(250,110,2)
2660 T=1
2670 GOTO 2690
2680 T=T-2
2690 CALL HCHAR(24,T,112)
2700 CALL HCHAR(24,T+1,42)
2710 CALL HCHAR(24,T+2,113)
2720 CALL HCHAR(23,T+1,152)
2730 CALL HCHAR(24,T+3,128,2)
2740 CALL HCHAR(23,T+3,128)
2750 GOTO 1550
2760 REM  MOVE TANK RIGHT
2770 IF T>27 THEN 2780 ELSE 2810
2780 CALL SOUND(250,110,2)
2790 T=29
2800 GOTO 2820
2810 T=T+2
2820 CALL HCHAR(24,T,112)
2830 CALL HCHAR(24,T+1,42)
2840 CALL HCHAR(24,T+2,113)
2850 CALL HCHAR(23,T+1,152)
2860 CALL HCHAR(24,T-2,128,2)
2870 CALL HCHAR(23,T-1,128)
2880 GOTO 1550
2890 GOTO 1040
2900 END
```



TMS9900 Machine & Assembly Language

Part 1 : Electrical Signals, Number Systems, & CPU Architecture

By Dennis Thurlow

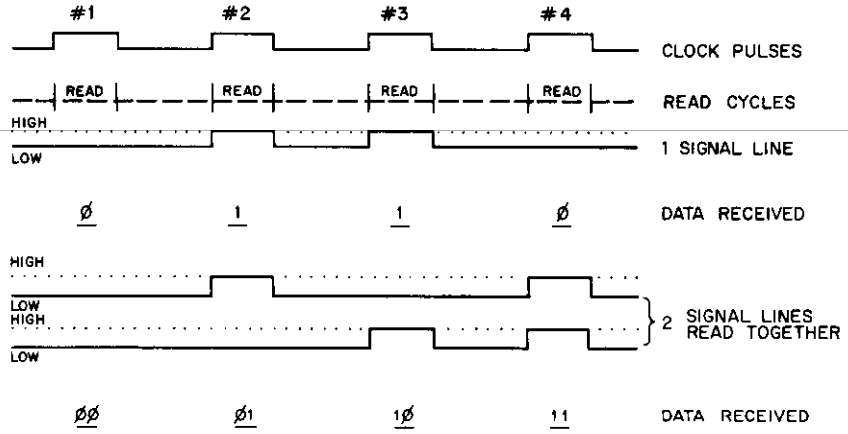


Chart 1

If you're a reader of this magazine, you are probably aware that there is a difference between 8-bit and 16-bit computers . . . although just exactly *what* that difference is—other than "16 bits are twice as many as 8 bits"—might not be that obvious. My purpose in this series of articles is, therefore, to discuss the "inner workings" of your 16-bit computer by gradually introducing you to its operation and *low-level* programming (in a language much closer to the way your computer operates *without* any BASIC interpreter slowing things down, or coming between you and the power of your machine).

The heart of any computer is its microprocessor, and the one we'll be examining is, naturally enough, the Texas Instruments TMS9900—the 16-bit chip around which *99'er Magazine* is dedicated. To understand its operation, we first have to know something about electrical signals and number systems, so let's begin our discussion here.

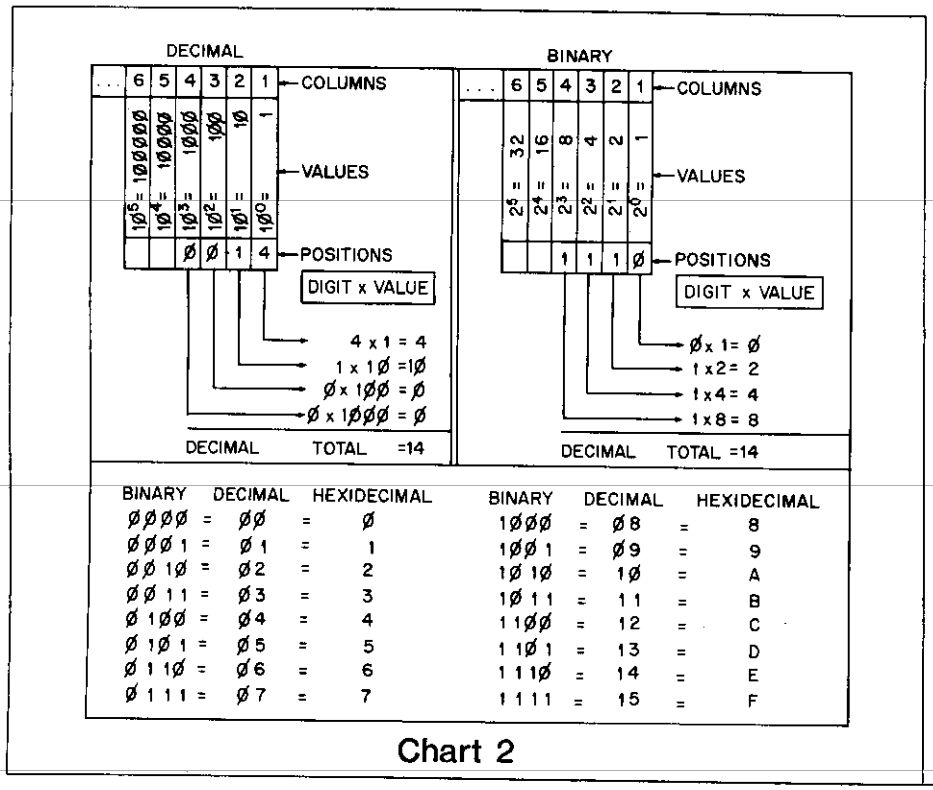
Clocks, Pulses, Bits & Bytes

The electrical signals used by a computer are labeled high and low, or 1 and 0, respectively. One of these signals is called a *bit*. Inside the computer this corresponds to one wire. All of the wires together are called a *bus*. The computer reads and writes to a part of the bus called the *data bus* at specific intervals, regulated by a *clock*. The signals that the clock produces to tell the computer when to read and write are called *clock pulses*.

At each pulse of the clock, the computer reads a group of lines. Your normal, run-of-the-mill microcomputer uses groups of 4, 8, or 16 bits. All the information read or written is called data. If the computer is reading or writing on 1 line, the data is called *serial*. If it is being read or written on a group of lines together, the data is called *parallel*. 4 bits in parallel are called a *nybble*; 8 are a *byte*; and 16 has no name, but I propose to call it a *gobbyl*.

Look at Chart 1. The top line is the clock. In this example when the pulse is high, the computer reads the signal lines. Notice that when there is only one signal line, the data received can only be a 1 (when the line is high) or a 0 (when the line is low). There are only two possible codes you could see during one clock pulse. You would see a 1 or a 0.

Now look at what happens when you have two signal lines grouped together: 4 different codes are possible. On clock pulse #1 both lines are low (code 00); on pulse #2 the bottom line is low and the top one is high (code 01); pulse #3 has the bottom high and the top low (code 10); and pulse #4 has both lines high (code 11).



BINARY	DECIMAL	HEXIDECIMAL	BINARY	DECIMAL	HEXIDECIMAL
ØØØØ	ØØ	Ø	1ØØØ	Ø8	8
ØØØ1	Ø1	1	1ØØ1	Ø9	9
ØØ1Ø	Ø2	2	1Ø1Ø	1Ø	A
ØØ11	Ø3	3	1Ø11	11	B
Ø1ØØ	Ø4	4	11ØØ	12	C
Ø1Ø1	Ø5	5	11Ø1	13	D
Ø11Ø	Ø6	6	111Ø	14	E
Ø111	Ø7	7	1111	15	F

Chart 2

Number Systems

These codes could also be considered numbers. Counting with only 0's and 1's is called *binary* (from the Latin word for two). Ordinary, plain, vanilla numbers that we use everyday are called *decimal* (from the Latin for ten, of course) or *base ten*. Even though we only have the ten digits from 0 to 9, we can make very large numbers by using the same digits in different positions. Follow along on chart 2.

The position on the extreme right in a decimal number is the one's column. For that matter, the position on the extreme right in *any* base is the one's column. Why? Because you find the column value by taking the number of digits you have and raising it to the power of column minus one. For example, if you have ten digits, and the column is number 1 (from the right) then the value of that column is 10 to the 1 minus 1, or 10 to the 0 power. Any number to the 0 power is 1, so the first column is always one's in any base.

The second column is a different matter. In base ten it is 10 to the 2 minus 1, or 10 to the 1st power, or 10. So if you write 14 what you mean is 4 groups of one's and 1 group of ten's. In base two the second column (from the right) would be 2 to the 2 minus 1, or 2 to the 1st, or 2. The second column or position in binary is the two's column.

The thing that makes the zero so neat is that it holds the position without giving it a value. Zero one's is zero! If you just left a blank there, people would have to write all their numbers in little boxes or pretty soon the columns would get all jumbled up. Is there one blank or two? . . . or three?? Better use the zero.

The columns in binary numbers are just like the signal lines in a computer. In theory, the columns go on forever —and so do the numbers. Regardless of the base you are in, you can keep writing numbers forever! But wait! I just said that signal lines are usually groups of 4 , 8 , or 16. If signal lines are the same as columns, then there is a limit to the size of number a computer can understand. How big of a number can you use?

To find out, raise the base to the same power as the number of positions you have. On chart 1 when we used two lines, that was 2 to the 2nd power, or 4 codes or numbers. With 4 lines, there are 16 (2 to the 4th); with 8 there are 256; and with 16 lines there are 65536.

The last code on the chart is 1111, which in decimal is 15. I said you could get 16 numbers with four lines, so where is the last number? Don't forget to count 0! 0 thru 15 is sixteen numbers, 0 thru 255 is 256 numbers, and so on.

There are other bases, of course. The numbers marked *hexadecimal* are from a base with 16 digits—the normal 10 digits from 0 to 9, plus the letters A to F. Use them just like any other digits. For instance, on the chart, 1111 binary is 15 decimal and F in hexadecimal (hex for short). The next number in hex is 10; in decimal it is 16; and in binary you have to add a new position (sixteens) and write 10000.

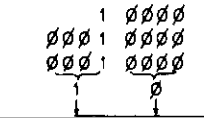
You can always add as many zeros to the front of a number as you want without changing it. However, if you make a binary number divisible into groups of four an interesting thing happens: Each group of four can represent 16 codes or numbers. Since that is exactly the number of digits in the hex number system, you can substitute! This makes long binary numbers much easier to read, and doesn't change their value at all.

Try a few yourself. They're easy!

Hardware

The TMS9900 is called a 16-bit *CPU (central processing unit)*. This means that when it "fetches" an instruction from memory, it gets 16 bits in parallel. And when it reads or writes data this is usually done in groups of 16 bits too. [In the TI-99/4, however, this 16-bit group is converted into an 8-bit data bus—Ed.] You may hear the term "word" used for 16 bits. If you are talking about a 16 bit machine, the term is correct. But remember, if you are talking about an 8-bit CPU, 8 bits (or byte) is a word; if the CPU is 32 bits, the word is 32 bits.

It is only necessary for a programmer to know about two kinds of memory. *Random-access memory (RAM)*, sometimes called *read/write memory*, is what stores the user's program, data, etc. The user or the computer can read or write in it. The memory location is chosen by the lines on the bus called *address lines*. The data that is being read or written appears on the data bus.



Binary number with 5 positions. Equal to 10 HEX.
Fill to 8 positions by adding zeros to front.
Break into groups of 4.
Give each group the proper HEX digit (see chart 2)
10 is the HEX value for 10,000 binary.

1Ø11 ØØ1Ø BINARY	1101	Ø11Ø	1Ø11	1Ø11	Ø1Ø1	1ØØØ	Ø1ØØ	Ø111	Ø110
B 2 HEX	D	6	B	B	5	C	4	7	6
1 Ø110 BINARY	VALUES					16^3 = 4Ø96	16^2 = 256	16^1 = 16	16^0 = 1
ØØØ1 Ø110	POSITIONS					4	3	2	1
1 6 HEX									

Chart 3

Read-only memory (ROM) comes in many varieties and works just like RAM except for one thing--it can't be written to. If you tell it to, the computer will go through the motions of writing, but it doesn't work. The old data is still there.

Inside of the CPU there are a few memory locations that are not addressed by the address bus. The chip itself knows where they are. These are called *registers*. All *machine language* and *assembly language* programming involves manipulating the data in these registers, because that is all that the computer really can do!

How many registers and how big they are varies widely. The chip manufacturer usually labels the registers and decides on a short code, called an *operation code*, for each of the register manipulations that the chip can do. An *assembler* is a program that reads these op-codes and writes them into memory in the binary form that the CPU understands. When you write a program using these op-codes you are writing in *assembly language*. If you write your own assembler you can devise your own op-codes. But since the manufacturer usually writes one, it is best to use theirs to avoid confusion.

About the only thing all CPUs have in common is a regis-

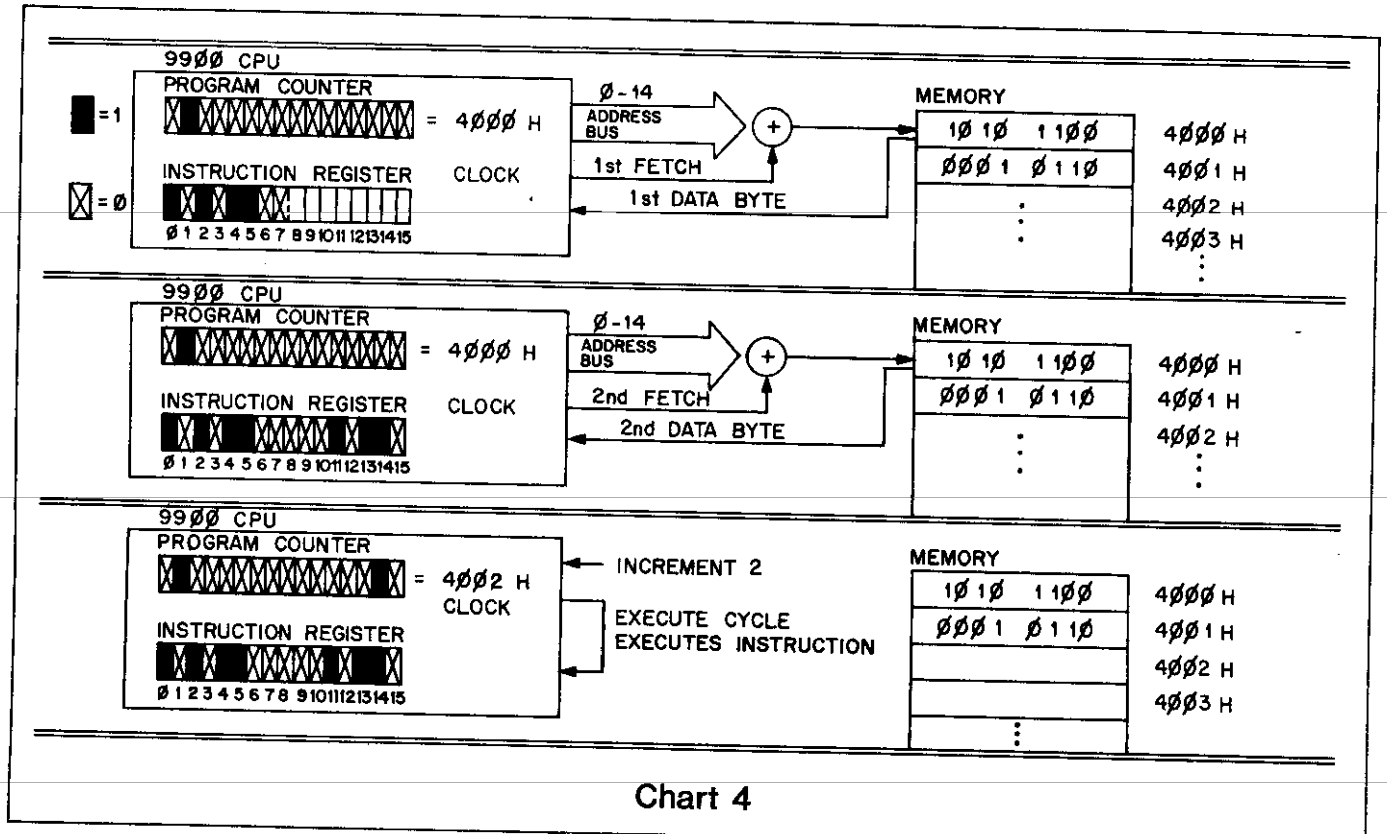


Chart 4

ter called the *program counter (PC)*. The address bus is just an extension of the PC. Each bit of the program counter is, in effect, connected to one signal line of the address bus. Since the TMS9900 chip was designed especially for dedicated control purposes (e.g., production line inspection or phone switching) where the program is always in ROM—and since at that time most ROMs were made for 8-bit computers—the address bus of the 9900 is a little unusual.

The bits of the PC allow the chip to address 65536 blocks of memory. The blocks could be any size, but as I said, most ROMs were in blocks of 8 because most computers had 8-bit data buses. The PC in the 9900 has 16 bits. These are labeled 0-15, from (left to right) *most significant bit (MSB)* to *least significant bit (LSB)*. Why are there only 15 address lines? Follow on Chart 4 as we go along.

Normally the PC increments after each instruction and/or parameter it "fetches" so that it points to the next memory byte. But the 9900 needs 16 bits instead of the 8 available at each location in most ROMs. So the 9900 has *two* different "fetch" cycles: It reads the byte indicated by the PC on the first cycle, "hooks" the next byte to it on the second cycle, then increments the PC by two. To the user this all appears as one fetch, except that the PC is incremented by *two* instead of by *one* as expected. By eliminating the last bit, however, the address line appears to step normally. The drawback is that you can only address 32767 words. It's still 65536 bytes though.

In the next issue, we will begin using assembler instructions as we examine additional registers, addressing modes, and interrupts.

What Is UCSD Pascal™ And Why Is Everybody Talking About It?

By Gary M. Kaplan

You can hardly pick up a computer publication, or attend a computer conference or fair these days without being inundated with mention of UCSD Pascal. To understand what all the fuss is about, you must first know something of what is meant by "portability," and understand the concept of "pseudocode," and its relation to the "P-machine."

Portability, Pseudocode, and P-Machines

Let's start by assuming that you already know that Pascal is a structured, high-level, compiled language (just as TI BASIC is a high-level interpreted language). In this article we won't go into the theory of compilers, interpreters, or the structure of Pascal as a computer language; we'll save this for future issues. For right now, let's imagine that your friend has written a really great Pascal compiler and operating system in his native 6502 assembly language for his Apple computer. You'd like to move it to your fully-configured TI-99/4 which has a TMS9900 microprocessor. What are your options? Sure, you could always recode the Pascal system for your 9900 (assuming you had a 9900 assembler), but it would probably be almost as much work as starting from scratch. How about first writing a 6502 simulation program for your 9900 and letting it re-write all the 6502 code? But even if you do this, the extra layer in between will result in a loss of speed and a greater memory overhead. This is what the microcomputer community has been up against—virtually *no* "portability" in moving languages or applications software from one system to another without a major re-working of the code.

Now let's design a hypothetical processor to provide a convenient "home" for Pascal. We'll give it built-in instructions for doing the type of things that the Pascal language likes to do. Let's call this pseudomachine a "P-machine" for short, and configure it to be a simple, idealized stack computer that uses "P-code"—the native language or machine code for the P-machine.

Great, but where do we go from here? What's the use of a P-machine, and how does it contribute to software portability? Must we throw all existing hardware and software out, and start over giving everyone P-machines? Obviously not. Rather, consider what would happen if we could eliminate the differences between the instruction repertoires of specific microprocessors, so that they all execute the identical P-code. Once an emulator for each CPU is written (in its native assembly language), one of the largest obstacles to portability would be overcome: Software could be written on different computers in a high-level language such as Pascal, then compiled to P-code, and finally "interpreted" for each specific CPU. Since the P-code would be universal, one program written on, say, an Apple could be theoretically run (if it consisted entirely of P-code) without modification, on a TI-99/4. Score one for portability!

This is, in effect, what has been done in the UCSD Software System. All high-level languages in this system—only one of which is Pascal—are compiled into P-code. One way of looking at it is that the *system software* is not portable at all, because it is always executed on a P-machine. The portability is provided by a P-machine emulator for each host. So when you think of a TMS9900-based system running Pascal, it is really running a simulation of a computer which is running Pascal object programs.

Speed vs Space : A Tradeoff

What price do we pay for the benefit of portability? The detour through a P-machine often produces slower execution than would native code. But raw execution speed is often overshadowed by P-code being considerably smaller than the corresponding native code—allowing the available memory to store a more capable program. And if a program can be represented with P-code that fits entirely into available memory, but using native code requires extensive overlaying, then the P-code version will actually run faster!

For best performance, it is desirable to optimize some portions of a program for space, and others for speed. Since the UCSD Pascal System provides communication between an assembly language routine and a Pascal host program, it is possible (with some reduction in portability) to code time-critical routines (usually less than 10% of a program) directly in assembly language. The low-level assembly routine can request access to host program global variables and constants, and can also allocate its own global storage space.

A project is underway at SofTech Microsystems (the firm responsible for the licensing and maintenance of the UCSD Pascal System) to alleviate many of the performance drawbacks of P-code (e.g., speed) without sacrificing portability. Code generators will translate time-critical procedures into native code through an optional step in the compilation process. A code generator will take a complete P-code program as input, and produce, as output, a mixture of unmodified P-code and translated native code procedures. Programs can then be written and maintained entirely in Pascal, with the P-code object version still completely portable. A prototype code generator for the TMS9900 demonstrated that improvement in execution performance compared to interpretative execution has been around a factor of 15! And if we take into account that translated native code for the 9900 is about 50% larger than the corresponding P-code, the improvement is indeed significant.

The Operating System

UCSD Pascal is not only a language compiler, but a complete operating system with utilities and libraries. In addition to the Compiler, you have a screen-oriented Editor and a File Manager (or "Filer"). The design philosophy behind

UCSD Pascal was to keep users continuously informed about the state of the system and the options available in that state. This is done with a prompt line that allows users to select options by typing single-character commands.

The screen orientation of the Editor means that you'll be doing lots of paging instead of scrolling. The editor positions a cursor into the text file being edited and surrounds it with a "window" into that area of the file. When you look at the display screen, you are peering into this window. To modify text you simply move the cursor to the place where the change is desired, and indicate the change. Commands are provided for cursor movement, finding and replacing patterns of text, making insertions and deletions, and copying text from elsewhere and moving it to any position indicated by the cursor. In addition to the powerful text editing commands, special facilities are provided for processing documents—e.g., user-specified left and right margins, and auto-indenting to encourage the writing of structured programs. In microcomputer systems without an 80-column display, horizontal scrolling allows users to move the text window left and right to view the entire Pascal page.

When you enter the Filer, you have access to another complete set of commands: (1) *housekeeping commands* such as listing directories, compacting files on a disk, and testing disks for bad sectors; plus (2) *program execution and file manipulation commands* for executing named object programs, invoking (with shortcuts) important system programs, designating files for removal, and for renaming or transferring among on-line devices.

The Pascal Compiler translates Pascal programs from a humanly readable text form (source code saved on disk by the Editor) into P-code form (object code) which is saved on disks for future execution. The Compiler is designed to translate the entire contents of a text file in one pass. But unlike the Editor and Filer, it has hardly any interactive commands. You can, however, change certain controls ("directives") which govern the way in which the Compiler does its work.

Error Handling

A big difference between an interpreted language (such as BASIC) and a compiled language (such as Pascal) is demonstrated in the way syntax and run-time errors are handled: If the Compiler finds a syntax error, it halts and displays an error message (if you've set it to automatically return to the Editor), or screen prints a progress display containing copies of the line (and previous line) where the program error is found, as well as the coded number of the syntax error. You can fix the error (by returning to the Editor) or attempt to compile the rest of the program. In some less drastic conditions, the program will, in fact, compile all the way to the end without the Compiler losing its way.

Run-time (execution) errors also cause all the action to stop. A three-line error message tells you the type of error, the segment and block where it occurred, and how far from the beginning of the block (which you convert to the actual line of code). In simple cases, this will be all the help that's needed to pinpoint the error; in more complex cases, you'll have to insert WRITELN statements (the equivalent of PRINT) to determine the values of variables before the program blew up. (There's no convenient BREAK statement as in TI BASIC.)

Additional Language Support

The UCSD Pascal System does, in fact, support additional compiled languages. At the present, FORTRAN-77 and BASIC Compiler are supported directly by SofTech Microsystems (Micro-Focus CIS COBOL is also presently running under the UCSD P-System).

SofTech also has a Cross-Assemblers Package (a complete set of native code generating cross assemblers for the Z80, 8080, Z8, PDP-11/LSI-11, 6502, 6800, 6809, and 9900 microprocessors) that allows programming on the host machine of your choice, for the object machine of your choice. Think of the possibilities . . .

UCSD Pascal and the TI-99/4 Community

Besides being a powerful tool for software *developers*, UCSD Pascal is of great importance to the TI-99/4 community. Texas Instruments has indicated that there is to be a user's version of Pascal in addition to the UCSD Pascal Development System. [See related article that follows.] Under this two-tier system, users will not have to buy all the software and hardware that software developers need to write and debug programs. The final configuration and price of the user's system (scheduled for release toward the end of 1981) has not yet been decided. The only detail made public at this time is that the P-code interpretation will be done by a "P-Code Box" accessory containing one type of read-only memory (an undisclosed amount).

This means that a TI-99/4 user will be able to run some very sophisticated and powerful *applications* software with only a minimal investment in the *system* hardware and software. To put this into proper perspective, it's quite possible that someone interested in only *running* (and not writing) UCSD Pascal applications programs will have to spend about *three times (!)* the cost of a minimum TI-99/4 user-Pascal system if he instead chooses UCSD Pascal-compatible hardware from another microcomputer manufacturer. This is quite a premium to pay if a user won't be *writing* his own software. Needless to say, this could turn out to be quite a marketing coup for Texas Instruments.

UCSD Pascal is a Trademark of the Regents of the University of California.

Invasion from Space, continued from p. 45

```
3270 GOTO 3410
3280 PRINT "YOU HAVE DEFEATED THE ALIENS"
3290 PRINT "AND SAVED THE EARTH FROM"
3300 PRINT "ETERNAL SLAVERY"
3310 CALL SOUND(200,440,2)
3320 CALL SOUND(300,740,2)
3330 CALL SOUND(500,1047,2)
3340 PRINT "YOUR SCORE IS:";POINTS
3350 GOTO 4570
3360 NEXT X3
3370 NEXT X2
3380 IF M2<30 THEN 3400
3390 M2=1
3400 RETURN
3410 PRINT ::
3420 PRINT "DO YOU WISH TO PLAY AGAIN"
3430 PRINT
3440 PRINT "IF SO TYPE ""Y"" TO CONTINUE"
3450 PRINT
3460 PRINT "TYPE ""N"" TO STOP"
3470 CALL KEY(0,K,ST)
3480 IF ST=0 THEN 3470
3490 IF K=78 THEN 4570
3500 IF K=89 THEN 3540
3510 GOTO 3470
3520 REM    SET UP VARIABLES
3530 REM    FOR NEXT GAME
3540 FOR X=1 TO 30
3550 M(1,X)=0
3560 M(3,X)=0
3570 M(5,X)=0
3580 POINTS=0
3590 NEXT X
3600 M1=0
3610 M2=0
3620 GOTO 130
3630 REM  DISPLAY SCORE
3640 PO=POINTS
3650 IF POINTS>=0 THEN 3680
3660 POINTS=POINTS*(-1)
3670 CALL VCHAR(1,9,45)
3680 SC1=INT(POINTS/1000)
3690 SC2=INT(POINTS/100)-SC1*10
3700 SC3=INT(POINTS/10)-((SC1*100)+(SC2*10))
3710 SC4=POINTS-((SC1*1000)+(SC2*100)+(SC3*10))
3720 IF PO<0 THEN 3740
3730 CALL VCHAR(1,9,ASC(STR$(SC1)))
3740 CALL VCHAR(1,10,ASC(STR$(SC2)))
3750 CALL VCHAR(1,11,ASC(STR$(SC3)))
3760 CALL VCHAR(1,12,ASC(STR$(SC4)))
3770 POINTS=PO
3780 RETURN
3790 REM    MOVE OPERATORS
3800 REM    MISSLE
3810 Z1=P2+Z2
3820 CALL GCHAR(23,P2+Z4,CH)
3830 IF CH=32 THEN 3880
3840 CALL VCHAR(23,Z1+Z4,32)
3850 Z=23
3860 Z1=P2+Z4
3870 GOTO 4010
3880 CALL VCHAR(23,P2+Z4,Z3)
3890 FOR Z=22 TO 2 STEP -1
3900 CALL GCHAR(Z,Z1,CH)
3910 CALL VCHAR(Z+1,Z1-Z4,32)
3920 CALL VCHAR(Z,Z1,Z3)
3930 IF CH>32 THEN 4010
3940 Z1=Z1+Z4
3950 IF Z1<3 THEN 4120
3960 IF Z1>30 THEN 4120
3970 CALL SOUND(20,1480,2)
3980 NEXT Z
3990 CALL VCHAR(Z+1,Z1-Z4,32)
4000 GOTO 4130
4010 CALL VCHAR(Z,Z1,32)
4020 CALL SOUND(200,110,2,-6,2)
4030 REM    CHECK FOR TYPE OF
4040 REM    HIT BY OPERATOR
4050 IF CH=104 THEN 1820
4060 IF CH=105 THEN 1900
4070 IF CH=106 THEN 1980
4080 IF CH=107 THEN 4330
4090 IF CH=108 THEN 4400
4100 IF CH=109 THEN 4470
4110 IF CH=112 THEN 2080
4120 CALL VCHAR(Z,Z1-Z4,32)
4130 RETURN
4140 REM    SET UP VARIABLES
4150 REM    TO FIRE OPERATOR
4160 REM    MISSLES IN ONE OF
4170 REM    THREE DIRECTIONS
4180 Z2=-2
4190 Z3=121
4200 Z4=-1
4210 GOTO 3810
4220 Z2=2
4230 Z3=122
4240 Z4=1
4250 GOTO 3810
4260 Z2=0
4270 Z3=120
4280 Z4=0
4290 GOTO 3810
4300 REM    ENEMY MISSLES
4310 REM    DESTROYED BY
4320 REM    THE OPERATOR
4330 FOR J=1 TO 30
4340 IF M(1,J)<>Z THEN 4390
4350 IF M(2,J)<>Z1 THEN 4390
4360 M(1,J)=0
4370 POINTS=POINTS+5
4380 RETURN
4390 NEXT J
4400 FOR J=1 TO 30
4410 IF M(3,J)<>Z THEN 4460
4420 IF M(4,J)<>Z1 THEN 4460
4430 M(3,J)=0
4440 POINTS=POINTS+5
4450 RETURN
4460 NEXT J
4470 FOR J=1 TO 30
4480 IF M(5,J)<>Z THEN 4530
4490 IF M(6,J)<>Z1 THEN 4530
4500 M(5,J)=0
4510 POINTS=POINTS+5
4520 RETURN
4530 NEXT J
4540 RETURN
4550 REM    END OF PROGRAM
4560 REM    MESSAGE
4570 PRINT "YOU COULDN'T FIGHT YOUR"
4580 PRINT "WAY OUT OF A PAPER BAG..."
4590 END
```

THIRD-PARTY SOFTWARE DEVELOPMENT SYSTEMS

By Gary M. Kaplan

Did you miss out on buying gold at $35 an ounce . . . or ignore those first few McDonald's franchises that "nobody seemed to want" . . . or perhaps, throw out your old comic book collection (dating back to the first Superman) because you needed the space? Well—now's your chance to get even. The software business is booming and the decade of the 80s will provide unprecedented opportunities for those with enough foresight to get in on the ground floor.

The microcomputer industry is very young. If as little as *four* years ago you started tinkering with hardware and software to sell, you're already considered to be an "oldtimer!" But four years, in a dynamic industry such as this, is a long time. Much of what has occurred over this period was *not* expected to happen so quickly. What the industry will be like four years hence—when consumers start buying microcomputers like they're presently buying microwave ovens—is anybody's guess. One thing, however, is certain: *Several million microcomputer owners are going to need software.*

Where will all this software originate from? Who will be supplying it to the end users? What machines will be the most popular with these non-hobbyist, applications-oriented consumers? Software developers with an eye on this eventual mass market can't help but notice how the present hardware from Texas Instruments has been "human engineered" to appeal to this group —much more so than the present breed of popular *hobbyist* computers. Supporting this TI product line would therefore seem to be as good a bet as any in the marketing scramble ahead . . .

With this in mind, let's take a brief look at what Texas Instruments is presently offering to third-party software developers. The first question that comes to any developer's mind concerns programming languages. Which ones are presently available for users to run, and which will be forthcoming? There's no need to discuss TI BASIC here; it's been around since the inception of the TI-99/4, and has been the only language that the *small* software developer could utilize. I am predicting, however, that more developers will turn to the new Extended BASIC because of its greater capabilities and protection feature. An explanation of Extended BASIC in relation to software development is covered in a separate article in this issue.

UCSD Pascal™ Software Development System

This brings us to the UCSD Pascal™ Development System, which, as we go to press, is being released with UCSD Pascal Version 4.0 (as upgraded from its predecessor Version 2.0 by SofTech Microsystems, the San Diego firm responsible for maintaining and licensing the UCSD software). As presently sold by Texas Instruments, the System allows you to program in UCSD Pascal and TMS9900 assembly language. With this system, you have the capability to access hardware through assembly subprograms, and take advantage of all machine capabilities such as sprites, sound, and speech. Developers who program in UCSD Pascal (the UCSD Pascal System includes UCSD Pascal as *one* of its languages) gain the benefit of portability: applications programs that can run on many different computers (not just TI-99/4s) without modification. [See related UCSD Pascal article in this issue.] And because the development costs of these programs may be quickly recouped (by selling potentially 50,000 to 100,000 copies), it becomes unnecessary to sacrifice features or reliability to hold down development time and cost.

The System that Texas Instruments is now making available is in a "hybrid" form: A "Super Memory Box" contains both the necessary expansion RAM for editing and compiling Pascal, and the extra GRAM (Graphic Random-Access Memory) for the P-Code interpreter (after downloading from disk). The final implementation (available in third-quarter 1981) will use the expansion 32K RAM peripheral and a separate "P-code Box" (with its own resident interpreter). The price is higher, however, for the presently available hardware and software (approximately $1,050 for the "Super Memory Box" and $900 for the software; vs. $400 for the 32K RAM peripheral, $600 for the software, and as yet, an undetermined amount for the "P-Code Box"). In addition to the UCSD software, TI provides both implementations with a run-time package of machine-specific utilities. [For a brief discussion of TI's forthcoming *end-user* Pascal system, see the last section in this issue's "What Is UCSD Pascal . . . "]

For software developers interested in programming in TMS9900 assembly language, there are some additional considerations. The UCSD Pascal Development System includes a text editor, macro assembler, and link-editor. During assembly a set of macros facilitates accessing specific TI-99/4 hardware features (e.g., the VDP or sound chip). An I/O utility is available which may be linked with an assembly routine to provide access to peripherals (e.g., disk drives, RS-232). Since assembly language programs must be called up through Extended BASIC on a user's system, there must be some way to convert assembly language object files produced on the UCSD Pascal Development System (employing a special disk format) to the format required by Extended BASIC. TI, therefore, provides a utility to do this conversion. [See related article on Extended BASIC in this issue.]

A Separate Assembler

If your main interest is assembly language programming and you don't feel like making the investment in the UCSD Pascal software and hardware, there will soon be another option. Texas Instruments has announced that a TMS9900 assembler will soon be forthcoming. No additional details have been released. Whether it is only a simple line-by-line assembler, or a more powerful package (complete with text editor, etc.) as in the UCSD System remains to be seen. Nevertheless, the fact that TI is releasing a separate assembler (CALLable through Extended BASIC) is a symbolic (no pun intended) gesture of great significance: It demonstrates a shift in policy

Power - Line Problems In Personal Computers

By G. R. Michaels

Although glitches, crashes, errors, false printouts, memory loss, and other forms of erratic microcomputer operation are usually blamed on software and hardware, most of these annoying problems actually come to you courtesy of your ordinary 120 Volt powerline! These problems are directly traceable to three general causes: (1) processor-memory-peripheral interaction, (2) power line noise/hash, and (3) transient voltage surges. Fortunately, serious computer users don't have to live with these problems, since many types of corrective devices are available.

Powerline Coupling

The fact that microcomputer systems are so easy to hook-up—just plug the computer and peripherals into the wall socket, and connect the components with a few convenient male/female preassembled cables—makes them susceptible to powerline coupling and the device interaction that results. Multiple-socket power strips that are integrated with RFI (radio-frequency interference) filters will effectively isolate the computer and peripherals from each other and from the power line—thus providing a convenient solution to the problem.

Hash

Hash is another problem altogether. When your favorite space-war game gets fouled up by "glitches," or your previously-proven-to-be-faultless program "blows up" or creates erroneous printout, externally created hash is the probable cause. Elimination of hash at the source is the most desirable solution. But with hundreds of potential sources (arcing in tools, motors, appliances, and other small electrical devices, plus loose, defective, or corroded light sockets, wall sockets, line-cord plugs, or wire connections), pinpointing the offender is often difficult. That's when hash filters are most effective. They often can completely eliminate the interference.

An alternate approach to the hash problem is first to make certain that all equipment covers and shields supplied by the manufacturer are securely fastened in place. If that doesn't work, you might try building and installing your own shield. Also don't forget to make sure that you have an adequate grounding system with *direct* ties to a good ground rather than *ground loops* (which often provide a home for system hum that can induce glitches).

Transient Voltage Surges

Transient voltage surges (transients) are certainly not friends of microcomputer circuitry. Semiconductor components are easily damaged by these momentary spikes—often 5 or 10 times the normal AC line voltage. And IEEE studies indicate that some transients have pulses up to 5,600 volts!

Common causes of destructive powerline transients include (1) demand power load switching by utility companies, (2) nearby lightning strikes, (3) static discharge, and (4) on/off switching of inductive motors, power supplies, air conditioning and refrigeration units. Any of these can cause a Differential Mode powerline surge—one in which short surges of extremely high voltage are developed *between* the AC lines. Anything connected to the AC lines will get a dose of this damaging voltage. The resulting "domino effect" could wipe out large sections of memory of a microcomputer.

A Common Mode surge occurs when *both* AC lines are brought to a very high voltage—a situation usually caused only by lightning. This high voltage may cause arcing between conductors and ground, destroying the insulation of power transformers (rendering the units worthless) and cables. Damage to switches and controls is also a frequent occurrence in this situation.

Besides the surge damages that are immediate and permanent, there are some harder-to-detect damages as well: performance characteristics and shortened life-spans. These damages can be the most irritating since equipment will require repeated servicing and will often seem to be falling apart.

Fortunately, a large measure of surge protection is possible with clamping devices that can be placed across the AC line and between each AC line and ground. These devices are frequently built into special AC line cords, and thus, like the other protective devices mentioned, can be attached without altering any equipment.

Third-Party, continued

and signifies a new willingness to allow outsiders to take fuller advantage of the powerful hardware residing the TI-99/4 product line.

Debugging Assembly Language

Software development in assembly language usually involves a great deal of debugging. To aid in this time-consuming process, most programmers depend heavily on debug utilities resident in monitors. So if you are using the UCSD Pascal Development System, Texas Instruments will presently modify your TI-99/4 console and RS-232 interface (for approximately $250) to provide the necessary hardware and software traps to send the information to an external CRT or printer terminal. (The ten commands and eight utilities of the TIBUG monitor are then at your disposal.) TI has also indicated that in third-quarter 1981, it will sell developers the modified console and RS-232 at the regular retail prices. The plans for debugging support of the separate 9900 assembler haven't as yet been disclosed.

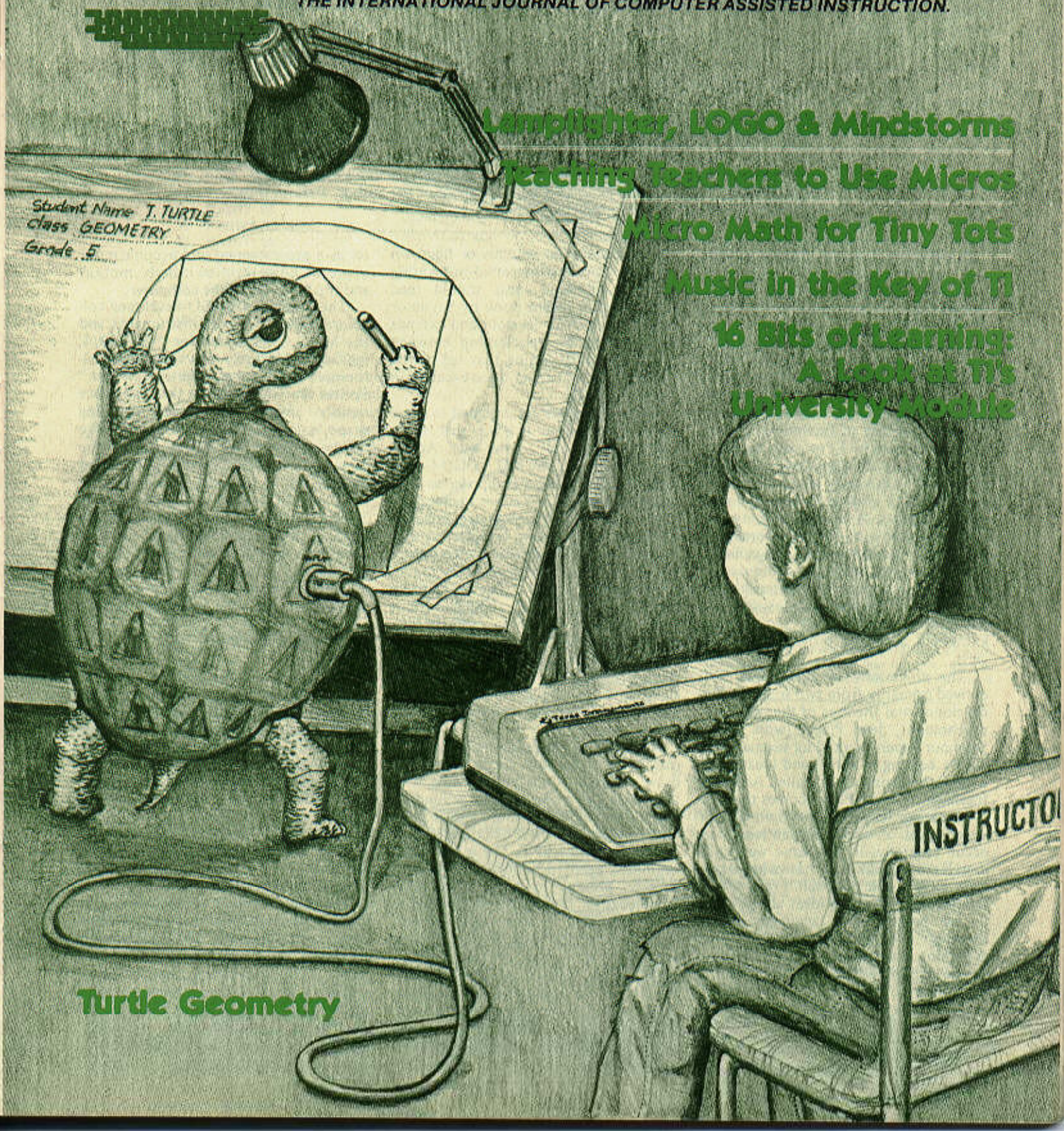
In Part 2 of this series on *Third-Party Software Development Systems* (in the July/August issue), we will look at GPL as a development language, consider the options in bringing up other languages to run under the UCSD Pascal Software Development System, and examine the technical and marketing considerations in the choice of programming media —cassettes, diskettes, and Command Modules. And in related articles in forthcoming issues, we'll explore other systems for programming and producing firmware in 9900 assembly language.

May/June 1981 Vol. 1, No. 1

$2.00 in USA

ON LOCATION™

THE INTERNATIONAL JOURNAL OF COMPUTER ASSISTED INSTRUCTION.



Lamplighter, LOGO & Mindstorms

Teaching Teachers to Use Micros

Micro Math for Tiny Tots

Music in the Key of TI

16 Bits of Learning: A Look at TI's University Module

Turtle Geometry

LOGO:

A Computer Language as a Learning Environment

By Daniel H. Watt Ph.D.

For many years the work of the MIT LOGO Group has been supported primarily by the National Science Foundation, and secondarily by a number of other public and private agencies. The Texas Instruments company has provided direct support for the technical development of TI LOGO. The author wishes to acknowledge the support of these agencies, and of the Public Schools of Brookline, Massachusetts. The opinions expressed here are those of the author, and not the responsibility of any of the supporting agencies.

LOGO—a powerful, high-level computer language designed for educational purposes especially as a programming language suitable for young children—will soon be available on Texas Instruments TI-99/4 Home Computers. For more than a dozen years, the LOGO language and related computer programming activities have been developed and tested by the LOGO Group at the Massachusetts Institute of Technology. Under the leadership of MIT Professor Seymour Papert, LOGO activities have been used with children as young as nursery school age, with MIT undergraduates, and with many students of all ages in between. The philosophy of LOGO's developers has been: "No threshhold, no ceiling." A beginner can make the computer do something meaningful and interesting in the very first programming session. Yet, the other extreme, LOGO is suitable for extremely advanced programming projects.

The philosophy of LOGO has been derived primarily from two sources: The developmental theories of the late Swiss psychologist, Jean Piaget (with whom Seymour Papert worked for several years before coming to MIT), and ideas from a modern scientific field called Artificial Intelligence. From Piaget comes the idea of creating learning environments in which most of what children learn can occur naturally—in the same way children learn to speak their native language, walk or run, and play ball. From Artificial Intelligence come ideas about ways to use programming languages to aid thinking and problem solving. Programming a computer in LOGO is seen as the act of teaching the computer a set of new commands, based on what it already knows how to do. Each user is, in effect, creating his or her own computer language, to suit his or her own purposes. Readers interested in learning more about these ideas should read *Mindstorms*, a recent book by Seymour Papert, in which he develops and extends his vision of the relationship between computers and learning that led to the development of LOGO. [See excerpts in this issue—Ed.]

LOGO activities are designed to allow use of the computer in a way that is personally meaningful to the user. Activities developed by the MIT LOGO Group have included using a computer to control the behavior of a robot turtle, draw pictures and explore geometric environments on a TV screen, create computer animations, invent interactive computer games, compose, arrange and play music, and produce "poetry." The best known LOGO activity is using a simulated robot turtle on a TV screen to produce geometric designs and cartoon-like drawings. Hundreds of children have learned computer programming and problem solving skills, and developed mathematical expertise while writing programs for the turtle.

The LOGO language includes commands to make the turtle move and draw pictures. A student drawing with the turtle can make it move around on the TV screen, by typing familiar commands such as FORWARD and BACK or RIGHT and LEFT. The information beginners need to control the turtle is already present in their own body knowledge of how to move forward or back, and how to turn right and left. Programming becomes an extension of something a learner already knows—rather than something requiring the mastery of an elaborate technical language or a complex coordinate system. The turtle becomes for the learner, what Seymour Papert has called "an object to think with." Students using the computer as a programming tool become more aware of both their own body motion and the behavior of the computer.

The version of LOGO developed collaboratively by Texas Instruments and the MIT LOGO Group for the TI-99/4, includes an entirely new graphics environment called a "Sprites World." Sprites are small objects that can move rapidly around the screen changing shape, color, speed and direction. Large numbers of sprites can appear at the same time to produce exciting animated designs, or to be used as elements in programs to create video games. The Sprites World promises to be one of the most exciting computer based learning environments yet invented, because of its inherent attraction for so many people and because of the geometric and problem solving ideas embedded in it.

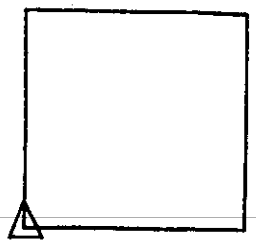
The World of the Turtle

Let's take a closer look at what actually happens when someone learns to program a computer using the LOGO turtle. The turtle responds to simple commands typed at the keyboard: FORWARD 100, BACK 50, RIGHT 90, LEFT 45, etc. FORWARD 100 moves the turtle forward "100 turtle steps," drawing a line on the TV screen in the process. LEFT 45 makes the turtle rotate 45 degrees to its own left. People learning LOGO find it natural to "identify" with the turtle, imagining themselves going through its motions, as it carries out a particular task. At the same time, controlling the turtle becomes a metaphor for controlling the computer itself: Like the turtle, the computer responds to an ordered series of commands, and to procedures that are defined as series of commands.

The ways in which the actions of the turtle can lead to geometric designs, as

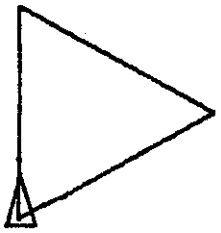
Technical Education Research Centers
8 Eliot Street, Cambridge, MA 02138

well as the method used to define procedures is illustrated in the following simple examples. The turtle can draw a square by repeating the commands FORWARD 100 RIGHT 90 four times. A *procedure* can be defined by choosing a name (BOX, for example) and typing a series of commands in order.



```
TO BOX
10 FORWARD 100
20 RIGHT 90
30 FORWARD 100
40 RIGHT 90
50 FORWARD 100
60 RIGHT 90
70 FORWARD 100
80 RIGHT 90
END
```

When the new command, BOX, is typed, the turtle immediately draws the shape shown in the figure. (The small triangle shown in the figure represents the turtle by showing its position and heading.) A similar procedure, TRI, can be defined as follows:

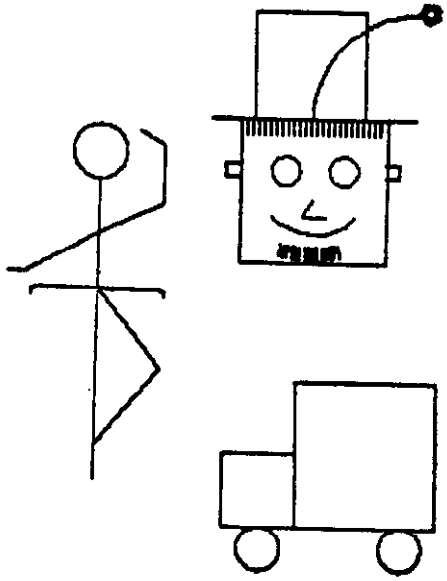


```
TO TRI
10 FORWARD 100
20 RIGHT 120
30 FORWARD 100
40 RIGHT 120
50 FORWARD 100
60 RIGHT 120
END
```

A student who has defined procedures such as BOX and TRI is beginning to "teach the computer" his or her own private language. BOX and TRI can now be used in the same way as other LOGO commands. They can be used to create other drawings such as a simple "house" or an abstract geometric "flower."

This approach to geometry and programming provides the basis for a rich universe of activities known as Turtle Geometry, which includes cartoon drawings, simple and complex geometric designs, mathematical theory building, and computer games. Extensions of Turtle Geometry have proven fruitful when used with advanced high school students or MIT undergraduates. The universe of Turtle Geometry provides a conceptual framework for such aspects of mathematics as the relation between shapes and angles, coordinate systems, positive and negative numbers, the use of variables, symmetry and similarity, and even calculus and differential geometry. The computer programming involved in beginning LOGO activities can include procedures and subprocedures, the naming of procedures and variables, procedural hierarchy, recursion and iteration, the use of conditional logic, and the development of problem solving strategies.

Within the universe of Turtle Geometry, there is room for different students working individually to create their own sub-universes or *microworlds*. They can do this with their own limited (but expandable) sets of concepts and related activities and projects. To teach LOGO is really to help learners create, explore, and extend their own microworlds.



I have used turtle geometry as an *example* of what can be done with LOGO because it is easy for a reader to visualize the commands and to see how they lead to procedures that produce the results in the pictures—just as it is for young children. Children learning LOGO have actually carried out many other types of projects as well: moving turtles, finding their way around racetracks or mazes, animated cartoons, interactive computer games such as Nim and Tic-Tac-Toe, programs which generate sentences or poetry (or even play Mad-Libs), and programs to translate English into Morse Code, or vice-versa. As LOGO becomes available to owners of TI-99/4 computers, I hope that these pages can be a forum for describing *your* LOGO projects. Since there will soon be more LOGO users than ever before, we can expect more and different LOGO projects to emerge. One of the best ways to build the culture of LOGO, is for users to share project ideas through the pages of magazines such as this.

Although TI LOGO is a recent entry to the LOGO World, a prototype version has already been tested with hundreds of students between the ages of three and nine at the Lamplighter School in Dallas, Texas, and by students in fifteen elementary and junior high schools in New York City. Using the Sprites World of animated graphics activities, these students are busily creating a new universe of LOGO activities, to delight and educate a new generation of computer users. In an age in which computers are omnipresent in society, and in which universal computer literacy is a pressing national need, computer-based learning environments like LOGO have become essential to the process of growing up literate in the last decades of the twentieth century.

Dr. Watt will be leading several LOGO workshops during the next few months. Contact him at Technical Education Research Centers.

References

Abelson, H. and A. diSessa. *Turtle Geometry*, MIT Press, Cambridge, Massachusetts, 1981

Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, New York, 1980.

Papert, S. diSessa, A., Watt, D., and S. Weir. "Final Report of the Brookline LOGO Project" Volumes II and III. MIT LOGO Memos 53 and 54, Cambridge, Massachusetts, 1979.

Papert, S. "Teaching Children to BE Mathematicians vs. Teaching About Mathematics," MIT LOGO Memo 4, Cambridge, Massachusetts, 1971.

Papert, S. and C. Solomon. "Twenty Things to do With a Computer," LOGO Memo 3, Cambridge, Massachusetts, 1971.

Watt, D. "A Comparison of the Problem Solving Styles of Two Children Learning LOGO," Proceedings, National Educational Computing Conference, 1979. Reprinted, *Creative Computing*, December, 1979.

Watt, D. "Computer Literacy: What Schools Should be Doing About It," *Classroom Computer News*, Volume 1, Number 2, November/December 1980.

Watt, D. and S. Weir. "LOGO: A Computer Environment for Learning Disabled Students," To be published in *The Computing Teacher*, Volume 8, Number 5, Spring, 1981.

The Lamplighter LOGO Project

By Henry Gorman Jr.

Department of Psychology, Austin College, Box 1584, Sherman, TX 75090

"A child is not a vessel to be filled, but a lamp to be lighted." The quote from Alexandrov is on the plaque outside the Lamplighter school. That sign advises any visitor that the school is very unusual.

The curriculum at Lamplighter is individually tailored to meet the needs of each student. Individualization is applied in science, language arts, math, drama, music, art, French, and physical education. The Lamplighter is strongly supported by the parents of its students and by its alumni, with graduates of Lamplighter frequently dropping by to see their former teachers. Such alumni loyalty might not be considered unusual, except that Lamplighter classes begin with preschool (age 3) and end with fourth grade-level!

The physical arrangement of the school reinforces its approach to learning. Classrooms have only three walls; the fourth side of each class opens onto an airy, bright shared-space. Classrooms are clustered around these shared-spaces by grade-level. Inside each classroom there are tables and chairs for writing work and, on one side, a small tiered well which is used for many other activities (e.g., reading, French, music, or story telling). The staff, the facilities, the students, and the parents all contribute to make Lamplighter a very special private school.

Lamplighter has been a leader in the use of new technology for learning. Calculators, Speak & Spells, Systems 80 units, and Little Professors are abundant throughout the school. Students regularly use these learning tools and other learning games found in the shared-spaces. Teachers make extensive use of slides, films, video and audio tapes. When Mr. Erik Jonsson (co-founder of Texas Instruments and Lamplighter Board of Directors Chairman and benefactor) first proposed introducing computers into Lamplighter, his idea was well received. Mr. Jonsson had earlier been in contact with Dr. Seymour Papert of the Division for Study and Research in Education (DSRE) of MIT, and found the LOGO language and philosophy of learning intriguing. Papert's initial explanation that LOGO allowed students to program computers and not vice-versa, enjoyed a favorable reception from the Lamplighter faculty. Later as Papert elaborated on the LOGO philosophy, it became clear that LOGO was very much in accord with the practice and philosophy of Lamplighter.

In the fall of 1978, Papert and several others from DSRE made a series of preparatory visits to Lamplighter to arrange for the introduction of LOGO to the school. The plan was to begin LOGO training for first the faculty, and then the students by using the Digital LSI-11 LOGO (in use at the Brookline Massachusetts project) and later, bring TI LOGO into the school as it developed.

Shortly after the first visit by Papert, Lamplighter rented the first of two LSI-11's that were to be used in the initial two years of the project. Training sessions helped the initial core of Lamplighter faculty (representing nursery school, second grade, third grade, and fourth grade) become facile with LOGO. This "Computer Group" then began working with third and fourth grade students. Shortly thereafter, a second LSI-11 was rented, and by the end of the spring term every third and fourth grade student had had at least one hour of LOGO instruction on a computer.

The third and fourth graders considered it a treat to work on the computer —partly because special activities allowed them to miss classes, and partly because they genuinely enjoyed LOGO. One student's remark reflects the sentiments of many of these pupils. After he had spent an hour figuring line lengths, turn angles, and sections of arcs in order to construct a computer picture of a cat, he thanked me for "getting out of math class."

In the summer of 1979, the computer Group was expanded, and two workshops were held to freshen the teachers' memories. Subsequently, a 10 day workshop at MIT introduced the teachers to more elaborate LOGO programming and allowed them to participate in discussions on the relationships between learning and LOGO. Then, as the new school year started, the teachers were really surprised to discover how

little the fourth graders had forgotten about LOGO. These students generally recalled all of the commands they had learned three months earlier—even though they had had no contact with LOGO in the interim!

Midway through the fall semester of 1979, several early prototypes of TI LOGO were tested at Lamplighter and revised by the MIT LOGO laboratory personnel in consultation with Lamplighter and Texas Instruments. In January 1980, the pace of computing at Lamplighter accelerated as an updated version of TI LOGO was implemented on the TI prototypes. By the end of January, a dozen prototypes were in use at Lamplighter, and very few students continued to use the LSI-11 LOGO. Most pupils, in fact, switched to the TI prototypes even though that meant relearning much of LOGO.

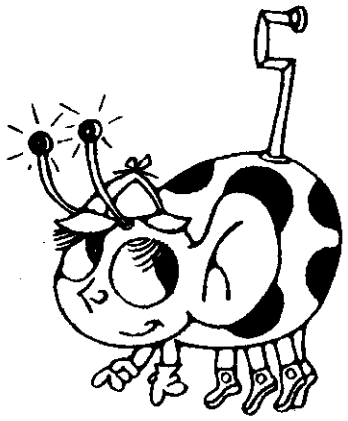
In the middle of the spring semester, a few more prototypes arrived and all the machines were upgraded to a later version of TI-based LOGO. Before the school year ended, all of the third and fourth graders had had at least one hour on the new machines. One of the rented LSI-11's was then returned (though few noticed its departure). At that time, several fourth graders were writing elaborate programs which made use of recursion to create "movies" or "rainbows" (changing colors), or elaborate scenes. Some students were so taken with LOGO that their parents happily bought them their own computers (at that time, TI LOGO was not yet commercially available); other students became so enthralled with their ability to produce perfectly printed letters and numerals on a keyboard, and later received typewriters as presents from their parents.

By September 1980, a total of 50 TI LOGO prototypes were in operation at Lamplighter. The version of LOGO on these units was very close to that which TI will market. Then, late in the fall, the second LSI-11 was returned, but its loss went *completely* unnoticed because all of the faculty and student interest was already focused upon the TI LOGO prototypes. Since September, the Computer Group has continued to work individually with third grade students. In addition, the other faculty are being trained in LOGO, and it has been introduced into all of the classes as part of the regular school curriculum.

The teams at each grade level decided the best way to introduce LOGO into their classes, and worked out various procedures for that introduction. For example, one teacher developed special simplified LOGO programs for the preschool children which required less typing in order to produce interesting effects. And personifications of LOGO constructs made LOGO easier for first and second graders to understand. Currently, students can be seen at every shared-space LOGO machine during lunch-hour, before school, after school, and whenever other school activities are completed. For the rest of the semester, LOGO will be used in class by the teachers as they feel it is relevant for their lessons, and will continue to be available (as are the other learning aids) to students during free periods.

The Lamplighter LOGO project was not intended to be a formal experiment. Since there are no control groups, strong causal claims for LOGO's effects are inappropriate. Several cognitive and psychological assessments, however, were made at the beginning of the project, and will be made again at the conclusion of the present school year. Nevertheless, there already have been some indications of student attitude and behavior change. This is best exemplified by the way in which the pupils express their keen regard and interest in acquiring new LOGO knowledge.

It's always interesting to observe what motivates children to learn. Because there are so many things which TI LOGO can do, it is not possible for the Lamplighter teachers to show students most commands during the initial sessions. The students have taken this as a challenge and approach the discovery of more LOGO commands as a treasure hunt. Whenever "unauthorized" LOGO information is found, it is disseminated through an "underground network" among the students. The teachers first became aware of this network late last year. When a few of the teachers were being instructed in how to use Makeshape (the LOGO mode by which users may make their own shapes in a 16x16 grid) they were apparently being secretly watched by one or more students. Shortly afterward, a hand-copied "underground" LOGO manual was found on the floor of a classroom; the purloined note concisely and accurately gave directions for use of Makeshape. And at about the same time, a number of students actually began using Makeshape.

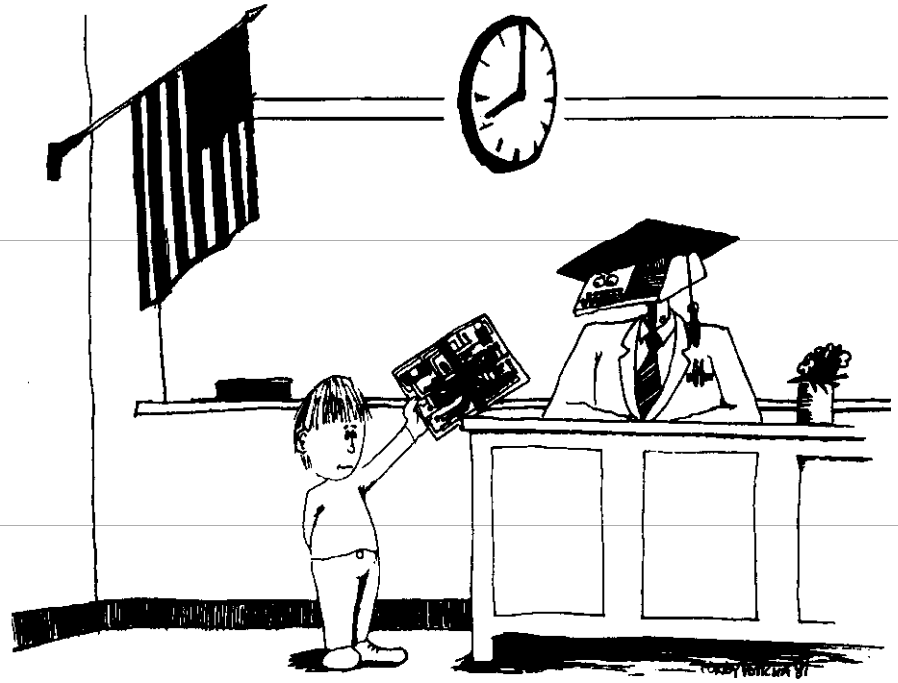


Other information has been uncovered by students accidentally. One student was trying to type M$ for Makeshape but typed MC by mistake. This put him into Makecharacter mode (the mode in which LOGO users can modify existing characters or make their own). That student proudly shouted his discovery to his classmates, who quickly confirmed his results and spread the news to still more students. Students in the lower grades also pick up some additional LOGO knowledge from their classmates with older brothers and sisters in the fourth grade. And occasionally, third graders see some of the fourth graders' inventions when they walk through fourth grade shared-space, and they in turn attempt (often successfully) to duplicate those effects. Turtle geometry, "movies" and "flash the screen" reached the third grade that way. Variations of the "explosion" program (in explosion, as the students call it, 32 colored shapes move from the center of the screen to splay out into a circle and then move back to the center in a tail recursive program), appeared one day in the third grade. These variants were traced back to a student whose father worked in the TI LOGO office.

This pattern of sharing LOGO among peers is the overwhelming response of Lamplighter students. Pupils eagerly and proundly explain how they accomplished something in LOGO to other pupils and even to teachers. However, there were—especially at first—a few exceptions. A couple of students were secretive about what they knew in LOGO and how they achieved some LOGO effects. One such student went so far as to make the screen's background color black so that no one could read the instructions he typed! Another student began collecting interesting LOGO programs which he tried to *sell* to his classmates! It would, however, be a distortion to present these two students as in any way typical of the Lamplighter pupils. The student who had hidden his typing on a black background soon discovered that others often found different ways to write programs that produced similar effects, and were willing to share. As a result of this discovery he is now also sharing *his* programs.

In at least one case, LOGO seems to be responsible for a *major* behavioral change. Late last year, a fourth grader who had not been performing well academically, and who had been somewhat disruptive in class, started programming in LOGO. As he played on the computer, his typing became very fast (QWERTY keyboards are quite properly regarded by the Lamplighter children as a stupid arrangement with which they reluctantly work), and his programs became sophisticated. He was heard to remark, "I can't believe how fast my fingers are typing." He also could not believe how much fun school had become. Not only did he do well with LOGO, but he also became an attentive, productive student.

At present, most of the third and fourth graders—and even some of the first and second graders—are writing LOGO programs. And this includes some fairly sophisticated programs which use recursion and the concepts of state transparency. It's obvious that LOGO has indeed furthered Lamplighter's goal of igniting the imaginations and intellects of its children. But more importantly, LOGO has the potential to fire up imaginations *everywhere.*



Hank Gorman will conduct a July 18th workshop on LOGO. Interested parties contact him at Austin College.

Please Check Our Dealer Directory
(Page 64)

DEALER DIRECTORY

Goleta, CA

See our complete line of personal and business computers including: Texas Instruments, Atari, Apple, Altos, Data General. Also All accessories, software and supplies. Plus all that's new in personal electronics. **Personal Electronics, 5674 Calle Real Center, Goleta, CA 93117, (805) 967-5322.**

Los Angeles, CA

A software/systems house primarily engaged in the sale of software for TI and Polymorphic computers. Hardware sales are usually included in a combined turnkey package. Prospective customers are seen by prior telephone appointment only. **AAAA Discount Computer How's, 13022 Psomas Way, Los Angeles, CA 90066, (213) 391-8777.**

Greeley, CO

K&K Electronics for everything you need! HBO receive dishes, Satellite earth stations, plus a full line of Texas Instruments Home Computer Hardware and Software. **K&K Electronics, 615 8th Avenue, Greeley, CO 80631, (303) 352-3237.**

Waterloo, IA

Texas Instruments Large selection of Accessories and Software for 99/4 at Dhein's True Value, 7 W. Airline Hwy, Waterloo, IA 50701. Sale prices at store are also given on mail orders. Write for price list. **Dhein's True Value, 7 W. Airline Highway, Waterloo, IA 50701, (319) 232-6225.**

Evansville, IN

Independent Register 2414 N. Governor (across from Town Center Mall) is the tri-state location for TI-99/4 Hardware/Software, Supplies etc. We supply the tri-state with popular magazines and software for other units. Open Mon - Sat. 8-5. **Independent Register, 2414 N. Governor, Evansville, IN 47711. (812) 424-8246.**

Lexington, KY

A complete line Texas Instruments dealer. We stock all Texas Instruments calculators and accessories, learning aids, home computer products, accessories, peripherals and software. Atari computers and Hewlett Packard calculators in stock. **CBM Incorporated, 198 Moore Drive, Lexington, KY 40503, (606) 276-1519.**

Minneapolis, MN

Authorized Texas Instruments TI-99/4 computer dealer offering you the best prices in the U.S.A. on the computer and all peripherals. Also, a wide variety of printers. CRT's available at lowest prices. No collect calls, please. **Calculators, Inc., 7409 Fremont Avenue South, Minneapolis, MN 55423, (612) 866-8908.**

Dayton, OH

The only full line TI distributor stocking Semiconductors, DSG (700-800 OMNI Products) TM990 Microcomputer Boards, Calculators, complete in depth inventory including 99/4 Products-Software. Ohio 1-800-762-9510. Surrounding area 1-800-543-9550. Will ship. Call us. **Esco Inc., P. O. Box 1166, 221 Crane St. Dayton, OH 45403, (513) 226-1133**

Medford, OR

We have the TI-99/4 computer, Atari computers, video games, plus a large selection of cartridges, software books and computer magazines. We offer membership in "The Source." You can have the world at your fingertips. **The Computer Chip, 2640 "B" Barnett Road, Medford, OR 97501, (503) 776-4091.**

Salem, OR

J. Harvey's Video Clubhouse VCR's, Tapes, Home Computers TI-99/4, Atari 400, 800. **J. Harvey's Video Clubhouse 3295 Triangle Drive #142, Salem, OR 97302, (503) 581-1003.**

Call a Dealer Today

Dealers:

Listings are **$25** per issue; minimum insertion, 3 issues (6 months). Prepayment of **$75** required. Ads include 35 words describing products and services, plus company name, address and phone. (No merchandise prices, please.) Call Pat at 503-485-8796 or write **99'er Magazine**, Ad Department, 2715 Terrace View Drive, Eugene, OR 97405.

New York, NY

"Your Personal Guide to Personal Technology." We carry the complete TI-99/4 System. For demonstration of the TI-99/4 at home, school or office, or more information call Next Tech, Inc. today. **Next Tech, Inc., 350 Fifth Avenue, New York, NY 10001, (212) 792-8768.**

Rye, NY

Wide assortment of TI software, hardware and accessories all at discount prices. Also authorized Scott, Foresman dealer. Write or phone for nine-page catalog with hundreds of items. **Microcomputers Corporation, P. O. Box 191, Rye, NY 10580, (914) 967-8370.**

Electric City, WA

Texas Instruments 99/4 Home Computer Systems — Wholesale prices and not on just the main console! Also good prices on Epson printers and Televideo Terminals. For price list, write to: **Komputar Works, P. O. Box 483, Electric City, WA 99123, (509) 633-2653.**

Green Bay, WI

We specialize in the Texas Instruments 99/4 computer. We handle a complete accessory line for it, at discount prices. Custom programming for the 99/4 also done. Write for a complete price list. **The Micro House, 527 Simonet Street, Green Bay, WI 54301, (414) 432-2871.**

Call a Dealer Today

ATTENTION TI DEALERS

Did You Know That You Can Be Reimbursed For Advertising Texas Instruments Products In This Magazine?

Call or Write

Our Ad Department Today!

99'er Magazine / Ad. Dept.
2715 Terrace View Drive
Eugene, Oregon 97405
(503) 485-8796

Music Maker

By Norma & John Clulow

To paraphrase Shakespeare, "The computer that hath no music in its chips nor is not programm'd with concord of sweet sound is fit but for business, mathematics, and sorts." The TI-99/4 is definitely not one of these.

Outstanding music and sound effects capabilities are among the many features which set the TI-99/4 apart from other personal computers. A user can generate three simultaneous tones and a noise, plus specify their duration, pitch, and loudness—all with a *single* TI BASIC statement. The sound is played through the speaker of the color monitor or TV display.

Of course, an assortment of beep's, "tadaa's," and outer space sounds can greatly enhance a graphics presentation and provide useful auditory feedback during program execution. But when the sophisticated sound capabilities of the TI-99/4 become the **focus** of the programmer's attention, the Texas Instruments machine becomes a musical instrument in its own right. Whether a Bach Invention or a contemporary composition of your own creation, a successful TI-99/4 performance is worth the programming effort.

With the introduction of TI's *Music Maker* Command Module, you can take full advantage of the TI-99/4's sound capabilities without having to write a complex BASIC program. The Music Maker allows you to write a composition using either of two methods—Traditional Mode or Sound Graphs. While Traditional Mode requires some knowledge of fundamental music theory, Sound Graphs does not. Both methods are graphics based, in contrast to other music editor formats which require entry of notes using ASCII characters. Both also make superb use of the TI-99/4's outstanding color graphics capabilities. Notes are entered by manipulating the cursor with either the joysticks or the arrow keys. A composer can then print out the bass and treble clefs of each measure—complete with all notes, sharps, flats, and rests—with TI's thermal printer (using its special graphic character set). It's also possible to save the completed musical score on cassette tape or diskette.

Traditional Mode

In Traditional Mode, notes are entered directly on the music staff using standard notation. The first step involves defining the key, meter, and tempo. All possible key signatures (0-7 sharps or flats) are allowed. The meter or time signature options for the denominator are 1, 2, 4, 8, and 16—corresponding to the unit of measure receiving one beat (i.e., whole, half, quarter, eighth, or sixteenth note). The numerator of the time signature indicates the number of such units which comprise a measure. Your options here are restricted to values equal to or less than the denominator. Examples of allowable time signatures are:

$\frac{4}{4}, \frac{6}{8}, \frac{2}{2}$, and $\frac{3}{4}$;

but $\frac{3}{2}, \frac{12}{8}$, and $\frac{5}{4}$ are not, since the numerator exceeds the denominator. The significance of this limitation is tied to the fact that when music is accurately interpreted by a performer, there is a natural accent which falls on the first beat of every measure. This regular impulse, together with phrasing and secondary accents in compound meters, gives a composition its underlying rhythmic structure. The *Music Maker* does not automatically provide for this natural rhythm. The implementation of accent is entirely up to you. For example, a composition written in $\frac{4}{4}$ time may be made to sound like $\frac{3}{2}$ time with proper phrasing and specification of accent. Therefore, the time signature limitation does not actually limit the music you can write with the module. Finally, tempo is specified as a number from 1 to 30, corresponding approximately to metronomic indications from 25 to 128 quarter notes per minute—a sufficient range for nearly all compositions.

After these parameters are defined, the graphics representation for the first measure appears. Some music editors for other machines do not use graphics at all. It is a great advantage to see your composition displayed in standard notation as you are writing it.

Up to three voices may be "drawn" using whole, half, quarter, eighth, and sixteenth notes and their corresponding rests. Single dotting can be used with notes but not with rests. The notes for

each voice are represented in a different color, which facilitates identification of voices when editing.

The pitch range is three octaves, extending from the second A below middle C (bottom space of bass clef) to the second A above middle C (first ledger line above treble clef). This may seem like a wide range. But in arranging several piano pieces for the TI-99/4, we found that it was frequently necessary to make octave transpositions for notes extending beyond the *Music Maker*'s pitch range in Traditional Mode. On the other hand, the *Music Maker* is really not intended for the transcription of existing music written for other instruments, but rather to facilitate original composition. And like all instruments, it does have limitations which must be taken into account when preparing an original composition.

Accidentals (sharps, flats, or naturals not planned for in the key signature) must be written for each note; once written they do not carry over through the entire measure as they do in standard notation. For someone who is accustomed to standard notation, this may take a little getting used to. Additionally, the large and legible graphic symbols that the cursor picks up from the menu, become too small to be easily read when placed beside a note.

Graphics characters for the notes themselves resemble square notation, but we do not feel this detracts from their readability (especially when compared with the legibility of many manuscripts). However, in drawing clusters of two or more notes, we encountered a peculiar graphics-related difficulty. This is a function of the position (up or down) of an existing note stem. You will find that a note for one voice can not be placed at a pitch immediately above or below an existing note if that pitch is occupied by the stem of the existing note. The stems for voices one and two go upward unless they are placed immediately below a note in another voice which has its stem going upward. The opposite is true of voice-three notes. This means that while it is possible to represent any two-note cluster, the process can be more involved than would seem necessary. For instance, suppose you have already written a voice-one quarter note at middle C, and you want to write a voice-two note at D immediately above it. Finding that you cannot do this simply. You would have to do the following: Change voices, erase the C, change voices, draw the D (voice-two), change voices, redraw the C (stem downward), and finally change back to voice-two to continue. A cluster of three notes with adjacent pitches can not be written at all. These problems will be troublesome only in the event that the composer wants to write dissonant chords in the form of clusters.

At the bottom of the display there is a double row of squares; the upper row is used to specify volume for each note. There are eight levels of volume which allow a very smooth crescendo or diminuendo without abrupt transitions from one level to the next. By way of contrast, some other music editors do not allow this degree of versatility in dynamics. The default value for loudness is the maximum level of eight. If you want to accent selected notes, say the first note of every measure, you must drop the volume of all other notes. A default loudness of six or seven might have been a little easier to use in this regard.

The bottom row of squares is used to indicate the width of each note; this is very helpful in positioning them. It also allows one to create rests without using rest graphics by simply leaving a gap between one note and the next. Two adjacent notes of the same pitch are automatically tied. The only way to articulate them is to leave a gap in between. For instance, one might write a dotted quarter rather than a half note or a dotted eighth rather than a quarter note, and the resulting gap would then prevent a tie with the next note.

At any point during the writing of a measure, you can play an individual voice or all voices. And if you decide to make a change, this is easily accomplished by erasing an individual note or the entire voice. However, you cannot insert or delete notes without making necessary adjustments to other notes in the measure.

Repetition is easily handled by copying an individual voice or all voices from a previous measure, and this can save a great deal of time. A given voice cannot be copied as another voice, however. So if you want to use the copy feature to write rounds, they have to be scored differently than they would be in traditional composition. Any two voices can be copied by copying all three, and then erasing the one which is not wanted.

When you are finished with a measure, you can either go on to the next measure or back to a menu which allows you to edit, play, save, or print your composition. If you choose to edit, you will be shown the number of measures completed and the percentage of file space used, and you will be given the option of changing the tempo. To play the composition, you specify which voices are to be played, and you are given the option of hearing the music transposed up or down by as many as eleven half steps (twelve half steps are an octave). However, if you transpose a note so that it falls *below* the *Music Maker*'s range, it will not be played. You can interrupt the playing of a composition and view the graphic representation of the measure being played at that point, but graphics are not used when the piece is actually being played.

There are a few features present in some music editors for other machines which are not present in *Music Maker*. For example, the only way to initiate repeats is by manually pressing "SHIFT R" during the playing of a piece; no form of looping can be structured into a composition. However, given the relatively vast storage space available (compared with music compositions written in TI BASIC), together with the copying feature, the lack of repeat capability is less significant than it might otherwise be. With 16K of RAM, you will be able to write about 900 notes for *each* of the three voices. For example, writing all sixteenth notes for three voices, the file could be 57 measures long; with all quarter notes, it could be 224 measures. Additionally, there is no capability to write phrases and then arrange them in different voices. This capability could be useful when employing the device of imitation, such as in writing canons and fugues. Even so, the same effect *can* be achieved with *Music Maker*—it just takes a little more effort.

In summary, despite the few shortcomings mentioned, the Traditional Mode provides a beautiful graphics-based editor which makes the process of writing music as enjoyable as listening to the finished product. Even if this were the extent of the *Music Maker*'s capabilities, we feel it would be an excellent investment at the suggested retail price of $39.95.

Sound Graphs

While some knowledge of music theory is essential for effective use of the Traditional Mode, the Sound Graphs method may be used without any prior understanding of music terminology. As the name implies, music is entered in a Cartesian coordinate graph format. The frequency graph can have a resolution of one hundred twenty vertical positions (frequency) by twenty horizontal positions (duration) per "measure." A Sound Graphs music file may contain up to 46 measures. A color-coded line is plotted on the graph with the cursor, and as in Traditional Mode, a different color is used for each voice.

The volume graph has a resolution of eight vertical positions (volume) by twenty horizontal positions (duration), and appears below the frequency graph. A separate volume graph may be plotted for each voice appearing in the frequency graph (default is the highest volume). In addition to the three voices, a Sound Graph may also include a noise which is plotted on the volume graph.

The user has the option of Discrete or "Continuous" tones. Under the Discrete option, the vertical axis is divided into thirty frequencies, consisting of C Major diatonic pitches from the

second A below middle C to the third B above middle C. However, you can change any or all of these pitches with the List Tones option. Although any frequency from 110 Hz to 20,000 Hz can be used, tables are provided in TI's excellent documentation, giving the frequencies for chromatic, pentatonic, and gypsy scales. The frequencies can be changed at any time, even during or after the plotting of a Sound Graph.

Under the Continuous option you specify the upper and lower limits of the frequency range. These can be changed as often as you wish. The frequency axis is divided into 120 steps within this range, giving a frequency "slide" which sounds continuous and can be used to create sound effects such as whistles and sirens as well as interesting experimental music sounds. When you take into consideration the fact that a noise can be used in addition to three voices and that the composition can be played as fast as twenty characters per second, the range of possibilities is quite extensive.

In evaluating the noise, we were surprised to find that we could not distinguish any difference between the periodic and "white noise" groups—i.e., noises 1-4 and 5-8, respectively. Noise 1 appears to be the same as noise 5; noise 2 the same as 6, etc. If you are familiar with the difference between periodic noise and white noise in TI BASIC, do not expect to find the same distinction using the *Music Maker*.

Other aspects of using Sound Graphs are identical with the corresponding procedures used in Traditional Mode (viz., editing, playing, saving, and printing).

If you have no knowledge of music theory, using Sound Graphs is a great way to begin exploring the TI-99/4's music capabilities. If you are familiar with music fundamentals, you will be amazed at the versatility of the Sound Graphs method, and you will find that your TI-99/4 has potential you would not have thought possible.

In conclusion, the *Music Maker* Command Module will greatly enhance one of the already outstanding features of your computer—its capacity for sound and music. We believe it is an accessory you will not want to be without.

A Music Text Editor & File Player for the TI-99/4

By Norma & John Clulow

For those readers who do not as yet have a *Music Maker*, but would like to experiment with music writing anyway, a primitive music text editor follows this article, plus a file player program with the input required to play Bach's "Invention in F."

The *Music Text Editor* program creates a tape file which is read and played by the *Music File Player* that follows it. Although the file can be played by the editor, the tempo will be somewhat slower than when performed by the separate player program.

Use the following symbols to write note values :

W - Whole	H - Half
Q - Quarter	E - Eighth
S - Sixteenth	

For a dotted note value, use one of these symbols followed by a " . "; e.g., "S .", "Q .", etc.

Use the following symbols for pitches :

A	A#	E	
B		F	F#
C	C#	G	G#
D	D#	R	(rest)

After each pitch, give the octave (1-4) :

Octave	Begins
1	Bottom space, bass clef
2	Top line, bass clef
3	2nd. space from bottom, treble clef
4	1st. ledger line above treble clef

For example, to play an F# in octave 2 and a C in octave 1 for a dotted eighth followed by a sixteenth rest, you would enter : E . /F#/C1/ followed by S/R/R/
The following commands may be used :

CHANGE, REDO, LIST, PLAY, and SAVE

The save command merges new data with data already stored in the tape file. You can SAVE a file in as many sections as you wish, but when you answer the question, "FINISHED?" with a "yes", an end-of-file mark will be written on the tape file. The *Music File Player* can read and play a file consisting of 550 lines maximum.

```
100 REM  **********************
110 REM  *                    *
120 REM  * MUSIC TEXT EDITOR  *
130 REM  *                    *
140 REM  **********************
150 REM
160 REM  BY NORMA & JOHN CLULOW
170 REM  99'ER VERSION 5.81.1
180 CALL CLEAR
190 DIM N$(11,4),V$(8,1),P(349,2)
200 INPUT "TITLE    - ":T$
210 INPUT "COMPOSER - ":C$
220 FOR I=0 TO 11
230 READ N$(I,0)
240 NEXT I
250 DATA A,A#,B,C,C#,D,D#,E,F,F#,G,G#
260 FOR I=1 TO 4
270 FOR J=0 TO 11
280 X=110*2^(S/12)
290 GOSUB 1420
300 N$(J,I)=STR$(X)
310 S=S+1
320 NEXT J
330 NEXT I
340 CALL CLEAR
350 PRINT "ENTER TEMPO (M.M.)":"QUARTER NOTES/MINUTE"::
360 INPUT "M.M. = ":T
370 IF T<127 THEN 410
380 PRINT :"MAXIMUM IS 126"::
390 CALL SOUND(150,220,1)
400 GOTO 360
410 CALL CLEAR
420 T=6E+4/T
430 FOR I=0 TO 8
440 FOR J=0 TO 1
450 READ V$(I,J)
460 NEXT J
470 NEXT I
480 DATA W,4,H,2,H.,3,Q,1.1,Q.,1.5,E,.5,E.,.75,S,.25,S.,.375
490 FOR I=0 TO 8
500 X=VAL(V$(I,1))
510 X=T*X
520 GOSUB 1420
530 V$(I,1)=STR$(X)
540 NEXT I
550 CALL CLEAR
560 KP=0
570 GOSUB 1330
580 P0=1
590 X$=""
600 FOR N=1 TO 4-LEN(STR$(K+KP+1))
610 X$=X$&" "
620 NEXT N
630 INPUT STR$(K+KP+1)&X$&" = ":X$
640 IF K=350 THEN 670
650 P(K,1)=111
660 P(K,2)=111
670 IF X$<>"CHANGE"THEN 700
680 GOSUB 1570
690 GOTO 580
700 IF X$<>"PLAY"THEN 730
710 GOSUB 1200
720 GOTO 580
730 IF X$<>"REDO"THEN 760
740 GOSUB 1470
750 GOTO 580
760 IF X$<>"LIST"THEN 790
770 GOSUB 1710
780 GOTO 580
790 IF X$<>"SAVE"THEN 810
800 GOSUB 2160
810 IF K<>350 THEN 840
820 PRINT "SAVE FILE BEFORE PROCEEDING"::
830 GOTO 580
840 GOSUB 1370
850 IF S$=""THEN 870
860 IF PN<>0 THEN 890
870 CALL SOUND(150,220,1)
880 GOTO 580
890 FOR I=0 TO 8
900 IF S$=V$(I,0)THEN 950
910 NEXT I
920 PRINT "ERR- VALUE"
930 CALL SOUND(150,220,1)
940 GOTO 580
950 P(K,0)=VAL(V$(I,1))
960 FOR I=1 TO 2
970 GOSUB 1370
980 IF S$=""THEN 1000
990 IF PN<>0 THEN 1020
1000 CALL SOUND(150,220,1)
1010 GOTO 580
1020 IF S$="R"THEN 1150
1030 IF SEG$(S$,LEN(S$),1)<"5"THEN 1060
1040 CALL SOUND(150,220,1)
1050 GOTO 580
1060 OC=VAL(SEG$(S$,LEN(S$),1))
1070 S$=SEG$(S$,1,LEN(S$)-1)
1080 FOR J=0 TO 11
1090 IF S$=N$(J,0)THEN 1140
1100 NEXT J
1110 PRINT "ERR- VOICE";I
1120 CALL SOUND(150,220,1)
1130 GOTO 580
1140 P(K,I)=VAL(N$(J,OC))
1150 NEXT I
1160 IF SR=0 THEN 1180
1170 RETURN
1180 K=K+1
1190 GOTO 580
1200 FOR L=0 TO K-1
1210 IF P(L,1)<>111 THEN 1270
1220 IF P(L,2)<>111 THEN 1250
1230 CALL SOUND(P(L,0),110,30,110,30)
1240 GOTO 1310
1250 CALL SOUND(P(L,0),110,30,P(L,2),1)
1260 GOTO 1310
1270 IF P(L,2)<>111 THEN 1300
1280 CALL SOUND(P(L,0),P(L,1),1,110,30)
1290 GOTO 1310
1300 CALL SOUND(P(L,0),P(L,1),1,P(L,2),1)
1310 NEXT L
1320 RETURN
1330 FOR I=0 TO 349
1340 P(I,0)=0
1350 NEXT I
1360 RETURN
1370 PN=POS(X$,"/",P0)
1380 IF PN=0 THEN 1410
1390 S$=SEG$(X$,P0,PN-P0)
1400 P0=PN+1
1410 RETURN
1420 Y=X-INT(X)
1430 IF Y<.5 THEN 1450
1440 X=X+1
1450 X=INT(X)
1460 RETURN
1470 M=K
1480 INPUT "START AT LINE - ":K
1490 IF K>KP THEN 1520
1500 CALL SOUND(150,220,1)
1510 GOTO 1480
1520 K=K-KP-1
1530 IF K<M+1 THEN 1560
1540 CALL SOUND(150,220,1)
1550 GOTO 1480
1560 RETURN
1570 SR=1
1580 M=K
1590 INPUT "CHANGE LINE - ":K
1600 IF K>KP THEN 1630
1610 CALL SOUND(150,220,1)
1620 GOTO 1590
1630 K=K-KP-1
1640 IF K<M THEN 1670
1650 CALL SOUND(150,220,1)
1660 GOTO 1590
1670 GOSUB 580
1680 K=M
1690 SR=0
1700 RETURN
1710 INPUT "FIRST LINE - ":O
1720 IF O>KP THEN 1750
1730 CALL SOUND(150,220,1)
1740 GOTO 1710
1750 INPUT "LAST LINE - ":Q
1760 IF Q>KP THEN 1790
1770 CALL SOUND(150,220,1)
1780 GOTO 1750
1790 IF O<=Q THEN 1820
1800 CALL SOUND(150,220,1)
1810 GOTO 1710
1820 IF Q-KP<K+1 THEN 1840
1830 Q=K+KP
1840 PRINT ::
1850 O=O-KP-1
1860 Q=Q-KP-1
1870 FOR R=O TO Q
1880 X$=""
1890 FOR I=1 TO 4-LEN(STR$(R+1))
1900 X$=X$&" "
1910 NEXT I
```

Music Editor, continued . . .

```
1920 PRINT STR$(R+1)&X$&" - ";
1930 FOR S=0 TO 8
1940 IF P(R,0)=VAL(V$(S,1))THEN 1960
1950 NEXT S
1960 PRINT V$(S,0)&"/";
1970 FOR I=1 TO 2
1980 IF P(R,I)<>111 THEN 2040
1990 IF I<>2 THEN 2020
2000 PRINT "R/"
2010 GOTO 2140
2020 PRINT "R/";
2030 GOTO 2120
2040 FOR I1=0 TO 11
2050 FOR I2=1 TO 4
2060 IF P(R,I)<VAL(N$(I1,I2))THEN 2090
2070 IF P(R,I)=VAL(N$(I1,I2))THEN 2100
2080 NEXT I2
2090 NEXT I1
2100 IF I=2 THEN 2130
2110 PRINT N$(I1,0)&STR$(I2)&"/";
2120 NEXT I
2130 PRINT N$(I1,0)&STR$(I2)&"/"
2140 NEXT R
2150 RETURN
2160 IF F$="1"THEN 2200
2170 F$="1"
2180 OPEN #1:"CS1",SEQUENTIAL,INTERNAL,OUTPUT,FIXED 192
2190 PRINT #1:T$;C$
2200 I1=0
2210 K=K-1
2220 FOR I=1 TO 12
2230 FOR J=0 TO 2
2240 PRINT #1:STR$(P(I1,J)),
2250 NEXT J
2260 I1=I1+1
2270 IF I1>K THEN 2330
2280 NEXT I
2290 PRINT #1:STR$(P(I1,0)),STR$(P(I1,1)),STR$(P(I1,2))
2300 I1=I1+1
2310 IF I1>K THEN 2340
2320 GOTO 2220
2330 PRINT #1:""
2340 PRINT "FINISHED? (Y/N)"
2350 CALL KEY(0,KEY,STATUS)
2360 IF KEY=89 THEN 2410
2370 IF KEY<>78 THEN 2350
2380 KP=KP+K+1
2390 K=0
2400 GOTO 570
2410 PRINT #1:"@@@@@"
2420 CLOSE #1
2430 PRINT T$&" SAVED"
2440 END
```

```
100 REM  *********************
110 REM  *                   *
120 REM  * MUSIC FILE PLAYER *
130 REM  *                   *
140 REM  *********************
150 REM
160 REM  BY NORMA & JOHN CLULOW
170 REM  99'ER VERSION 5.81.1
180 CALL CLEAR
190 PRINT "THIS PROGRAM PLAYS A MUSIC  FILE
    CREATED BY THE 2-VOICE EDITOR."
200 PRINT :"* PLACE DATA IN CASSETTE CS1":"
    THEN PRESS ENTER"
210 CALL SOUND(150,1400,1)
220 CALL KEY(0,KEY,STATUS)
230 IF KEY<>13 THEN 220
240 DIM P(549,2)
250 OPEN #1:"CS1",SEQUENTIAL,INTERNAL,INPUT ,
    FIXED 192
260 PRINT :"* READING"
270 CALL SOUND(150,1400,1)
280 INPUT #1:N$,C$
290 FOR I=0 TO 549
300 FOR J=0 TO 2
310 INPUT #1:X$,
320 IF X$=""THEN 310
330 IF X$="@@@@@"THEN 370
340 P(I,J)=VAL(X$)
350 NEXT J
360 NEXT I
370 CALL CLEAR
380 PRINT TAB((28-LEN(N$))/2);N$::TAB(13);"BY"
    ::TAB((28-LEN(C$))/2);C$:::::::::
390 FOR K=0 TO I-1
400 IF P(K,1)<>111 THEN 440
410 IF P(K,2)<>111 THEN 470
420 CALL SOUND(P(K,0),110,30,110,30)
430 GOTO 500
440 IF P(K,2)<>111 THEN 490
450 CALL SOUND(P(K,0),P(K,1),1,110,30)
460 GOTO 500
470 CALL SOUND(P(K,0),110,30,P(K,2),1)
480 GOTO 500
490 CALL SOUND(P(K,0),P(K,1),1,P(K,2),1)
500 NEXT K
510 CALL CLEAR
520 PRINT "SELECT FROM...":::"(1) PLAY IT AGAIN"
    ::"(2) NEXT PIECE"::"(3) STOP"
530 CALL KEY(0,KEY,STATUS)
540 IF KEY<49 THEN 530
550 IF KEY>51 THEN 530
560 KEY=KEY-48
570 ON KEY GOTO 370,280,580
580 CLOSE #1
590 STOP
```

Entries for Invention in F

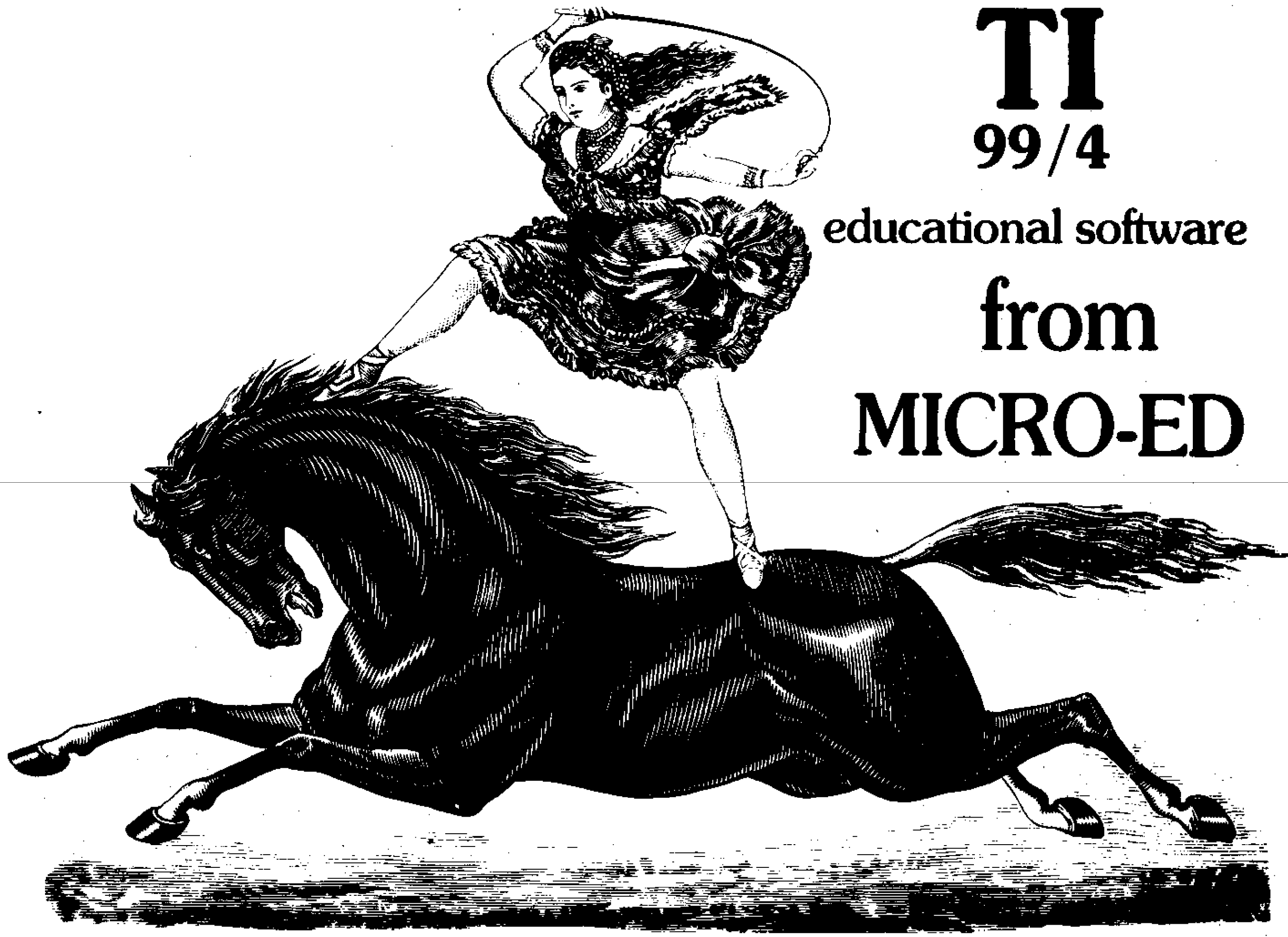
Primitive Music Editor

M.M. = 126

1 = S/F2/R/	29 = S/F3/A2/	57 = S/D3/F2/	85 = S/E3/C2/	113 = S/C3/F1/	141 = S/C3/B2/
2 = S/R/R/	30 = S/R/A#2/	58 = S/F3/A3/	86 = S/F3/R/	114 = S/R/G1/	142 = S/R/A2/
3 = S/A3/R/	31 = S/C3/A2/	59 = S/E3/G2/	87 = S/E3/G1/	115 = S/E2/F1/	143 = S/E3/G1/
4 = S/R/R/	32 = S/R/G1/	60 = S/F3/A3/	88 = S/D3/R/	116 = S/R/E1/	144 = S/R/A2/
5 = S/F2/R/	33 = S/A4/F1/	61 = S/G3/F2/	89 = S/C3/E2/	117 = S/F2/D1/	145 = S/C3/G1/
6 = S/R/R/	34 = S/C4/R/	62 = S/F3/A3/	90 = S/D3/R/	118 = S/R/E1/	146 = S/R/F1/
7 = S/C3/R/	35 = S/A#4/A2/	63 = S/E3/G2/	91 = S/C3/C2/	119 = S/C3/D1/	147 = S/G3/E1/
8 = S/R/R/	36 = S/C4/R/	64 = S/F3/A3/	92 = S/A#3/R/	120 = S/R/C1/	148 = S/R/F1/
9 = S/F2/R/	37 = S/A4/C2/	65 = S/G3/F2/	93 = S/A3/F2/	121 = S/E2/G1/	149 = S/C3/E1/
10 = S/R/R/	38 = S/C4/R/	66 = S/F3/A3/	94 = S/R/G2/	122 = S/R/F1/	150 = S/R/D1/
11 = E/F3/R/	39 = S/A#4/A2/	67 = S/E3/G2/	95 = S/D3/F2/	123 = S/C3/E1/	151 = E/C4/C1/
12 = S/E3/F1/	40 = S/C4/R/	68 = S/F3/A3/	96 = S/C3/E2/	124 = S/R/F1/	152 = S/B4/E1/
13 = S/D3/R/	41 = S/A4/F2/	69 = S/B3/B2/	97 = S/B3/D2/	125 = S/D2/G1/	153 = S/A4/R/
14 = S/C3/A2/	42 = S/C4/R/	70 = S/R/F2/	98 = S/C3/E2/	126 = S/R/R/	154 = S/G3/G1/
15 = S/D3/R/	43 = S/A#4/C2/	71 = S/G2/E2/	99 = S/B3/D2/	127 = S/B3/G1/	155 = S/A4/R/
16 = S/C3/F1/	44 = S/C4/R/	72 = S/R/F2/	100 = S/A3/C2	128 = S/R/R/	156 = S/G3/E1/
17 = S/A#3/R/	45 = S/F3/A3/	73 = S/D3/D2/	101 = S/G2/B2/	129 = E./C3/R/	157 = S/F3/R/
18 = S/A3/C2/	46 = S/A4/C3/	74 = S/R/F2/	102 = S/A3/C2/	130 = S/C3/C1/	158 = S/E3/C2/
19 = S/A#3/R/	47 = S/G3/A#3/	75 = S/B3/E2/	103 = S/G2/B2/	131 = S/C3/R/	159 = S/F3/R/
20 = S/A3/F1/	48 = S/A4/C3/	76 = S/R/F2/	104 = S/F2/A2/	132 = S/R/E1/	160 = S/E3/G1/
21 = S/G2/R/	49 = S/F3/A3/	77 = S/F3/D2/	105 = S/E2/G1/	133 = S/R/R/	161 = S/D3/R/
22 = E/F2/R/	50 = S/A4/C3/	78 = S/R/F2/	106 = S/F2/R/	134 = S/R/C1/	162 = S/C3/D#2/
23 = S/A3/E2/	51 = S/G3/A#3/	79 = S/D3/E2/	107 = S/E2/C2/	135 = S/R/R/	163 = S/A#3/R/
24 = S/R/D2/	52 = S/A4/C3/	80 = S/R/F2/	108 = S/D2/B2/	136 = S/R/G1/	164 = S/C3/F#1/
25 = S/C3/C2/	53 = S/F3/A3/	81 = S/G3/B2/	109 = S/C2/A2/	137 = S/R/R/	165 = S/A4/R/
26 = S/R/D2/	54 = S/A4/C3/	82 = S/A4/R/	110 = S/R/B2/	138 = S/R/C1/	166 = S/C3/D#2/
27 = S/A3/C2/	55 = S/G3/A#3/	83 = S/G3/G1/	111 = S/C3/A2/	139 = S/R/R/	167 = S/A4/R/
28 = S/R/A#2/	56 = S/A4/C3/	84 = S/F3/R/	112 = S/B3/G1/	140 = E/R/C2/	168 = S/A#3/F#1/

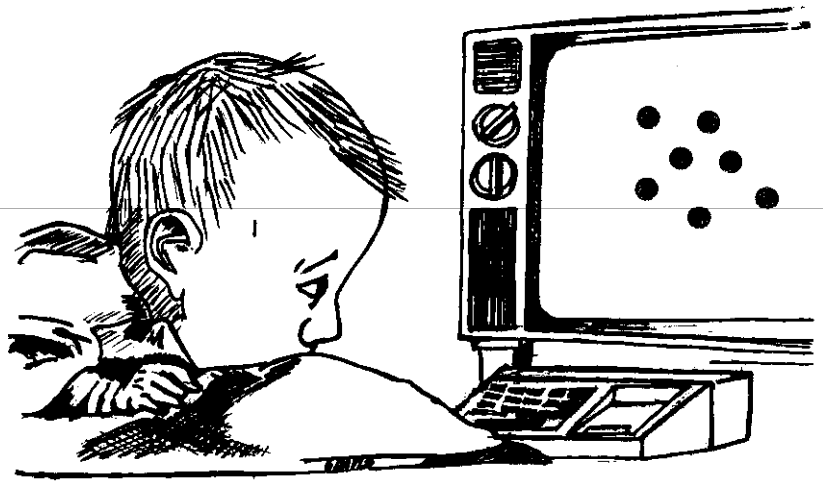
Data for Invention in F, continued . . . .

```
169 = S/A4/R/
170 = S/C3/D#2/
171 = S/A4/R/
172 = S/A3/F#1/
173 = S/A4/R/
174 = S/A#3/G1/
175 = S/R/G1/
176 = S/G2/F1/
177 = S/R/D#1/
178 = S/A#3/D1/
179 = S/R/D#1/
180 = S/G2/D1/
181 = S/R/C1/
182 = S/D3/A#1/
183 = S/R/C1/
184 = S/G2/A#1/
185 = S/R/A1/
186 = E/G3/G1/
187 = S/F3/G1/
188 = S/D#3/R/
189 = S/D3/A#2/
190 = S/D#3/R/
191 = S/D3/G1/
192 = S/C3/R/
193 = S/A#3/D2/
194 = S/C3/R/
195 = S/A#3/G1/
196 = S/A3/R/
197 = E/G2/R/
198 = S/A#3/F2/
199 = S/R/D#2/
200 = S/D3/D2/
201 = S/R/D#2/
202 = S/A#3/D2/
203 = S/R/C2/
204 = S/G3/A#2/
205 = S/R/C2/
206 = S/D3/A#2/
207 = S/R/A2/
208 = S/A#4/G1/
209 = S/R/F1/
210 = S/C#3/G1/
211 = S/R/E2/
212 = S/A#4/G1/
213 = S/R/E2/
214 = S/C#3/F1/
215 = S/R/E2/
216 = S/A#4/G1/
217 = S/R/E2/
218 = S/C#3/E1/
219 = S/R/E2/
220 = S/D3/F1/
221 = S/R/E1/
222 = S/A3/F1/
223 = S/R/D2/
224 = S/F3/F1/
225 = S/R/D2/
226 = S/D3/E1/
227 = S/R/D2/
228 = S/A4/F1/
229 = S/R/D2/
230 = S/F3/D1/
231 = S/R/D2/
232 = S/G3/A#2/
233 = S/F3/R/
234 = S/G3/G1/
235 = S/A#4/R/
236 = S/C3/E1/
237 = S/A#4/R/
238 = S/D3/G1/
239 = S/A#4/R/
240 = S/E3/C1/
241 = S/A#4/R/
242 = S/C3/E1/
243 = S/A#4/R/
244 = S/F3/A2/
245 = S/E3/R/
246 = S/F3/F1/
247 = S/A4/R/
248 = S/B3/D1/
249 = S/A4/R/
250 = S/C#3/F1/
251 = S/A4/R/
252 = S/D3/B1/
253 = S/A4/R/
254 = S/B3/D1/
255 = S/A4/R/
256 = S/E3/G1/
257 = S/D3/R/
258 = S/E3/E1/
259 = S/G3/R/
260 = S/A3/C#1/
261 = S/G3/R/
262 = S/B3/E1/
263 = S/G3/R/
264 = S/C#3/A1/
265 = S/G3/R/
266 = S/A3/C#1/
267 = S/G3/R/
268 = S/F3/D1/
269 = S/R/D1/
270 = S/D3/C1/
271 = S/R/D1/
272 = S/A#3/G1/
273 = S/R/D1/
274 = S/D3/A1/
275 = S/R/D1/
276 = S/G2/A#1/
277 = S/R/D1/
278 = S/F3/G1/
279 = S/R/D1/
280 = S/E3/C1/
281 = S/R/C1/
282 = S/C3/A#1/
283 = S/R/C1/
284 = S/A3/F1/
285 = S/R/C1/
286 = S/C3/G1/
287 = S/R/C1/
288 = S/F2/A1/
289 = S/R/C1/
290 = S/D#3/F1/
291 = S/R/C1/
292 = S/D3/A#1/
293 = S/F3/R/
294 = S/D#3/D1/
295 = S/F3/R/
296 = S/D3/F1/
297 = S/F3/R/
298 = S/D#3/D1/
299 = S/F3/R/
300 = S/D3/A#2/
301 = S/F3/R
302 = S/D#3/F1/
303 = S/F3/R/
304 = S/A#3/D2/
305 = S/D3/F2/
306 = S/C3/D#2/
307 = S/D3/F2/
308 = S/A#3/D2/
309 = S/D3/F2/
310 = S/C3/D#2/
311 = S/D3/F2/
312 = S/A#3/D2/
313 = S/D3/F2/
314 = S/C3/D#2/
315 = S/D3/F2/
316 = S/G2/A#2/
317 = S/A#3/D2/
318 = S/A3/C2/
319 = S/A#3/D2/
320 = S/G2/A#2/
321 = S/A#3/D2/
322 = S/A3/C2/
323 = S/A#3/D2/
324 = S/G2/A#2/
325 = S/A#3/D2/
326 = S/A3/C2/
327 = S/A#3/D2/
328 = S/E2/G1/
329 = S/R/A#2/
330 = S/C2/A2/
331 = S/R/A#2/
332 = S/G2/G1/
333 = S/R/A#2/
334 = S/E2/A2/
335 = S/R/A#2/
336 = S/A#3/G1/
337 = S/R/A#2/
338 = S/G2/A2/
339 = S/R/A#2/
340 = S/C3/E1/
341 = S/D3/R/
342 = S/C3/C1/
343 = S/A#3/R/
344 = S/A3/F1/
345 = S/A#3/R/
346 = S/A3/C1/
347 = S/G2/R/
348 = S/F2/A2/
349 = S/G2/R/
350 = S/F2/F1/
351 = S/D#2/R/
352 = S/D2/A#2/
353 = S/R/C2/
354 = S/G2/A#2/
355 = S/F2/A2/
356 = S/E2/G1/
357 = S/F2/A2/
358 = S/E2/G1/
359 = S/D1/F1/
360 = S/C2/E1/
361 = S/D2/F1/
362 = S/C2/E1/
363 = S/A#2/D1/
364 = S/A2/C1/
365 = S/A#2/R/
366 = S/A2/F1/
367 = S/G1/E1/
368 = S/F1/D1/
369 = S/R/E1/
370 = S/F2/D1/
371 = S/E2/C1/
372 = S/F2/A#1/
373 = S/R/C1/
374 = S/A2/A#1/
375 = S/R/A1/
376 = S/A#2/G1/
377 = S/R/A2/
378 = S/F2/G1/
379 = S/R/F1/
380 = S/A2/C1/
381 = S/R/A#1/
382 = S/F2/A1/
383 = S/R/A#1/
384 = S/G1/C1/
385 = S/R/R/
386 = S/E2/C1/
387 = S/R/R/
388 = Q./F1/F2/
```



Tiny Math for Tiny Tots

By Pat Kaplan



One evening, about a year ago, I picked out a book and curled up in front of the fireplace with it, expecting to soon become drowsy and have to go to bed. As it turned out, the book completely captured my imagination and wouldn't let me put it down until I had read it cover-to-cover, re-read selected parts, and had to hurry off, bleary-eyed to work the following morning. What made the experience so unusual was that the book didn't make any bestseller lists, wasn't made into a movie, and wasn't a thriller, exposé or sweeping Gothic romance . . .

The title of this book intrigued me the first time I saw it sitting on a bookstore shelf: *Teach Your Baby Math*—an improbable-sounding title, at best . . . The author, Glenn Doman, is the founder and director of The Institutes for the Achievement of Human Potential. He and his staff have worked many years with brain-injured children, and have been extremely successful in bringing these children from neurological dysorganization to average or superior neurological organization. By the mid-1960s, they were directly responsible for hundreds of severely brain-injured children who could read and read well, with total understanding at two years of age. These children were taught by their parents at home. Doman's work then naturally progressed to *well* children, and showed equally encouraging results.

Mathematics was then the next logical step. Doman and his staff already knew that you can teach a baby anything that can be presented in a factual way, and they knew that the younger a baby is, the easier it is to teach him or her. Their problem was simply to decide what were the actual facts of mathematics. Once you have taught a child the facts about anything, he will intuit the rules.

> Once you have taught a child the facts about anything, he will intuit the rules.

It took ten years before they were able to solve the problem. The solution that emerged was a daily program for teaching small children number perceptions, addition, subtraction, multiplication and division through the use of numerical dot cards. A child learns to recognize the number of dots (from one to one hundred) on the cards, and to use these quantitative relationships to do mathematical operations. The method goes beyond simply teaching a child to do math problems; it also has parents enhancing their baby's intelligence by actually increasing his brain growth.

I was hooked. This was something I just had to try with my two year old son. Well, to make a long story short—I have tried it, and it actually works! So, when my husband was starting this magazine, it seemed like a good idea for me to suggest, at one of the early staff meetings, that we implement the dot card method on the TI-99/4 and publish the program listing in the first issue. This we have done, and named *Tiny Math 1*.

Actually there are three reasons why I felt we should adapt the method for use via computer-assisted instruction, and publish the software: First, the actual production of the dot cards is both time-consuming and expensive (about $50 for materials); but it could instead be simulated quite well (by the computer) on a color TV or monitor. I want as many parents as possible to *try* the method with young children, without having to spend a lot of money. (By the way, this program can be played creatively as a family "number guessing game" – for toddler, older child and adult.) Secondly, I wanted to take advantage of the computer's ability to truly randomize the dot placement (instead of easier-to-recognize fixed pattern clusters inherent in a card system), and be able to change the dots to other shapes, and the white background to other colors. Additionally, I wanted to have speech synthesis as an option. Although the introduction of all these "bells and whistles" might have detracted from the basic simplicity of the method, I now feel that the element of novelty and interaction that has resulted, more than compensates. By the way, the third reason why I wanted to set up this unique learning method on the TI-99/4 and publish the details was, I'll admit, a little vain—I wanted to see my name in print . . .

Note:

To really understand this method, and put it into practice, you will also need the book, *Teach Your Baby Math*. As a convenience to our readers, we are stocking this title in the **99'er Bookstore**. See the book pages in this issue for price and ordering information.

Tiny Math 1

By W. K. Balthrop

The program starts by asking whether speech is desired. It then gives a child a choice of five different shapes to use for the screen "dots." The color of the shape will always be a dark red, but the background color of the screen can be changed. The parent or child then selects the range of dots to be displayed from two to one hundred. An example of this would be: Lowest possible number of shapes = 10. Highest possible number of shapes = 20. This would cause a number of shapes from 10 to 20 to appear randomly on the screen. The program then asks how long the shapes stay on the screen, from 1 to 7 seconds. If desired, this time range can be altered by changing the maximum time allowed in line 1000 of the program, or changing line 1610 for the time delay control number.

Before the exercise starts, the program offers a choice of either going into a teaching mode, or directly into the quiz mode. The teaching mode will cycle 25 times. The shapes will be displayed in the same fashion as in the quiz mode, but will remain on the screen with the exact number of dots indicated at the top. They remain there until a key is pressed—giving a child adequate time to observe the dots.

An important aspect of this program is that it will flash all of the shapes on the screen at the *same* time, this eliminates the problem of counting the shapes as they would appear, one by one. This is done by using CALL COLOR to make the shapes invisible. At the time the shapes are to appear, the CALL COLOR command is used to make them red. By the way, it's important that the child doesn't have the time to count.

In coding the program, I was faced with the problem of keeping the shapes centralized on the screen without crowding them. There shouldn't be two shapes touching each other, for that might be confusing; they shouldn't be spread out all over the screen either. It was necessary to do a lot of checking for adjacent characters, and give pattern size limits, depending on the number of shapes. All of this slows things down a little, so don't be surprised if it takes some time to display the shapes when there are a lot of them used.

Another feature is the re-testing of missed guesses. Approximately every other display will be one that the user has already missed—providing that he or she has missed any. When that number has been guessed right, it will be eliminated from the wrong guess storage.

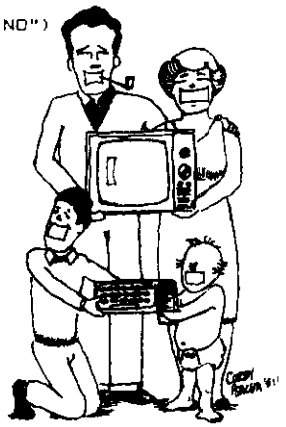
PROGRAM EXPLANATION

Line Nos.	
100-170	REM statements.
180-420	Print title page, and play music.
430-490	INPUT option to use speech.
500-680	INPUT type of shape desired.
690-770	INPUT number of shapes desired.
780-930	INPUT the screen color desired.
940-1010	INPUT time for shapes to be on the screen.
1020-1070	INPUT choice to teach first, or learn mode.
1080-1140	Teaching mode control loop.
1150-1210	Learning mode control loop.
1220-1320	END of program. Quit or continue.
1330-1580	PUT invisible dots on the screen; check for dots next to each other.
1590-1640	Blink dots on for a given time delay.
1650-1750	INPUT user's guess. Check for right or wrong answer.
1760-1900	Answer was right. If answer was in the wrong answer storage, it will now be removed.
1910-2100	Answer was wrong. Store missed number to be used later.
2110-2260	Turn dots on until enter is pressed. Display number of dots at bottom of screen.
2270-2880	Set up string variables to say the correct answer, when using speech.
2890-3000	Set up variables for next test.
3010-3040	END of test. Print score, and END.

```
100 REM  ****************
110 REM  *               *
120 REM  * TINY MATH 1   *
130 REM  *               *
140 REM  ****************
150 REM   BY W.K. BALTHROP
160 REM  99'ER VERSION 5.81.1
170 REM   INITIALIZATION
180 RANDOMIZE
190 CALL CLEAR
200 CALL SCREEN(16)
210 PRINT TAB(8);"TINY MATH 1"
220 PRINT ::::::::::
230 DIM G$(5),W(50)
240 G$(1)="183C7EFFFF7E3C18"
250 G$(2)="3C7EFFFFFFFF7E3C"
260 G$(3)="FFFFC3C3C3C3FFFF"
270 G$(4)="81667E3C3C7E6681"
280 G$(5)="FFFFFFFFFFFFFFFF"
290 DATA 523,494,440,494,523,523,523
291 DATA 494,494,494,523,523,523,523
292 DATA 494,440,494,523,523,523,523
300 DATA 494,494,523,494,440,44000
310 FOR X=1 TO 5
320 CALL CHAR(95+X,G$(X))
330 NEXT X
340 FOR ENT=1 TO 5
350 RESTORE
360 FOR ENR=1 TO 27
370 READ S
380 IF ENR=27 THEN 410
390 CALL SOUND(100,S,5)
400 CALL VCHAR(INT(RND*24)+1,INT(RND*32)+1,95+ENT)
410 NEXT ENR
420 NEXT ENT
430 REM   USER INPUTS
440 CALL CLEAR
450 CALL SCREEN(8)
460 PRINT "  DO YOU WANT TO USE SPEECH"
470 PRINT "   1 FOR YES"
480 INPUT "   2 FOR NO";S
490 IF (S<1)+(S>2)THEN 440
500 CALL CLEAR
510 PRINT "WHAT SHAPE DO YOU WANT TO   USE"
520 IF S<>1 THEN 540
530 CALL SAY("WHAT+SHAPE+DO+YOU+WANT+TO+USE")
540 PRINT "1 FOR ";CHR$(96)
550 PRINT
560 PRINT "2 FOR ";CHR$(97)
570 PRINT
580 PRINT "3 FOR ";CHR$(98)
590 PRINT
600 PRINT "4 FOR ";CHR$(99)
610 PRINT
620 PRINT "5 FOR ";CHR$(100)
630 INPUT CA$
```

Tiny Math 1, continued . . .

```
640 IF (ASC(CA$)<49)+(ASC(CA$)>54)THEN 630
650 CA=VAL(CA$)
660 IF CA>5 THEN 630
670 CALL CHAR(96,G$(CA))
680 CALL CLEAR
690 PRINT "ENTER THE NUMBER OF SHAPES  YOU WANT, FROM 1 TO 99."
700 IF S<>1 THEN 730
710 CALL SAY("ENTER+THE+NUMBER+OF+SHAPES+YOU+WANT")
720 CALL SAY("FROM 1 TO+NINETY 9 SHAPES")
730 INPUT "LARGEST NUMBER OF SHAPES TO BE ON THE SCREEN":MNOS
740 PRINT :::
750 INPUT "MINIMUM NUMBER OF SHAPES TO BE ON THE SCREEN":MIOS
760 IF MNOS<=MIOS THEN 680
770 IF (MNOS>99)+(MIOS<1)THEN 680
780 CALL CLEAR
785 PRINT "ALL SHAPES ARE RED"
790 PRINT "WHAT SCREEN COLOR DO YOU    WANT"
800 IF S<>1 THEN 820
810 CALL SAY("ALL+SHAPES ARE+RED.WHAT+SCREEN+COLOR+DO YOU+WANT")
820 PRINT
830 PRINT "3: MEDIUM GREEN"
840 PRINT "4: LIGHT GREEN"
850 PRINT "6: LIGHT BLUE"
860 PRINT "8: CYAN"
870 PRINT "11: DARK YELLOW"
880 PRINT "12: LIGHT YELLOW"
890 PRINT "15: GRAY"
900 PRINT "16: WHITE"
910 INPUT COL
920 CALL SCREEN(COL)
930 CALL CLEAR
940 PRINT "ENTER THE TIME YOU WANT THE"
950 PRINT "SHAPES ON THE SCREEN, FROM  1 TO 7 SECONDS."
960 IF S<>1 THEN 980
970 CALL SAY("ENTER+THE+TIME+YOU+WANT+THE+SHAPES+ON+THE+SCREEN,
    FROM 1 TO 7 SECOND")
980 PRINT ::::
990 INPUT T
1000 IF T>7 THEN 930
1010 CALL CLEAR
1020 PRINT "DO YOU WANT TO LEARN FIRST ENTER ""Y"" FOR YES,""N""FOR NO"
1030 IF S<>1 THEN 1050
1040 CALL SAY("DO+YOU+WANT+TO+LEARN+FIRST..YES OR NO")
1050 INPUT LE$
1060 IF LE$="N"THEN 1150
1070 IF LE$<>"Y"THEN 1030
1080 REM   TEACHING MODE CONTROL LOOP
1090 OP=1
1100 GOSUB 1320
1110 GOSUB 2120
1120 GOSUB 2270
1130 OP=OP+1
1140 IF OP<25 THEN 1100
1150 REM   LEARNING MODE CONTROL LOOP
1160 LO=1
1170 GOSUB 1320
1180 GOSUB 1600
1190 GOSUB 1660
1200 LO=LO+1
1210 IF LO<51 THEN 1170
1220 REM   END OF TEST. CONTINUE OR QUIT.
1230 CALL CLEAR
```



```
1240 PRINT "DO YOU WANT TO PLAY AGAIN."
1250 PRINT "ENTER ""Y"" FOR YES,""N""   FOR NO."
1260 IF S<>1 THEN 1280
1270 CALL SAY("DO+YOU+WANT+TO+PLAY+AGAIN")
1280 INPUT NG$
1290 IF NG$="N"THEN 3020
1300 IF NG$="Y"THEN 2900
1310 GOTO 1230
1320 NC=MIOS+INT(RND*(MNOS-MIOS))+1
1330 REM   DISPLAY INVISIBLE DOTS ON THE SCREEN
1340 RD=INT(RND*KT*2)+1
1350 IF RD>50 THEN 1340
1360 IF W(RD)=0 THEN 1380
1370 NC=W(RD)
1380 NOS=INT(MNOS/10)
1390 CALL CLEAR
1400 CALL COLOR(9,1,1)
1410 FOR LOOP=1 TO NC
1420 X=10-NOS+(INT(RND*((3+NOS)*2)+1))
1430 IF (X>23)+(X<4)THEN 1420
1440 Y=13-NOS+(INT(RND*((4+NOS)*2)+1))
1450 IF (Y>30)+(Y<3)THEN 1440
1460 CALL GCHAR(X,Y,CH)
1470 IF CH<>32 THEN 1420
1480 CALL GCHAR(X+1,Y,CH)
1490 IF CH<>32 THEN 1420
1500 CALL GCHAR(X-1,Y,CH)
1510 IF CH<>32 THEN 1420
1520 CALL GCHAR(X,Y+1,CH)
1530 IF CH<>32 THEN 1420
1540 CALL GCHAR(X,Y-1,CH)
1550 IF CH<>32 THEN 1420
1560 CALL HCHAR(X,Y,96)
1570 NEXT LOOP
1580 RETURN
1590 REM   BLINK DOTS ON FOR A GIVEN TIME DELAY.
1600 CALL COLOR(9,7,1)
1610 FOR LOOP=1 TO T*100
1620 NEXT LOOP
1630 CALL COLOR(9,1,1)
1640 RETURN
1650 REM    ASK FOR USER'S GUESS ON HOW MANY DOTS
1660 CALL CLEAR
1670 PRINT "WHAT IS THE NUMBER OF SHAPESPLEASE."
1680 IF S<>1 THEN 1700
1690 CALL SAY("WHAT+IS+THE+NUMBER+OF+SHAPES PLEASE")
1700 INPUT N$
1710 IF N$="Q"THEN 1230
1720 IF N$>STR$(MNOS)THEN 1680
1730 IF N$<STR$(MIOS)THEN 1680
1740 G=VAL(N$)
1750 IF G<>NC THEN 1920
1760 REM   RIGHT ANSWER
1770 CALL CLEAR
1780 PRINT "GOOD, THAT IS RIGHT."
1790 IF S<>1 THEN 1810
1800 CALL SAY("GOOD #THAT IS RIGHT# ")
1810 CALL SOUND(300,440,2,784,2,988,2)
1820 CALL SOUND(400,554,2,659,2,880,2)
1830 FOR TD=1 TO 1000
1840 NEXT TD
1850 FOR ER=1 TO 50
1860 IF W(ER)<>NC THEN 1880
```

Tiny Math 1, continued . . .

```
1870 W(ER)=0
1880 NEXT ER
1890 RIGHT=RIGHT+1
1900 RETURN
1910 REM   WRONG ANSWER.
1920 PRINT "U HOH, THAT IS INCORRECT"
1930 PRINT "THE RIGHT ANSWER IS";NC
1940 IF S<>1 THEN 1980
1950 GOSUB 2270
1960 CALL SAY("UHOH #THAT IS INCORRECT# ")
1970 CALL SAY("THE+RIGHT+ANSWER+IS","",NUM$,"",TE$,"")
1980 CALL SOUND(300,440,2,330,2,165,2)
1990 CALL SOUND(400,330,2,220,2,110,2)
2000 CALL COLOR(9,2,1)
2010 FOR V=1 TO 100
2020 NEXT V
2030 W(KT)=NC
2040 KT=KT+1
2050 PRINT "WE WILL TRY AGAIN"
2060 IF S<>1 THEN 2080
2070 CALL SAY("WE+WILL+#TRY AGAIN# ")
2080 FOR TD=1 TO 500
2090 NEXT TD
2100 RETURN
2110 REM    TURN ON DOTS PERMANENTLY. DISPLAY NUMBER OF DOTS.
2120 CALL COLOR(9,7,1)
2130 PRINT TAB(10);"#ENTER ANY KEY#"
2140 IF NC<10 THEN 2170
2150 CALL VCHAR(2,5,ASC(STR$(INT(NC/10))))
2160 GOTO 2180
2170 CALL VCHAR(2,5,32)
2180 CALL VCHAR(2,6,ASC(STR$(NC-(INT(NC/10)*10))))
2190 IF S<>1 THEN 2240
2200 NUM$=""
2210 TE$=""
2220 GOSUB 2270
2230 CALL SAY("THERE+ARE","",NUM$,"",TE$,"","SHAPES+ON+THE+SCREEN")
2240 CALL KEY(0,K,STAT)
2250 IF STAT=0 THEN 2240
2260 RETURN
2270 FC=INT(NC/10)
2280 SC=NC-(INT(NC/10)*10)
2290 IF FC=0 THEN 2320
2300 ON FC GOSUB 2540,2710,2730,2750,2770,2790,2810,2830,2850
2310 GOTO 2330
2320 FC$=""
2330 IF SC=0 THEN 2360
2340 GOSUB 2870
2350 GOTO 2370
2360 SC$=""
2370 IF NC>20 THEN 2480
2380 IF NC>9 THEN 2410
2390 NUM$=SC$
2400 RETURN
2410 IF SC>5 THEN 2450
2420 NUM$=FC$
2430 TE$=""
2440 RETURN
2450 NUM$=SC$
2460 TE$=FC$
2470 RETURN
2480 NUM$=FC$
2490 IF SC>0 THEN 2520
2500 TE$=""
2510 RETURN
2520 TE$=SC$
2530 RETURN
2540 IF SC>0 THEN 2570
2550 FC$="TEN"
2560 RETURN
2570 ON SC GOSUB 2590,2610,2630,
     2650,2670,2690,2690,
     2690,2690
2580 RETURN
2590 FC$="ELEVEN"
2600 RETURN
2610 FC$="TWELVE"
2620 RETURN
2630 FC$="THIRTEEN"
2640 RETURN
2650 FC$="FOURTEEN"
2660 RETURN
2670 FC$="FIFTEEN"
2680 RETURN
2690 FC$="TEEN"
2700 RETURN
2710 FC$="TWENTY"
2720 RETURN
2730 FC$="THIRTY"
2740 RETURN
2750 FC$="FOURTY"
2760 RETURN
2770 FC$="FIFTY"
2780 RETURN
2790 FC$="SIXTY"
2800 RETURN
2810 FC$="SEVENTY"
2820 RETURN
2830 FC$="EIGHTY"
2840 RETURN
2850 FC$="NINETY"
2860 RETURN
2870 SC$=STR$(SC)
2880 RETURN
2890 REM   SET UP VARIABLES
     FOR NEXT TEST.
2900 FOR X=1 TO 50
2910 W(X)=0
2920 NEXT X
2930 X=0
2940 Y=0
2950 NC=0
2960 NOS=0
2970 S=0
2980 OP=0
2990 LO=0
3000 GOTO 440
3010 REM   END OF TEST MESSAGE
3020 PRINT "YOUR SCORE WAS";RIGHT
3030 PRINT "OUT OF A POSSIBLE 50"
3040 END
```

Want to Get Published?

99'er Magazine is looking for articles in all areas of personal computing that concern the Texas Instruments TI-99/4 and other TMS9900-family systems (e.g., Marinchips, Technico, and the TM990/189). Here are the kinds of articles that we want you to write for us:

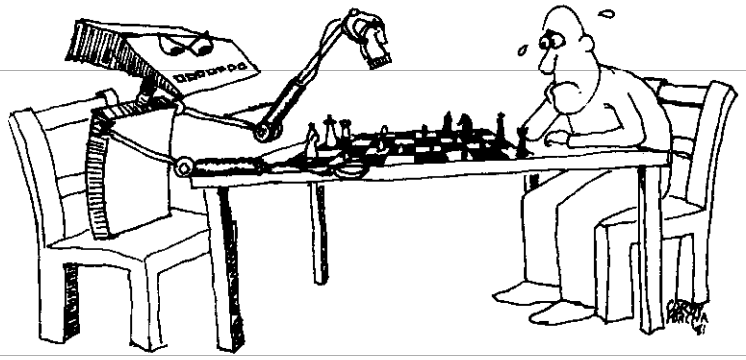
- Are you a businessman, professional, hobbyist, scientist, or engineer with an interesting microcomputer application? Tell us how it works, what problems you've had to overcome, and what recommendations you have for others. We're especially interested in sharing user-written software with our readers.
- Have you recently purchased a piece of hardware or software that hasn't quite come up to your expectations, or has, on the other hand, impressed you with its performance? We're looking for comprehensive product and book reviews from different perspectives.
- Are you an educator or parent with something to contribute to computer-assisted instruction (CAI)? We're always looking for new ideas and fresh approaches to educational problems.
- Have you created any unusual computer games or simulations? Let our readers experience your excitement and pleasure.
- Perhaps you've modified your microcomputer or have interfaced it with some unique or useful hardware. Send us your how-to-do-it story.

These are just some ideas. Perhaps you have others. Don't worry if you're not a professional writer. Our editorial staff stands ready to help polish up your manuscripts. And we'll be more than happy to send you a copy of our author's guidelines.

Send your manuscripts and correspondence to:

99'er Magazine / Editorial Dept.
2715 Terrace View Drive,
Eugene, Oregon 97405

COMPUTER CHESS CORNER



By Jerry Wolfe

The game of chess has fascinated men and women for hundreds of years. People from all walks of life and of all ages have enjoyed the challenges and entertainment it provides. The universal popularity of chess is undoubtedly due to its resemblance to life: Mastery of chess requires many of the same elements necessary to mastery of one's life—logical thought, long-range planning, the ability to recognize and act on sudden opportunities, persistence, patience, concentration, steady nerves, confidence, objectivity, and of course, lots of experience! Yes, to do well in chess does require all these things, but interestingly enough, practicing the game greatly helps develop and nourish these *same* characteristics and abilities! "Learning to Play" is, in reality, one and the same as "playing to learn."

And yet, for all its challenges and self-improvement attributes, the game is enjoyable at all levels of skill—from the raw novice, to the international master. Over the years, chess has indeed provided me with hundreds of hours of engrossing entertainment and many cherished friendships.

With the advent of strong chess-playing computer programs, chess has entered an important new stage of development. People can learn chess much more rapidly than before *without* the often deflating experience of losing many games "in public." This is especially true for children; losing badly to adults or other children can often drive them from the game. Having a ready and discreet opponent does indeed have its advantages . . .

About the Author

Jerry Wolfe is a professor of mathematics at the University of Oregon in Eugene, Oregon. He has been playing chess since the age of eleven and began playing in chess tournaments at the age of fourteen. He is the 1979 Oregon Open champion and has won numerous other local tournaments in the Pacific Northwest during his chess "career." Currently he holds the official rating of candidate master.

The TI *Video Chess* Command Module is one of the programs now available. It is a unique implementation since it is contained in 30K of ROM (no time-consuming cassette loading), can run on a "bare-bones" TI-99/4 (no disk drives or other peripherals are needed), uses a keyboard overlay to simplify commands, and has built-in chess clocks. This last feature is useful for users who wish to eventually play in chess tournaments where the use of chess clocks in mandatory. Thus, playing under tournament conditions is actually now possible in the privacy of one's own home!

Learning can progress much faster when a tireless, willing opponent is always ready to play!

In this initial article, I'll look briefly at the main features of the video chess program, examine some of its strengths and weaknesses, and make some suggestions for using the program to learn chess.

There are four main options available for using the program. You can (1) play chess against the computer, (2) play against another human opponent, (3) set up a problem for the computer to solve, or (4) have the program play as many as nine (!) opponents simultaneously. In addition, games or positions may be stored on cassette—an especially useful feature for postal players, or players without enough time to finish their games in one sitting.

When playing against the computer, you can control the playing characteristics of the program by choosing the experience level (beginner, novice, or intermediate), the time allotted to the computer for each move (30 seconds to 200 seconds), and the style of play (normal, defensive, or aggressive). The program also allows you to take back a move, ask for advice, have your move evaluated, or even switch sides!

In the problem mode, you can ask the program to solve a mate in two, three, or four moves. This is, of course, a potentially valuable learning tool, but the program's versatility doesn't stop there: you can also set up *any* position and have the computer play a normal game starting from the given position.

Based on many years of tournament experience, I would estimate the maximum strength of the program to be slightly less than the average player in a typical chess tournament. This is superior to probably 90% of chess players as a whole! And presumably, stronger versions of the program will be available in the future. To put this in perspective, the strongest chess-playing program in the world, running on the enormous and fast CYBER or CRAY computers, still does not play at the level of a human chess master. (It will, however, defeat 99% of chess players as a whole!)

As an educational tool, the video chess program is excellent. A beginning player can make rapid improvement in his game by adjusting the strength of the program as his own playing strength increases. If you're a new player, you should have at least one good book on chess that is designed for beginners. (There are many good ones on the market.) Then as you learn new ideas and techniques from reading, you can try them out against the computer immediately. For example, it is important for every player to master the basic checkmates—king and queen vs. king; rook and king vs. king; two bishops and king vs. king; etc. All good chess manuals discuss these in detail. After reading about how to mate with king and rook against king, say, you can immediately try it out using the program. Learning can progress much faster when a tireless, willing opponent is always ready to play!

The weakest part of the program is in the problem mode when asking for a mate in two, three, or four moves. For example, when I gave the program Problem No. 1 (below), it worked for two and one half hours without coming to a conclusion. I finally turned it off. And I have had similar disappointing results with rather easy mates in two. Fortunately, this defect is not terribly important, and may be alleviated in future

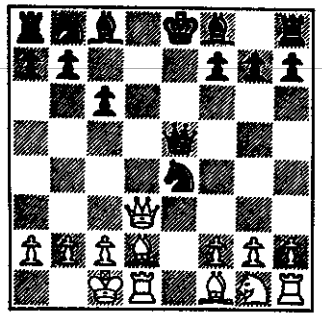
versions of the program. The best use for this problem solving mode seems to be in setting up positions from which the computer will commence playing as in a normal game. (Note: you can do this to learn the basic checkmates mentioned previously.)

In the next article I will report the results of a yet-to-be-played match between the *Video Chess* program and one of the best chess playing machines (the so-called "Boris" machine) which reportedly plays at a somewhat higher level than the average tournament player. I'd also appreciate hearing any comments, experiences, or questions that you might have pertaining to the *Video Chess* program or chess in general.

As a regular feature of this column, I will leave you with two problems to solve. Solutions will appear in the following issue. Problem No. 1 comes from a game between two grandmasters about sixty years ago. Problem No. 2 is a famous position—especially memorable because after black made the beautiful winning move, spectators showered the playing stage with gold coins. As it turned out, however, they were *not* showing their admiration . . . but rather, paying off their bets *in disgust!*

Problem No. 1

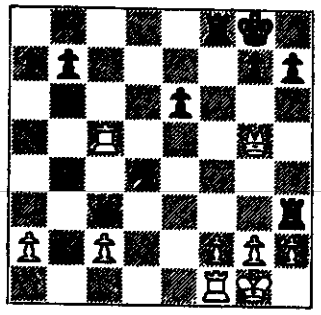
White:	Pawns:	A2 , B2 , C2 , F2 , G2 , H2
	Knights:	G1
	Bishops:	D2 , F1
	Rooks:	D1 , H1
	Queen:	D3
	King:	C1
Black:	Pawns:	A7 , B7 , C6 , F7 , G7 , H7
	Knights:	B8 , E4 ,
	Bishops:	C8 , F8
	Rooks:	A8 , H8
	Queen:	E5
	King:	E1



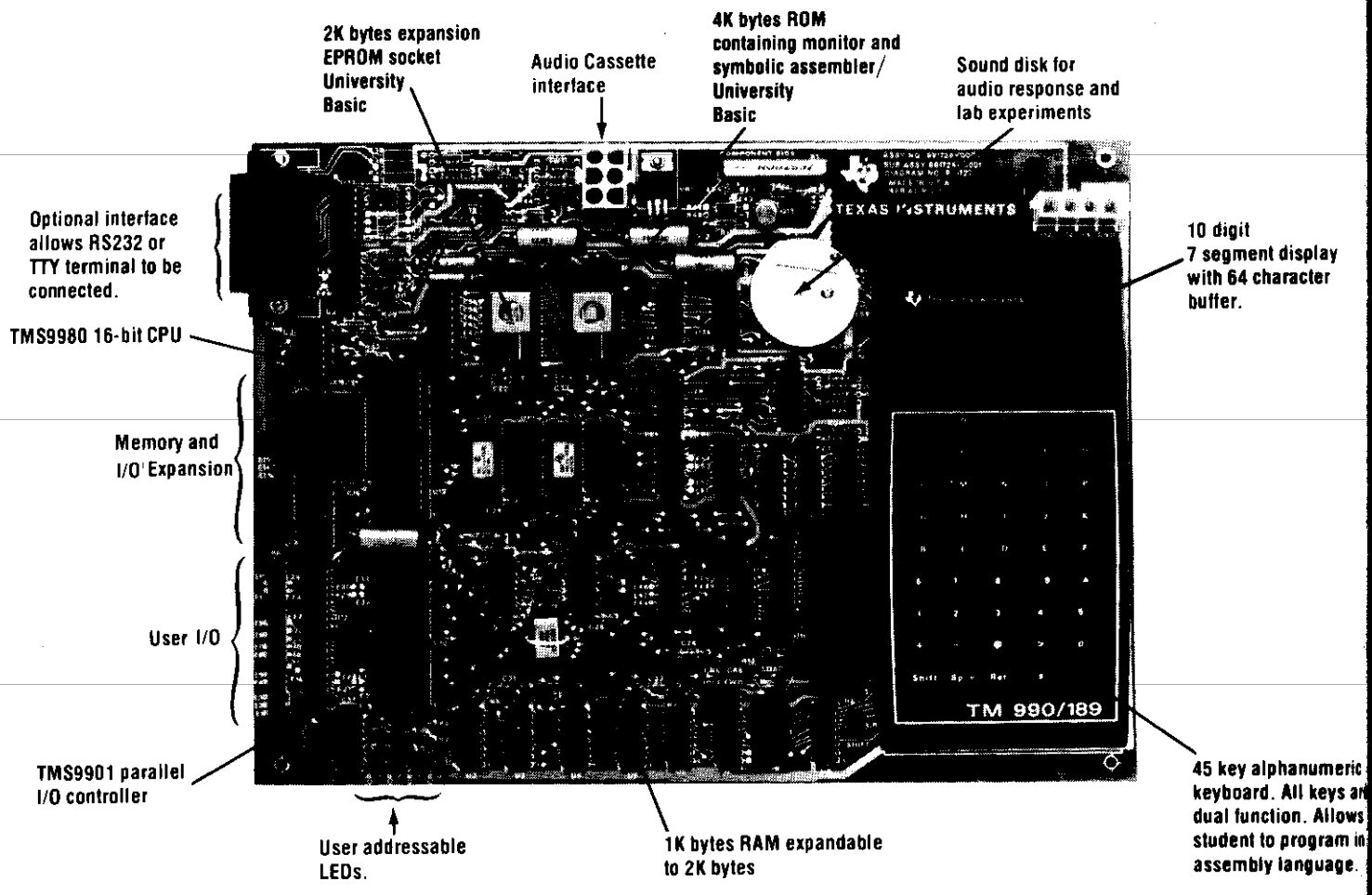
White to move and checkmate in three moves

Problem No. 2

White:	Pawns:	A2 , C2 , F2 , G2 , H2
	Knights:	None
	Bishops:	None
	Rooks:	C5 , F1
	Queen:	G5
	King:	G1
Black:	Pawns:	A7 , B7 , E6 , G7 , H7
	Knights:	D4
	Bishops:	None
	Rooks:	H3 , F8
	Queen:	C3
	King:	G8



Black to move and win (Black has a single crushing move)



The Texas Instruments TM 990/189 University Module

By Gary M. Kaplan

Computer-assisted instruction (CAI) continues to find exciting, new applications in many fields. Yet, in all the attempts by educators to promote these glamorous applications and exotic uses, the most "logical" instructional assistance that a computer can provide us with is often forgotten or largely ignored. I'm referring, of course, to using CAI to learn more about how computers themselves work. This is, indeed, a most practical application, since it fosters the particular experience and knowledge that is absolutely *essential* to the development of better CAI tools. As readers of this magazine undoubtedly know by now, our hardware focus is on the 16-bit TMS9900-family of microprocessors and its associated microcomputer configurations. And although the TI-99/4 is the most familiar 9900 consumer product, it is *not* the one best configured for hands-on experience with hardware and software operation and design. The product that is *best* for this purpose is, in fact, the TM 990/189 University Module—a self-contained, single-board microcomputer system that is intended for use as a learning aid in the instruction of microcomputer fundamentals, machine and assembly language programming, and microcomputer interfacing.

In this introductory article, we'll take a brief look at the components of this system, and examine its suitability for classroom work, for an in-house industrial trainer, and as an interactive private teacher for engineers or serious hobbyists. For TI-99/4 users interested in "getting into" assembly language programming and the "guts" of their machines, we'll follow this article with a tutorial series on using this University Module as both a teacher and an inexpensive hardware and software development tool—one that can interface with other 990-series boards, as well as the TI-99/4 personal computer. Applications, programs, and interfacing techniques from TM 990/189 users are most welcome.

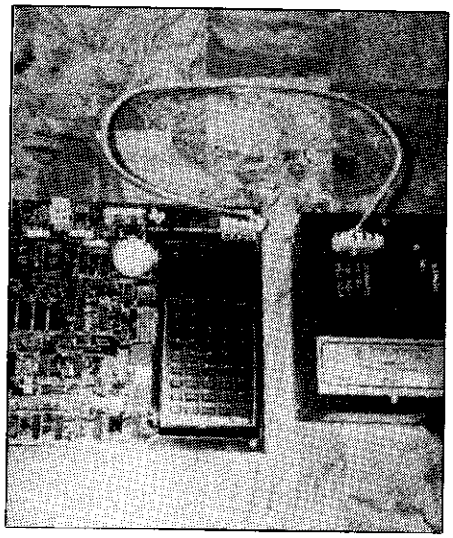
Those of you who have been yearning to rip apart your personal computer and find out what makes it tick, now have a more convenient option with this magazine-size, non-enclosed microcomputer board. Just about all the components found inside a TI-99/4 console are also found on this board—with the major exception of a video display processor chip. The layout, as seen from the above photo, is clean and uncluttered.

If you try to find the monstrous, 64-pin TMS9900 microprocessor (the

one used in the TI-99/4) on the University Module board, you'll soon discover that *it isn't there!* Instead, the University Module uses the TMS9980A, a 40-pin DIP package that is instruction-set compatible (69 basic op codes) with the 990 and 9900 family of minicomputers and microprocessors manufactured by Texas Instruments. It has a 16-bit CPU, but has a convenient 8-bit data bus (it doesn't have to be converted to 8-bits as does the TMS9900 for use in the TI-99/4), and an on-chip clock. The chip has been designed to minimize the system cost for smaller applications.

The board also comes with 1024 bytes of RAM (expandable on board to 2048 bytes), 4096 bytes of ROM (expandable on board to 6144 bytes) containing a monitor and symbolic assembler, a 16-bit programmable I/O port and interrupt monitor (parallel TMS9901 chips), a 45-key alphanumeric keyboard, a ten-digit, seven-segment L.E.D. alphanumeric display, plus a pair of built-in programmable audiovisual peripherals (four L.E.D. indicators and a sound disc).

Besides the board, the University Module comes with a 300-page *TM 990/189 Microcomputer User's Guide* which provides all the circuit diagrams and schematics, timing charts, explanations of theory, interfacing options, troubleshooting suggestions, and operational data that any user would want to know. The book is well organized and especially thorough. But the board and *User's Guide* alone, "do not a University Module make . . . " Purchasers also receive the 570-page *Introduction to Microprocessors*—a tutorial text especially written to satisfy textbook requirements at universities, colleges, and technical institutes for a three credit-hour course. The only other thing needed to get started is a power supply. Module users can provide their own, or purchase the TM 990/519 which connects to the University board with convenient snap-on connectors.



A word about the tutorial text, *Introduction to Microprocessors*: Before receiving my copy, I was somewhat skeptical about the utility of a "college textbook" for a course that was supposedly capable of being "self-taught." Traditional textbooks (more often than I'd care to remember) have frequently needed willing professors—standing ready in the wings—to "de-mystify" them. I was, however, pleasantly surprised by this one. Evidently, the three-year historical development of the course material (in industrial seminars and an SMU School of Engineering course) with ample opportunity for student feedback and suggestions, had a lot to do with this easy-to-understand text that finally emerged. The material is presented in an orderly, progressive manner—perfectly suitable for self-paced free-time study. Only a minimal background in digital logic and computer operation is necessary. Students are taken on an exciting tour of microprocessor architecture, arithmetic logic, computer addressing and program development, assembly language, memory systems, input/output concepts and design, modular programming, software engineering, and product development.

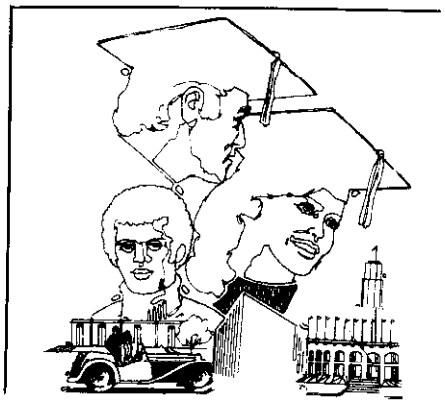
In addition to the on-board RAM expansion kit, there is an I/O expansion kit that provides the board with (1) an asynchronous communications controller with interface circuits for terminals that are either RS232C-compatible or use 20 mA current loops; (2) an on-board relay for the audio cassette interface that can control the tape recorder's motor drive; and (3) a bus expansion interface through which you can address up to 8192 bytes of off-board memory, and expand the CRU's (the communication register unit of the microprocessor) capability of accepting input from off-board devices. With the installation of this I/O kit—a very simple procedure, with the sockets and removable jumpers already on-board—you can effectively transform this microprocessor trainer into a development tool and real-world process control device capable of doing many kinds of sensing and control with only minor additional circuitry.

This brings us to the subject of programming languages. As mentioned earlier, the 4096-byte ROM contains a monitor and symbolic assembler. A user can program directly in TMS9900 assembly language through the 45-key, calculator-type alphanumeric keyboard (or with an external CRT or printer terminal if the I/O kit has been installed). Even though there are only 45 keys, a shift function enables practically the full ASCII character and control set to be represented (except for lowercase and the ampersand (&) character). The UNIBUG monitor allows us to converse with the system, so we can write better programs, and effectively debug them. UNIBUG gives us 16 commands and 7 I/O utilities —things like inspect/change functions for the CRU, status register, memory, workspace register, workspace pointer, etc.

After a finished program runs correctly in the on-board RAM, you can have it burned into EPROM (using a separate EPROM programmer), and plug it into the empty on-board EPROM socket. Although 2K of RAM or ROM doesn't seem like a lot of memory to a BASIC programmer with a 16K machine, remember that assembly is a low-level language that allows a programmer to communicate much more directly with the microcomputer through very compact instructions. And don't forget that the TM 990/189 also has the capability to address one block of off-board 8K RAM.

The board was obviously designed with flexibility in mind. Write your own "exotic" language and operating system in assembly, burn it into one or two EPROMs, snap it into the board's 4K and 2K sockets, and you're off and running with your own dedicated machine! Or, if you prefer a high-level language, you can buy TI's University BASIC (in 6K of PROM). It has all the "software hooks" in it for interfacing with the TMS9918 video display processor chip (the same one used in the TI-99/4) to create sprites on a color TV or monitor. Note: TI also sells the TM990/189-1 Packaged University Module. It has the expansion RAM and I/O kits installed, and comes with the BASIC in PROM. Purchasers of this configuration will have to buy separately the 4K ROM containing the UNIBUG monitor and symbolic assembler to program in assembly language.

In the next issue, we will explore the world of 16-bit assembly language programming—with its powerful instruction set, hardware multiply and divide, vectored interrupts, single-bit I/O manipulation, parallel I/O, and seven addressing modes. And for readers with TI-99/4s, I'll show you how to interface the two, and CALL your assembly subroutines through Extended BASIC to have them run on the larger machine.



Involving Preservice Teachers With Microcomputers

By Gary G. Bitter, Ph.D.

Effectively implementing microcomputers into the elementary and secondary schools is very much dependent on teacher preparation. Recognizing this, Arizona State University (one of the largest teacher degree granting institutions in the U.S.) has started a program to involve preservice teachers in microcomputer awareness activities. The first of these activities (for undergraduate elementary education majors in their mathematics methods class) is designed to familiarize the students with the College of Education's microcomputer laboratory. Since they are expected to complete required tasks in the class, the microcomputer assignment was developed as part of the course task requirements.

The assignment had the students (1) operate a microcomputer to run a prepared software program, and (2) enter a short BASIC program into the microcomputer. On the Texas Instruments TI-99/4 microcomputer, the students used *Number Magic* for the first task. This pre-programmed, snap-in software module is designed for addition, substraction, multiplication, and division drills for ages six years and up. Handouts provide detailed instructions for operating the TI-99/4 and information on the *Number Magic* program including a description, objectives, background, sample runs, a lesson plan, and a student work sheet. Figure 1 contains parts of the task.

Part two of the task was to enter a BASIC program into a microcomputer. For the TI-99/4, we provided a simple entry-type program (printing the student's name and performing a computation), plus detailed instructions for accessing the computer (Figure 2). Students were required to report the output they obtained from the microcomputer, and answer several questions pertaining to the activity (Figure 3).

College of Education/B225 Payne
Arizona State University
Tempe, AZ 85281

The Task

(A) DESCRIPTION . . .

A game of varying difficulty, Number Magic provides enrichment, drill, and practice in basic mathematics. The student chooses the method of computation, the computer will present the problem on the screen, pause for student answer and respond to the answer. The computer will total the players score.

(B) OBJECTIVES . . .

1. **To build skill and speed in problem solving.**
2. **To strengthen recall of basic mathematic equations.**
3. **To develop strategies in working with certain number pairs.**
4. **To make the drill of mathematics enjoyable to the user.**

(C) BACKGROUND INFORMATION

There are 3 activities in the Number Magic Module: Quick Quiz, Comp Quiz, and Electroflash. For "Comp Quiz" the player will be asked to either let the computer present ten problems randomly, or use the Memory Bank feature to store up to ten problem to answer . . . If the player chooses to let the computer give the problems, he or she can then choose the level of difficulty (easiest, simple, harder and hardest) and the kind of problem to be given (+, −, x, or ÷). In addition, the player has the option of supplying the missing number or missing answer in the problems. Timer (to race against the clock) or no time limit can also be chosen when playing. For the Memory Bank feature, the computer will assist the player in filling the memory bank. Once all choices have been made the computer asks whether the player is ready to start. The player presses ENTER and the computer begins to present the problems.

If the player is racing against the clock, a timer will be seen in the lower left corner. It begins at ten seconds and counts down. The time it takes to answer the problem correctly is the player's score for that problem. When incorrect answers are entered on the first or second attempt, the computer gives a "TRY AGAIN" message. On the third incorrect answer, the computer presents the correct answer and no score is given for that problem.

If the player does not use the timer, the computer gives ten points when the answer is correct on the first attempt. On the second and the third attempts, seven points and four points are given respectively. After three tries, the computer will provide the correct answer with a "NO SCORE" message. On the first two incorrect answers, the computer gives a "TRY AGAIN" message.

The computer keeps the total score and presents reinforcement at the end of the program. The computer then gives the player the chance to keep playing, select a new activity, replay the quiz, or stop the program.

Figure 1

The Task

(D) LESSON PLAN . . .

Present Number Magic as a fun way to drill and practice using varying number sizes and functions. By introducing two opposite presentational levels, the children will be able to see an indication of the computer's capacity within the game. At the start, divide the children into two teams. A running score can be kept on the blackboard. Allow captains of each team to also keep the score. (With younger children talley marks can be used.) Have one team decide whether to use the time limit; the next team can decide whether to fill in the answer or missing number, and so on, until each area has been chosen. As the computer presents each problem, a different member from each team will compete to see who can find the missing answer first. The first team member who answers the problem correctly gets the score associated with that problem. The game can be played until each member gets an opportunity to participate. Teams record their scores on the worksheet.

Sample Number Magic Worksheet Entry :

NAME:_____

Against the clock: yes or no
Filling in the: answer or missing number
Select the problem: computer or memory bank
Level of difficulty: 1 2 3 4 5
Method of calculation: + − x ÷
My score : _____

Figure 1 (cont.)

EED 380 TEACHING MATH IN THE ELEMENTARY SCHOOL

Assignment - Microcomputer - Programming
Microcomputers - Any

1. Follow the directions of the microcomputer called "Programming in BASIC."
2. Type in this program (if you make a mistake type Return or Enter and retype the line).

```
 5   PRINT "YOUR NAME" ENTER
10   LET A=3 ENTER
20   LET B=4 ENTER
30   LET C=A*B ENTER
40   PRINT "THE PRODUCT OF A AND B IS ";C ENTER
50   END ENTER
```

(If an error, retype the line correctly and again type RUN ENTER.)

3. Type RUN ENTER
4. On the back of this page write down the computer's output after you typed RUN.
5. When done, turn off microcomputer and monitor (CRT or TV).

Figure 2

Complete the Following Questions

What problems did you have? _____
Were you successful? _____
Which machine did you use? _____
How can this activity be improved? _____

Figure 3

Over 100 students were involved with the *Number Magic* module in the 1980 fall semester. Ninety percent indicated having a successful experience. For most students this was their first experience with a microcomputer.

Although this was a limited microcomputer experience, students nevertheless became aware of the role of microcomputers in education. In fact, some students have decided to continue their study of microcomputers by spending their free time in the microcomputer laboratory.

The task will continue to be used and upgraded each semester. Since the NCTM recommendations for the 80's include the statement, "Mathematics programs [should] take full advantage of the power of calculators and computers at all grade levels," microcomputer involvement for preservice teachers is obviously necessary. As one student commented, "I am sure I will be teaching with a microcomputer in the future. I am glad that I had an early awareness so I can adequately start getting prepared for its use in my classroom."

Santa Paravia and Fiumaccio

By W.K. Balthrop

You have heard it said many, many times before: "Power corrupts . . . " Do you believe it? Sure, why not? It's only an expression of human nature, isn't it? But now let me pose the question in a slightly different way: If the power was placed in *your* hands, would it corrupt *you*? Or more precisely: If you were given the reins of absolute command, would you become an amoral and calculating tyrant, or a forceful-yet-benevolent ruler—an iron fist in a velvet glove? There's only one way to find out. Let us give you that power by taking a trip back in time to 15th century Italy. And what better place is there for testing your abilities than in those turbulent, Machiavellian times when many ambitious rulers of small city-states vied for consolidation of power in hopes of eventually acquiring enough to become king. This time trip (or more literally, "power trip") capability is provided by the people at Instant Software—the producers of *Santa Paravia and Fiumaccio*, a computer simulation program for your TI-99/4.

Operation

Santa Paravia and Fiumaccio is an interractive, non-graphic simulation for one to six players competing at one of four skill levels. Through clever economic and political management, each player attempts to amass enough points to become the first to build his or her city-state into a kingdom—before death intervenes. (Life was short in those days . . .) Players take turns ruling their individual cities. Each turn consists of four parts: (1) Harvest Phase—when labor, planting, feeding, warehousing, and buy/sell decisions must be made; (2) Tax and Justice Phase—allowing rulers to change tax rates and level of justice; (3) Public Works Phase—providing opportunities to invest in markets, palaces, cathedrals, the wool industry, and equipping a portion of available serfs as soldiers; and (4) Display Phase—helping rulers plan strategy by displaying the present status of each kingdom-to-be.

Documentation and Packaging

The four-page typeset program documentation is very straightforward and easy to understand. Well-organized instructions allow even a first-time player to understand what is happening throughout his or her turn. Strategy tips are liberally sprinkled throughout, and are indeed more than welcome in a game with as many decisions to be made as this one.

The cassette and documentation are packaged in a small, two-color cardboard box. Along with it comes Instant Software's policy of accidental erasing: They'll redo your cassette for a nominal service charge of $2. This is one common-sense practice I'd definitely like to see more of in the software industry.

Performance and Engrossment

The first time I played *Santa Paravia*, I was immediately impressed with the detail for accuracy and provision for realistic interaction that were built into the program. The game is well organized, and allows players to concentrate on strategy, rather than having to keep track of a superabundance of rules (that other games of this type frequently employ). Strategy can, in fact, become quite complicated, but the program thoughtfully provides each player with the ability to experiment with different decisions and compare outcomes. For example, after making the decision to increase one of the tax rates, the menu screen will display the updated figures; a player can then go back and change the decision if it turned out to cause unfavorable results. This nice feature prevents one bad, irretrievable move from "ruining the game." We'd all be better off if present-day politicians had this option!

I've never had occassion to become bored when running this simulation. And, although it's not the type of program fraught with engrossing visual action, *Santa Paravia* will keep you constantly thinking and wondering "What if . . . "

Santa Paravia and Fiumaccio
TI BASIC; 16K
$9.95 (Cassette)
Instant Software
Peterborough, NH 03458

advoCAIte Course Authoring Guide :

A Preliminary Look

By Marilyn Latham
Gary G. Bitter, Ph.D.

College of Education/B225 Payne
Arizona State University
Tempe, AZ 85281

A new alternative now exists for educators without computer programming skills who, nevertheless, want to produce CAI (Computer Assisted Instruction) course material. **eduCAItor, inc.** of Hudson, Massachusetts, offers a system of programs on 5¼" floppy diskettes designed to direct the teacher through a series of instructions for creating his or her own software. This step-by-step procedure, the "advoCAIte Course Authoring Guide," allows a teacher to use music, color, and graphics (eventually, animated graphics), in developing curriculum.

It appears that the advoCAIte system will find a comfortable niche in the educational community. Many schools are discovering that CAI courseware available from computer dealers and software clearinghouses is not suitable for their particular needs. The fear that many teachers have for "complicated computer courses" is another factor contributing to the delay in getting CAI into the classroom. **advoCAIte** addresses itself to both of these concerns.

The advoCAIte system consists of Texas Instruments TI-99/4 personal computer, color monitor, disk controller, three disk drives, one Disk Manager Command Module, a three-ring binder (of approximately 100 pages), and seven floppy disks containing the programs and file space.

One of the programs automatically formats the material that will appear in the finished courseware. The five choices consist of (1) Text, (2) True-False, (3) Multiple Choice, (4) Match, and (5) Completion. Course authors can also choose from 185 graphics patterns stored in the "Permanent Graphics Dictionary," and use them in the artwork design for each frame. Original drawings may be added to the existing designs. Music may also be added to the frames by exchanging music notation for numbers represented on each line of the treble and bass clefs. The numbers are assigned to note values for establishing the sound, chords, and duration of tones. The result: A knowledge of music theory is not required for writing songs in this courseware!

The most difficult part of working with advoCAIte is learning how to properly use the disk drives. It does take some time. Furthermore, even with this tool, the amount of time required to actually produce finished courseware is something to be considered. The first-time user should be prepared to spend many hours reviewing the instructions and setting up the sequence of screen numbers.

Since the finished, "customized" programs can be saved and transported on floppy disks, the large amount of development time can be recovered, in part, through curriculum exchanges with other schools or institutions. The possibilities of such applications are exciting.

Watch for an in-depth review in the July/August issue of 99'er Magazine.

Personal Computing To Aid The Handicapped

The first national search for ideas and inventions through which the full spectrum of personal computing technology can be harnessed to assist the handicapped has been announced by The Johns Hopkins University.

The effort will be highlighted by a national competition for ideas, devices, methods, and computer programs to help handicapped people overcome difficulties in learning, working, and successfully adapting to home and community settings. Categories that may be addressed include computer-based aids for the blind, deaf and mentally retarded; for individuals with learning disabilities, neurological or neuromuscular conditions; and the orthopedically handicapped.

One hundred awards will be made, including a $10,000 grand prize, personal computer equipment, other cash prizes, computer training, and certificates of merit. Entries will be sought from computer specialists, full-time high school and college students, and from interested people generally, including those with handicaps. Regional and national awards will be made in all categories.

Paul L. Hazan, director of the Personal Computing to Aid the Handicapped project, said the competition is a challenge to the American people to use their conceptual skills in bringing forth practical aids based on computer technology that will help an individual or group of people with a handicap. "Just as important will be the opportunity provided the inventors and developers to make contact and form partnerships with the handicapped in a way that can lead to wide acceptance and use of the new computing technology" Hazan stressed.

Contestants will have until June 30, 1981 to prepare and submit their entries. To obtain additional information including a descriptive flyer and contest application, write to Personal Computing to Aid the Handicapped, Johns Hopkins University, P.O. Box 670, Laurel, Maryland 20810.

Spelling Flash

By Regena

"Spelling Flash" was designed to help students review their weekly spelling lists. This program does not use the speech synthesizer.

First the spelling words need to be typed in with DATA statements, starting with statement number 380. The words will be read as string variables. They may be entered as separate statements for each word, or several words may be listed separated by commas in each DATA statement. Words in this context may, of course, also consist of phrases or names with imbedded spaces or other special characters. Such phrases must be enclosed in quotes. Typing in ZZZ after the last word will prevent a data error message after the word list is used up.

The screen is cleared and a spelling word flashed on the screen. After a short delay the word is cleared and the student is asked to type in the spelling word. The end of the word is signalled by pressing the ENTER key. Positive reinforcement is given with some sounds and the statement, "YOU SPELLED IT RIGHT !!" If the word is incorrectly spelled the student must try again until it is correct.

The screen is cleared and the next word is flashed. The program will go through the list of words once. The program may be saved on cassette tape for use each day by the student. Each week the data statements may be changed for the new list of spelling words.

If the delay is too long or too short, the number "800" in Statement 340 may be adjusted.

```
100 REM   ****************
110 REM   *              *
120 REM   *SPELLING FLASH*
130 REM   *              *
140 REM   ****************
150 REM
160 REM   BY REGENA
170 REM   99'ER VERSION 5.81.1
180 CALL CLEAR
190 PRINT "SPELLING FLASH"::::
200 READ WORD$
210 IF WORD$="ZZZ"THEN 410
220 PRINT "SPELL ";WORD$
230 GOSUB 340
240 PRINT "SPELLING WORD:"
250 INPUT W$
260 IF WORD$=W$ THEN 290
270 PRINT ::"SORRY-PLEASE TRY AGAIN"::
280 GOTO 220
290 CALL SOUND(500,-1,2)
300 CALL SOUND(500,-2,2)
310 PRINT ::"YOU SPELLED IT RIGHT!!"
320 GOSUB 340
330 GOTO 200
340 FOR DELAY=1 TO 800
350 NEXT DELAY
360 CALL CLEAR
370 RETURN
380 DATA TIGER,BEHAVE,BEGIN,"JOHN ADAMS"
390 DATA TOMORROW,OZONE,SPRIG,GOOSE
400 DATA CREAM,COVER,AROMA,ABATE,ZZZ
410 END
```

MINDSTORMS

Children, Computers and Powerful Ideas

By Seymour Papert

All of us, professionals as well as laymen, must consciously break the habits we bring to thinking about the computer. Computation is in its infancy. It is hard to think about computers of the future without projecting onto them the properties and the limitations of those we think we know today. And nowhere is this more true than in imagining how computers can enter the world of education. It is not true to say that the image of a child's relationship with a computer I shall develop here goes far beyond what is common in today's schools. My image does not go beyond: It goes in the opposite direction.

In many schools today, the phrase "computer-aided instruction" means making the computer teach the child. One might say the *computer is being used to program* the child. In my vision, *the child programs the computer* and, in doing so, both acquires a sense of mastery over a piece of the most modern and powerful technology and establishes an intimate contact with some of the deepest ideas from science, from mathematics, and from the art of intellectual model building.

I shall describe learning paths that have led hundreds of children to becoming quite sophisticated programmers. Once programming is seen in the proper perspective, there is nothing very surprising about the fact that this should happen. Programming a computer means nothing more or less than communicating to it in a language that it and the human user can both "understand." And learning languages is one of the things children do best. Every normal child learns to talk. Why then should a child not learn to "talk" to a computer?

In my vision, the child programs the computer and, in doing so, both acquires a sense of mastery over a piece of the most modern and powerful technology and establishes an intimate contact with some of the deepest ideas from science, from mathematics, and from the art of intellectual model building.

There are many reasons why someone might expect it to be difficult. For example, although babies learn to speak their native language with spectacular ease, most children have great difficulty learning foreign languages in schools and, indeed, often learn the written version of their own language none too successfully. Isn't learning a computer language more like the difficult process of learning a foreign written language than the easy one of learning to speak one's own language? And isn't the problem further compounded by all the difficulties most people encounter learning mathematics?

Two fundamental ideas run through this book. The first is that it is possible to design computers so that learning to communicate with them can be a natural process, more like learning French by living in France than like trying to learn it through the unnatural process of American foreign-language instruction in classrooms. Second, learning to communicate with a computer may change the way other learning takes place. The computer can be a mathematics-speaking and an alphabetic-speaking entity. We are learning how to make computers with which children love to communicate. When this communication occurs, children learn mathematics as a living language. Moreover, mathematical communication and alphabetic communication are thereby both transformed from the alien and therefore difficult things they are for most children into natural and therefore easy ones. The idea of "talking mathematics" to a computer can be generalized to a view of learning mathematics in "Mathland"; that is to say, in a context which is to learning mathematics what living in France is to learning French.

People often ask whether in the future children will program computers or become absorbed in pre-programmed activities. The answer must be that some children will do the one, some the other, some both and some neither. But which children, and most importantly, which social classes of children, will fall into each category will be influenced by the kind of computer activities and the kind of environments created around them.

As an example, we consider an activity which may not occur to most people when they think of computers and children: the use of a computer as a writing instrument. For me, writing means making a rough draft and refining it over a considerable period of time, My image of myself as a writer includes the expectation of an "unacceptable" first draft that will develop with successive editing into presentable form. But I would not be able to afford this image if I were a third grader. The physical act of writing would be slow and laborious. I would have no secretary. For most children rewriting a text is so laborious that the

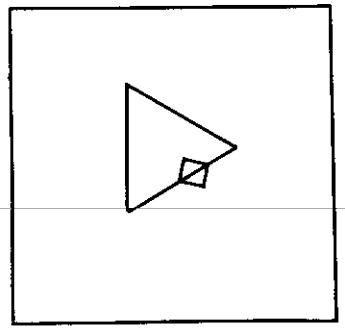


Figure 10c Bugged Man

ment we contrast his program with a different strategy of programming known as "structured programming." Our aim is to subdivide the program into natural parts so that we can debug programs for each part separately. In Keith's long, featureless set of instructions it is hard to see and trap a bug. By working with small parts, however, bugs can be confined and more easily trapped, figured out. In this case a natural subdivision is to make a program to draw a V-shaped entity to use for arms and legs and another to draw a square for the head. Once these "subprocedures" have been written and tested it is extremely easy to write the "superprocedure" to draw the stick figure itself. We can write an extremely simple program to draw the stick figure:

```
TO MAN
VEE
FORWARD 50
VEE
FORWARD 25
HEAD
END
```

This procedure is simple enough to grasp as a whole. But of course it achieves its simplicity only by making the assumption that the commands VEE and HEAD are understood by the computer. If they are not, the next step must be to define VEE and HEAD. We can do this in the same style of always working with a procedure we can understand as a whole. For example:

```
TO VEE
RIGHT 120
LINE 50
RIGHT 120
LINE 50
RIGHT 120
END
```

(In this program we assume we have defined the command LINE, which causes the Turtle to go forward and come back.)

To make this work we next define LINE:

```
TO LINE :DISTANCE
FORWARD :DISTANCE
BACK :DISTANCE
END
```

Since the last procedure uses only innate LOGO commands, it will work without further definitions. To complete MAN we define HEAD by:

```
TO HEAD
RIGHT 45
SQUARE 20
END
```

Keith, the author of the nonstructured MAN program, had been exposed to the idea of using subprocedures but had previously resisted it. The "straight-line" form of program corresponded more closely to his familiar ways of doing things. He had experienced no compelling need for structured programming until the day he could not debug his MAN program. In LOGO environments we have seen this happen time and again. When a child in this predicament asks what to do, it is usually sufficient to say: "You *know* what to do!" And often the child will say, sometimes triumphantly, sometimes sheepishly: "I guess I should turn it into subprocedures?" The "right way" was not imposed on Keith; the computer gave him enough flexibility and power so that his exploration could be genuine and his own.

If you've found these excerpts stimulating, you will definitely want to read the entire book. See the 99'er Bookstore section (p. 32-35) for price and ordering information.

LOGO is not a toy, but a POWERFUL COMPUTER LANGUAGE. . .

LOGO is the name of a philosophy of education in a growing family of computer languages that goes with it. Characteristic features of the LOGO family of lanuages include procedural definitions with local variables to permit recursion. Thus, in LOGO it is possible to define new commands and functions which then can be used exactly like primitive ones. LOGO is an interpretive language. This means that it can be used interactively. The modern LOGO systems have full list structure, that is to say, the language can operate on lists whose members can themselves be lists, lists of lists, and so forth.

Some versions have elements of parallel processing and of message passing in order to facilitate graphics programming. An example of a powerful use of list structure is the representation of LOGO procedures themselves as lists of lists so that LOGO is not a "toy," a language only for children. The examples of simple uses of LOGO in this book do however illustrate some ways in which LOGO is special in that it is designed to provide very early and easy entry routes into programming for beginners with no prior mathematical knowledge. The subset of LOGO containing Turtle commands, the most used "entry route" for beginners, is referred to in this book as "TURTLE TALK" to take account of the fact that other computer languages, for example SMALLTALK and PASCAL, have implemented Turtles on their systems using commands originally developed in the LOGO language. The TURTLE TALK subset of LOGO is easily transportable to other languages.

It should be carefully remembered that LOGO is never conceived as a final product or offered as "the definitive language." Here I present it as a sample to show that something better is possible.

Precisely because LOGO is not a toy, but a powerful computer language, it requires considerably larger memory than less powerful languages such as BASIC. This has meant that until recently LOGO was only to be implemented on relatively large computers. With the lowering cost of memory this situation is rapidly changing.

99'er magazine™ SUBSCRIPTION

☐ YES—Please sign me up as a subscriber. Enclosed is my payment or credit card billing information.

Term	U.S.A.	Canada & Mexico	Foreign Surface	Foreign Air
1-yr (6 issues)	☐ $15	☐ $18	☐ $25	☐ $40
2-yr (12 issues)	☐ $28	☐ $34		
3-yr (18 issues)	☐ $39	☐ $48		

Attention Dealers: Please write or call for bulk subscription rates. Also ask about our inexpensive magazine directory listings to publicize your retail locations. Call: (503)-485-8796

99'er BOOKSTORE

QTY	TITLE	PRICE	TOTAL AMOUNT

No. C.O.D. orders accepted. Add $1.50 postage & handling for 1 book, $2.00 for 2 books, or $2.50 for 3 or more books.

POSTAGE

TOTAL

99'er-ware™

*U.S. ONLY—FOR SHIPMENT OUTSIDE THE U.S. INQUIRE SEPARATELY

QTY	ITEM	PRICE	TOTAL AMOUNT
	TI-SETTE ADAPTOR	$3.50	
	TOTAL SHIPPING AND HANDLING		.75*
	DIGITAL COMPUTER CASSETTES in LIBRARY ALBUM	$9.95 or 3 for $27.00	
	TOTAL SHIPPING AND HANDLING		$2.00*
	MINI-FLEX DISKETTE FILE	$24.95	
	TOTAL SHIPPING AND HANDLING		$2.50*
	TOTAL		

Address shown is: ☐ Business ☐ Home

NAME _____ PLEASE PRINT

ADDRESS _____

CITY _____ STATE _____ ZIP _____

99'er MAGAZINE SUBSCRIPTION	
TOTAL 99'er BOOKSTORE	
TOTAL 99'er-ware	
TOTAL	

☐ Check enclosed MUST BE IN U.S. FUNDS DRAWN ON A U.S. BANK

Bill my:

☐ VISA ☐ Master Charge

Account No. _____

MC Only — List 4 digits above your name. _____

Expiration Date _____

Signature _____

PLACE STAMP HERE

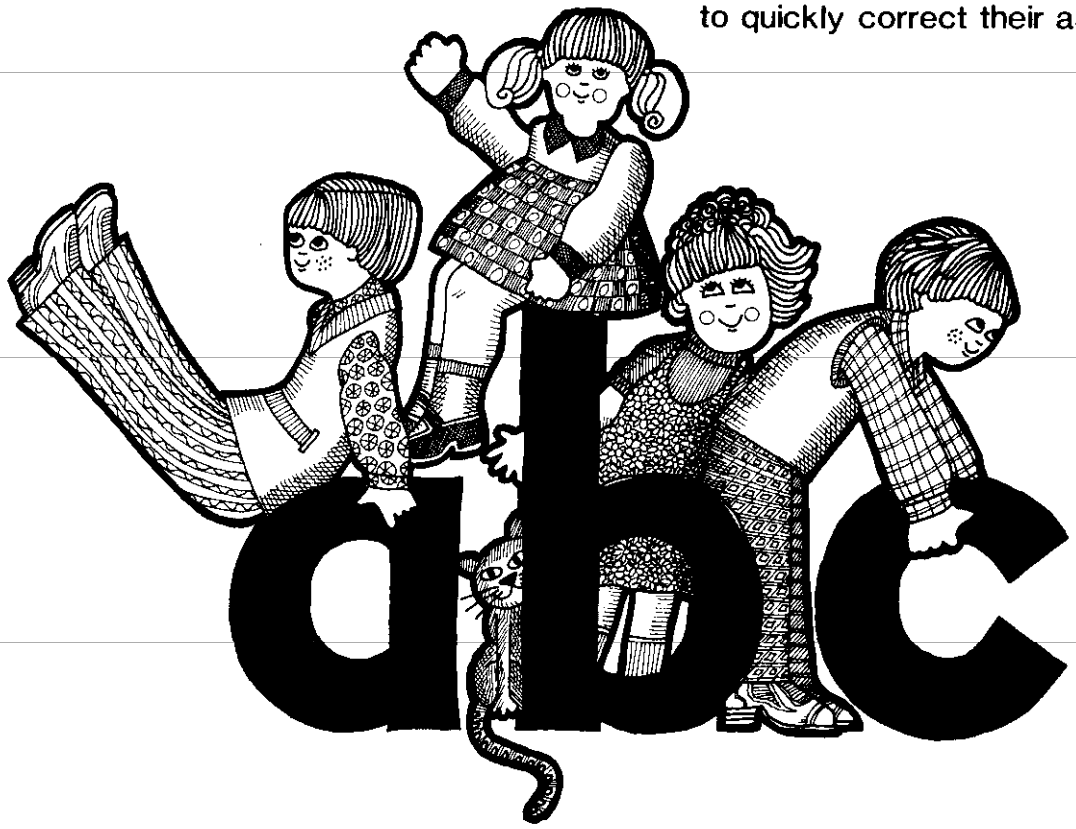
99'er magazine™

P.O. Box 5537

Eugene, OR 97405

Homework Helper:

Students do their class assignments on paper in the usual way . . . . and then can use the Homework Helper to quickly correct their assignments.



By Regena

FRACTIONS

The "Homework Helper" series is designed to quickly give answers to students checking their assignments. It is not meant to be a tutorial; it does not teach concepts nor quiz the student. Rather, it gives the answers to problems without showing all the intermediate steps.

The students are encouraged to do their class assignments on paper in the usual way, writing the problem down and working the problem step-by-step. Then, they can use the "Homework Helper" to quickly correct their assignments.

This program, involving fractions, is for correcting the homework problems of elementary school math students (4th, 5th, and 6th graders). Written in TI BASIC, it employs color graphics and sound, and is user interactive. There are seven sections—each introduced with a simple color representation of what that section is doing with fractions. Musical phrases from Mendelssohn, Handel, and Beethoven are also present.

1. Equivalence. Two fractions are of the form $\frac{A}{B} = \frac{C}{D}$. Any one of the four positions can be the unknown. The user designates the unknown, and inputs the three given values. The computer finds the unknown and prints the equivalent fractions. A student can also use this section to find equivalent ratios.

2. Simplification. The user inputs a numerator and a denominator. The computer simplifies (reduces) the fraction or tells if it cannot be simplified.

3. Multiplication. The user designates the number of fractions to be multiplied, then enters the numerator and denominator for each one. The computer multiplies them and simplifies the final fraction.

4. Division. Two fractions are entered; the first is then divided by the second, and the answer is simplified.

5. Addition—Like Denominators. The user specifies the number of fractions to be added, the common denominator, and then enters the numerators. The computer adds the numbers and simplifies the result.

6. Addition—Unlike Denominators. This section may be used to add fractions with like or unlike denominators. The user specifies the number of fractions up to 5 (which should be sufficient for elementary school mathematics), and then inputs the numerator and denominator of each. The computer adds the fractions and simplifies the results. A student can also use either Section 5 or 6 for subtraction problems by entering a negative numerator.

7. Comparisons. Up through ten fractions may be compared on a number line. The user enters the number of fractions to be compared (up to ten), and then enters the numerator and denominator of each. The computer then arranges the fractions from the smallest to the largest and prints them.

To stop any section of the program, press SHIFT C. To restart, enter RUN.

Efficiency Techniques

Simplifying Fractions

One basic technique of simplifying fractions is to start with the numerator as the first factor and see if it can be divided evenly into the denominator. If it can, both numerator and denominator are divided by that factor to immediately yield the simplified fraction. If the denominator cannot be evenly divided, the factor is reduced by one, and the numerator and denominator are tested to see if they are divisible by the new factor.

In each successive test, the factor is reduced by one. When both numerator and denominator can finally be evenly divided by the factor, that factor is the greatest common factor. The numerator and denominator are then divided by this factor to yield the reduced fraction.

For larger numbers, the technique can take a lot of time. In this program, the algorithm has been made more efficient by first checking to see which is smaller, the numerator or the denominator. In improper fractions the denominator will be smaller. The starting factor, PLIM, is set equal to the smaller number (Statements 1380 to 1410).

Another efficiency technique is to eliminate testing all even factors if either numerator or denominator is an odd number. This technique cuts the search time in half. In Statements 1420 to 1450 the step size, S, is set equal to −2 if either the numerator or the denominator is odd; S is set equal to −1 if both numerator and denominator are even numbers.

The simplifying algorithm is implemented with a FOR-NEXT loop. The starting trial factor is reduced by the step size, S, to a lower limit of 2 in line 1460.

Within the loop, Statements 1460 to 1510 set A=NS/P (where NS is the numerator) and set B=DS/P (where DS is the denominator). Then they check to see if A=INT(A); if equal, then B=INT(B) is checked. If both statements are true, the simplified fraction is A/B. Otherwise, P is incremented by S, and the loop continues. If the lower limit is reached without finding a successful factor, the user is notified that the fraction cannot be simplified (Statements 1520-1540).

When combining several fractions in multiplication or addition, another efficiency technique sets the starting factor equal to the largest denominator of the original fractions (Statements 2250-2340). The common denominator may be much larger than the original denominators, but the largest factor will always be the largest original denominator.

Comparisons

The schoolroom technique for comparing fractions is to find the common denominator and then compare the adjusted numerators. This technique is far too slow, especially when comparing many fractions and/or fractions with large numbers. A very fast technique which achieves the same end result is to compute and compare the decimal equivalents of the fractions.

As the fractions are read in, the numerator NNN (I) is divided by the denominator DDD (I) and stored as a decimal fraction in two identical arrays, FRC (I) and FRD (I) (Statements 5170-5230). A standard sort routine sorts the first array FRC from the smallest to the largest. The subscripts are changed as the decimal fractions are arranged in order (Statements 5250-5330).

The first element of FRC is compared with each element of the second array, FRD. When a match is made, the subscript value J is used to retrieve the numerator and denominator of the corresponding fraction for printing. The process is repeated in order for each element in the FRC array (Statements 5340-5390).

Homework Helper: Fractions — Program Explanations

Line Nos.

160-170	Sets T and T2 for the time in the music statements.
180-250	Defines characters and colors in four different character sets for use in graphics.
260-390	Prints title screen, "HOMEWORK HELPER".
410-550	Prints "FRACTIONS" and blinks an outline of asterisks around it.
580-640	Prints the menu screen for the seven sections of the program.
670-730	The user presses a key to choose which of the 7 sections is wanted, and the computer branches to that selection.
740-890	Prints the screen for Equivalence.
900-960	Asks for the unknown, A, B, C, or D.
970-1190	Depending on which is the unknown, asks for the given values and calculates the unknown. If the unknown is not a whole number, it will be rounded to two decimal places.
1200-1230	Prints the equivalent fractions.
1240-1310	Asks if there is another problem or to stop. If "2" is pressed, the menu screen is returned.
1320-1350	Prints screen for Simplifying.
1360-1370	Asks for the fraction.
1460-1540	Simplifies and prints the result.
1550-1620	Continue or go to menu screen.
1630-1650	Prints screen for Multiplying.
1660-1710	Asks for the fractions.
1720-1770	Multiplies the fractions.
1780-1800	Prints the problem and the simplified answer.
1810-1880	Continue or go to menu screen.
1890-1930	Subroutine for printing the problem.
1940-2150	Subroutine for simplifying and printing.
2160-2360	Subroutine for sorting and simplifying. These three subroutines are used for simplifying and printing in other sections of the program also.
2370-2430	Prints screen for Dividing.
2440-2470	Asks for the two fractions.
2480-2490	Performs division.
2500-2540	Prints problem and simplified solution.
2550-2620	Continue or go to menu screen.
2630-2680	Prints screen for Adding with like denominators.
2690-2760	Asks for the fractions and adds the numerators.
2770-2820	Prints the problem and the simplified sum.
2830-2900	Continue or go to menu screen.
2910-2960	Prints screen for Adding with unlike denominators.
2970-3090	Asks for the fractions and calculates a common denominator.
3100-3150	Adds the adjusted numerators and prints the problem and the simplified result.
3160-3230	Continue or go to menu screen.
3240-4090	Sound subroutines—musical phrases.
4100-5020	Draws color graphics for each title screen.
5030-5120	Prints screen for Comparisons.
5130-5230	Asks for fractions and converts fractions to decimals.
5240-5330	Sorts fractions from the smallest to the largest.
5340-5390	Prints fractions in order.
5400-5470	Continue or go to menu screen.
5480-5670	Music and graphics for Comparisons.

To stop the program, press SHIFT C (BREAK). For the student's convenience, at the end of each problem he can choose to do another problem of the same type or go to the menu screen and do a problem of a different type.

continued

```
100 REM  **********************
110 REM  *                    *
120 REM  * HOMEWORK HELPERS:  *
130 REM  *      FRACTIONS     *
140 REM  *                    *
150 REM  **********************
160 REM  BY REGENA
170 REM  99'ER VERSION 5.81.1
180 REM
190 T=300
200 T2=150
210 CALL CHAR(128,"F")
220 CALL COLOR(13,7,7)
230 CALL CHAR(136,"F")
240 CALL COLOR(14,11,11)
250 CALL CHAR(144,"F")
260 CALL COLOR(15,5,5)
270 CALL CHAR(152,"F")
280 CALL COLOR(16,13,13)
290 CALL CLEAR
300 CALL COLOR(2,16,7)
310 DATA 72,79,77,69,87,79,82,75
320 RESTORE 310
330 FOR Y=9 TO 23 STEP 2
340 READ L
350 CALL HCHAR(7,Y,L)
360 NEXT Y
370 DATA 72,69,76,80,69,82
380 RESTORE 370
390 FOR Y=11 TO 21 STEP 2
400 READ L
410 CALL HCHAR(10,Y,L)
420 NEXT Y
430 GOSUB 3270
440 CALL HCHAR(14,10,42,13)
450 CALL VCHAR(15,22,42,3)
460 CALL VCHAR(15,10,42,3)
470 CALL HCHAR(18,10,42,13)
480 DATA 70,82,65,67,84,73,79,78,83
490 RESTORE 480
500 FOR Y=12 TO 20
510 READ L
520 CALL HCHAR(16,Y,L)
530 NEXT Y
540 FOR I=1 TO 30
550 CALL COLOR(2,7,16)
560 CALL COLOR(2,16,7)
570 NEXT I
580 GOSUB 3380
590 CALL COLOR(2,2,1)
600 CALL CLEAR
610 PRINT "1 EQUIVALENCE"
620 PRINT "2 SIMPLIFY"
630 PRINT "3 MULTIPLY"
640 PRINT "4 DIVIDE"
650 PRINT "5 ADD-LIKE DENOMINATORS"
660 PRINT "6 ADD-UNLIKE DENOMINATORS"
670 PRINT "7 COMPARISONS"::::
680 GOSUB 4890
690 GOSUB 3470
700 CALL KEY(0,CHOICE,STATUS)
710 IF STATUS<=0 THEN 700
720 IF CHOICE<49 THEN 700
730 IF CHOICE>55 THEN 700
740 CHOICE=CHOICE-48
750 CALL CLEAR
760 ON CHOICE GOTO 770,1350,1660,2400,2660,
    2940,5060
770 DATA 69,81,85,73,86,65,76,69,78,67,69
780 RESTORE 770
790 CALL CLEAR
800 FOR Y=11 TO 21
810 READ L
820 CALL HCHAR(9,Y,L)
830 NEXT Y
840 CALL CHAR(96,"000000FF")
850 GOSUB 4130
860 DATA 14,13,65,15,13,96,16,13,66,15,16,
    61,14,19,67,15,19,96,16,19,68
870 RESTORE 860
880 FOR I=1 TO 7
890 READ X,Y,L
900 CALL HCHAR(X,Y,L)
910 NEXT I
920 GOSUB 3550
930 INPUT "WHICH IS THE UNKNOWN?":X$
940 IF X$="A"THEN 1000
950 IF X$="B"THEN 1060
960 IF X$="C"THEN 1120
970 IF X$="D"THEN 1180
980 PRINT "CHOOSE A,B,C,OR D"
990 GOTO 930
1000 INPUT "ENTER B = ":B
1010 INPUT "ENTER C = ":C
1020 INPUT "ENTER D = ":D
1030 A=B*C/D+.005
1040 A=1E-2*(INT(A*1E2))
1050 GOTO 1230
1060 INPUT "ENTER A = ":A
1070 INPUT "ENTER C = ":C
1080 INPUT "ENTER D = ":D
1090 B=A*D/C+.005
1100 B=1E-2*(INT(B*1E2))
1110 GOTO 1230
1120 INPUT "ENTER A = ":A
1130 INPUT "ENTER B = ":B
1140 INPUT "ENTER D = ":D
1150 C=A*D/B+.005
1160 C=1E-2*(INT(C*1E2))
1170 GOTO 1230
1180 INPUT "ENTER A = ":A
1190 INPUT "ENTER B = ":B
1200 INPUT "ENTER C = ":C
1210 D=B*C/A+.005
1220 D=1E-2*(INT(D*1E2))
1230 CALL CLEAR
1240 PRINT "       ";A,C
1250 PRINT "      ___ = ___":::
1260 PRINT "       ";B,D:::
1270 PRINT "PRESS 1 FOR NEXT PROBLEM"
1280 PRINT "PRESS 2 TO STOP"
1290 CALL KEY(0,K,STATUS)
1300 IF STATUS<=0 THEN 1290
1310 CALL CLEAR
1320 IF K=49 THEN 770
1330 IF K=50 THEN 590
1340 GOTO 1290
1350 PRINT "** SIMPLIFYING FRACTIONS **"
1360 GOSUB 4230
1370 PRINT ::
1380 GOSUB 3650
1390 INPUT "NUMERATOR?":NS
1400 INPUT "DENOMINATOR?":DS
1410 IF DS>NS THEN 1440
1420 PLIM=DS
1430 GOTO 1450
1440 PLIM=NS
1450 S=-2
1460 IF DS=INT(DS/2)THEN 1480
1470 IF NS=INT(NS/2)THEN 1490
1480 S=-1
1490 FOR P=PLIM TO 2 STEP S
1500 A=NS/P
1510 IF A<>INT(A)THEN 1540
1520 B=DS/P
1530 IF B=INT(B)THEN 1570
1540 NEXT P
1550 PRINT :NS;"/";DS;" CANNOT BE SIMPLIFIED"::::
1560 GOTO 1580
1570 PRINT :NS;"/";DS;" = ";A;"/";B::::
1580 PRINT "PRESS 1 FOR NEXT PROBLEM"
1590 PRINT "PRESS 2 TO STOP"
1600 CALL KEY(0,K,STATUS)
1610 IF STATUS<=0 THEN 1600
1620 CALL CLEAR
1630 IF K=49 THEN 1350
1640 IF K=50 THEN 590
1650 GOTO 1600
1660 PRINT "** MULTIPLYING FRACTIONS **"::::
1670 GOSUB 4430
1680 GOSUB 3750
1690 INPUT "HOW MANY FRACTIONS? ":F
1700 FOR I=1 TO F
1710 PRINT "FRACTION";I
1720 INPUT "    NUMERATOR = ":NN(I)
1730 INPUT "    DENOMINATOR = ":DD(I)
1740 NEXT I
1750 TN=1
1760 TD=1
1770 FOR I=1 TO F
1780 TN=TN*NN(I)
1790 TD=TD*DD(I)
1800 NEXT I
1810 PRINT :::"** MULTIPLY **"::
1820 GOSUB 1920
1830 GOSUB 2190
1840 PRINT ::"PRESS 1 FOR NEXT PROBLEM"
1850 PRINT "PRESS 2 TO STOP"
1860 CALL KEY(0,K,STATUS)
1870 IF STATUS<=0 THEN 1860
1880 CALL CLEAR
1890 IF K=49 THEN 1660
1900 IF K=50 THEN 590
1910 GOTO 1860
1920 FOR I=1 TO F
1930 PRINT NN(I);"/";DD(I)
1940 NEXT I
1950 PRINT "________________"
1960 RETURN
1970 IF TN<TD THEN 2000
1980 PLIM=TD
1990 GOTO 2010
2000 PLIM=TN
2010 FOR P=PLIM TO 2 STEP -1
2020 A=TN/P
```

Fractions, continued . . .

```
2030 IF A<>INT(A)THEN 2060
2040 B=TD/P
2050 IF B=INT(B)THEN 2090
2060 NEXT P
2070 A=TN
2080 B=TD
2090 IF A>=B THEN 2120
2100 PRINT ::A;"/";B::::
2110 GOTO 2160
2120 IF B=1 THEN 2170
2130 C=INT(A/B)
2140 R=A-C*B
2150 PRINT C;"   ";R;"/";B::::
2160 RETURN
2170 PRINT A::::
2180 RETURN
2190 FOR I=1 TO F
2200 P=DD(I)
2210 A=TN/P
2220 IF A<>INT(A)THEN 2270
2230 B=TD/P
2240 IF B<>INT(B)THEN 2270
2250 TN=A
2260 TD=B
2270 NEXT I
2280 SW=0
2290 FOR I=1 TO F-1
2300 IF DD(I)<=DD(I+1)THEN 2350
2310 J=DD(I)
2320 DD(I)=DD(I+1)
2330 DD(I+1)=J
2340 SW=1
2350 NEXT I
2360 IF SW=1 THEN 2280
2370 PLIM=DD(F)
2380 GOSUB 2010
2390 RETURN
2400 PRINT "** DIVIDING FRACTIONS **":::
2410 GOSUB 4520
2420 PRINT "THE FIRST FRACTION IS"
2430 PRINT "DIVIDED BY THE"
2440 PRINT "SECOND FRACTION.":::
2450 PRINT "(N1/D1) / (N2/D2)"::
2460 GOSUB 3830
2470 INPUT "ENTER N1 = ":N1
2480 INPUT "ENTER D1 = ":D1
2490 INPUT "ENTER N2 = ":N2
2500 INPUT "ENTER D2 = ":D2
2510 TN=N1*D2
2520 TD=D1*N2
2530 PRINT :::N1;"/";D1
2540 PRINT "_____":::
2550 PRINT N2;"/";D2:::
2560 PRINT "EQUALS"::
2570 GOSUB 1970
2580 PRINT :"PRESS 1 FOR NEXT PROBLEM"
2590 PRINT "PRESS 2 TO STOP"
2600 CALL KEY(0,K,STATUS)
2610 IF STATUS<=0 THEN 2600
2620 CALL CLEAR
2630 IF K=49 THEN 2400
2640 IF K=50 THEN 590
2650 GOTO 2600
2660 PRINT "** ADDING FRACTIONS **"::
2670 GOSUB 4590
2680 PRINT "THIS SECTION ADDS"
2690 PRINT "FRACTIONS THAT ALL HAVE"
2700 PRINT "THE SAME DENOMINATOR.":::
2710 GOSUB 3910
2720 INPUT "HOW MANY FRACTIONS? ":F
2730 INPUT "WHAT IS THE DENOMINATOR?":TD
2740 PRINT "ENTER THE NUMERATORS"
2750 TN=0
2760 FOR I=1 TO F
2770 INPUT NN(I)
2780 TN=TN+NN(I)
2790 NEXT I
2800 FOR I=1 TO F
2810 DD(I)=TD
2820 NEXT I
2830 PRINT :::
2840 GOSUB 1920
2850 GOSUB 1970
2860 PRINT "PRESS 1 FOR NEXT PROBLEM"
2870 PRINT "PRESS 2 TO STOP"
2880 CALL KEY(0,K,STATUS)
2890 IF STATUS<=0 THEN 2880
2900 CALL CLEAR
2910 IF K=49 THEN 2660
2920 IF K=50 THEN 590
2930 GOTO 2880
2940 PRINT "** ADDING FRACTIONS **"
2950 GOSUB 4740
2960 PRINT ::"THIS SECTION ADDS"
2970 PRINT "FRACTIONS THAT MAY HAVE"
2980 PRINT "UNLIKE DENOMINATORS.":::
2990 GOSUB 4020
3000 INPUT "HOW MANY FRACTIONS? ":F
3010 IF F<=5 THEN 3040
3020 PRINT "SORRY, I CAN ONLY HANDLE
     UP TO 5"
3030 GOTO 3000
3040 TN=0
3050 TD=1
3060 FOR I=1 TO F
3070 PRINT "FRACTION";I
3080 INPUT "     NUMERATOR = ":NN(I)
3090 INPUT "     DENOMINATOR = ":DD(I)
3100 TD=TD*DD(I)
3110 NEXT I
3120 PRINT ::
3130 FOR I=1 TO F
3140 C=TD/DD(I)
3150 TN=TN+NN(I)*C
3160 NEXT I
3170 GOSUB 1920
3180 GOSUB 2190
3190 PRINT "PRESS 1 FOR NEXT PROBLEM"
3200 PRINT "PRESS 2 TO STOP"
3210 CALL KEY(0,K,STATUS)
3220 IF STATUS<=0 THEN 3210
3230 CALL CLEAR
3240 IF K=49 THEN 2940
3250 IF K=50 THEN 590
3260 GOTO 3210
3270 CALL SOUND(T,880,2,698,5)
3280 CALL SOUND(T,932,2,784,5)
3290 CALL SOUND(T,784,2,659,5)
3300 CALL SOUND(T,880,2,698,5)
3310 CALL SOUND(T,698,2,587,5)
3320 CALL SOUND(T2,784,2)
3330 CALL SOUND(T2,698,2)
3340 CALL SOUND(T2,659,2)
3350 CALL SOUND(T2,784,2)
3360 CALL SOUND(T,698,2,687,5)
3370 RETURN
3380 CALL SOUND(T,440,2)
3390 CALL SOUND(T2,466,2)
3400 CALL SOUND(T2,523,2)
3410 CALL SOUND(T2,587,2)
3420 CALL SOUND(T2,523,2)
3430 CALL SOUND(T,466,2)
3440 CALL SOUND(T,440,2)
3450 CALL SOUND(1000,392,2,330,5)
3460 RETURN
3470 CALL SOUND(T2,440,2)
3480 CALL SOUND(T2,466,2)
3490 CALL SOUND(T,523,2)
3500 CALL SOUND(T,440,2)
3510 CALL SOUND(T,587,2)
3520 CALL SOUND(T,784,2)
3530 CALL SOUND(500,659,2)
3540 RETURN
3550 CALL SOUND(T,698,2)
3560 CALL SOUND(T,932,2)
3570 CALL SOUND(T,784,2)
3580 CALL SOUND(T,880,2)
3590 CALL SOUND(T2,932,2)
3600 CALL SOUND(T2,880,2)
3610 CALL SOUND(T2,784,2)
3620 CALL SOUND(T2,880,2)
3630 CALL SOUND(500,698,2)
3640 RETURN
3650 CALL SOUND(T2,659,2)
3660 CALL SOUND(T2,587,2)
3670 CALL SOUND(T,523,2)
3680 CALL SOUND(T,440,2)
3690 CALL SOUND(T,698,2,440,5)
3700 CALL SOUND(T,784,2,587,5)
3710 CALL SOUND(T2,698,2,392,5)
3720 CALL SOUND(T2,659,2)
3730 CALL SOUND(1000,698,2,440,5)
3740 RETURN
3750 DATA 262,349,392,440,523,440,
     392,349,392
3760 RESTORE 3750
3770 FOR I=1 TO 9
3780 READ M
3790 CALL SOUND(T2,M,2)
3800 NEXT I
3810 CALL SOUND(500,440,2)
3820 RETURN
3830 CALL SOUND(600,262,10)
3840 CALL SOUND(600,311,7)
3850 CALL SOUND(450,392,4)
3860 CALL SOUND(150,349,4)
3870 CALL SOUND(300,311,6)
3880 CALL SOUND(300,294,8)
3890 CALL SOUND(500,262,10)
3900 RETURN
3910 CALL SOUND(T,523,2)
3920 CALL SOUND(T,440,2)
3930 CALL SOUND(T2,440,2)
3940 CALL SOUND(T2,494,2)
3950 CALL SOUND(T,523,2)
3960 CALL SOUND(T2,494,2)
3970 CALL SOUND(T2,523,2)
3980 CALL SOUND(T2,494,2)
3990 CALL SOUND(T2,392,2)
4000 CALL SOUND(1000,440,2)
4010 RETURN
4020 CALL SOUND(400,440,8)
4030 CALL SOUND(200,392,8)
4040 CALL SOUND(200,440,7)
4050 CALL SOUND(400,587,6)
4060 CALL SOUND(200,523,5)
4070 CALL SOUND(200,587,4)
4080 CALL SOUND(400,494,3)
4090 CALL SOUND(200,440,4)
4100 CALL SOUND(200,494,5)
4110 CALL SOUND(500,392,6)
4120 RETURN
4130 CALL HCHAR(12,4,128,3)
4140 CALL HCHAR(13,4,128,3)
4150 FOR Y=4 TO 6
4160 CALL VCHAR(14,Y,144,4)
4170 NEXT Y
4180 CALL VCHAR(11,27,136,3)
4190 CALL VCHAR(11,28,136,3)
4200 CALL VCHAR(14,27,152,6)
4210 CALL VCHAR(14,28,152,6)
4220 RETURN
4230 FOR X=10 TO 14 STEP 2
4240 FOR Y=9 TO 13 STEP 2
4250 CALL HCHAR(X,Y,144)
4260 CALL HCHAR(X,Y-1,136)
4270 NEXT Y
4280 NEXT X
4290 FOR X=11 TO 13 STEP 2
4300 FOR Y=8 TO 12 STEP 2
4310 CALL HCHAR(X,Y,144)
4320 CALL HCHAR(X,Y+1,136)
4330 NEXT Y
4340 NEXT X
4350 CALL HCHAR(12,16,61)
4360 FOR Y=19 TO 21
4370 CALL VCHAR(10,Y,136,5)
4380 NEXT Y
4390 FOR Y=22 TO 24
4400 CALL VCHAR(10,Y,144,5)
4410 NEXT Y
4420 RETURN
4430 Y=6
4440 FOR I=1 TO 5
4450 CALL VCHAR(10,Y,136,2)
4460 CALL VCHAR(10,Y+1,136,2)
4470 CALL VCHAR(12,Y,128,4)
4480 CALL VCHAR(12,Y+1,128,4)
4490 Y=Y+5
4500 NEXT I
4510 RETURN
4520 CALL HCHAR(10,11,136,13)
4530 CALL HCHAR(11,11,136,13)
4540 FOR X=12 TO 14
4550 CALL HCHAR(X,11,144,13)
4560 NEXT X
4570 CALL VCHAR(7,17,93,11)
4580 RETURN
4590 CALL HCHAR(10,8,128,2)
4600 CALL VCHAR(11,8,152,4)
```

Fractions, continued . . .

```
4610 CALL VCHAR(11,9,152,4)
4620 CALL VCHAR(10,13,128,4)
4630 CALL VCHAR(10,14,128,4)
4640 CALL HCHAR(14,13,152,2)
4650 CALL VCHAR(10,18,128,2)
4660 CALL VCHAR(10,19,128,2)
4670 CALL VCHAR(12,18,152,3)
4680 CALL VCHAR(12,19,152,3)
4690 CALL VCHAR(10,23,128,3)
4700 CALL VCHAR(10,24,128,3)
4710 CALL VCHAR(13,23,152,2)
4720 CALL VCHAR(13,24,152,2)
4730 RETURN
4740 CALL VCHAR(10,8,128,4)
4750 CALL VCHAR(14,8,136,3)
4760 CALL VCHAR(10,12,144,2)
4770 CALL VCHAR(10,13,144,2)
4780 CALL VCHAR(12,12,128,3)
4790 CALL VCHAR(12,13,128,3)
4800 CALL HCHAR(10,17,136,3)
4810 CALL HCHAR(11,17,152,3)
4820 CALL HCHAR(12,17,152,3)
4830 CALL HCHAR(13,17,152,3)
4840 CALL VCHAR(10,23,152,2)
4850 CALL VCHAR(10,24,152,2)
4860 CALL VCHAR(12,23,144,4)
4870 CALL VCHAR(12,24,144,4)
4880 RETURN
4890 CALL HCHAR(4,15,128,2)
4900 CALL HCHAR(5,14,128,3)
4910 CALL HCHAR(6,13,128,4)
4920 CALL HCHAR(7,13,128,4)
4930 CALL HCHAR(4,17,136,2)
4940 CALL HCHAR(5,17,136,3)
4950 CALL HCHAR(6,17,136,4)
4960 CALL HCHAR(7,17,136,4)
4970 CALL HCHAR(8,17,152,4)
4980 CALL HCHAR(9,17,152,4)
4990 CALL HCHAR(10,17,152,3)
5000 CALL HCHAR(11,17,152,2)
5010 CALL HCHAR(11,15,144,2)
5020 CALL HCHAR(10,14,144,3)
5030 CALL HCHAR(9,13,144,4)
5040 CALL HCHAR(8,13,144,4)
5050 RETURN
5060 DATA 67,79,77,80,65,82,73,83,79,
     78,83,32
5070 RESTORE 5060
5080 FOR Y=11 TO 22
5090 READ L
5100 CALL HCHAR(14,Y,L)
5110 NEXT Y
5120 GOSUB 5510
5130 DIM NNN(10),DDD(10),FRC(10),FRD(10)
5140 PRINT "THIS ARRANGES FRACTIONS"
5150 PRINT "FROM SMALLEST TO LARGEST."::
5160 INPUT "HOW MANY FRACTIONS?":NF
5170 IF NF<11 THEN 5200
5180 PRINT "SORRY; UP TO 10 ONLY."
5190 GOTO 5160
5200 FOR I=1 TO NF
5210 PRINT ::"FRACTION ";I
5220 INPUT "     NUMERATOR:":NNN(I)
5230 INPUT "     DENOMINATOR:":DDD(I)
5240 FRC(I)=NNN(I)/DDD(I)
5250 FRD(I)=FRC(I)
5260 NEXT I
5270 PRINT ::::
5280 SW=0
5290 FOR I=1 TO NF-1
5300 IF FRC(I)<=FRC(I+1)THEN 5350
5310 FF=FRC(I)
5320 FRC(I)=FRC(I+1)
5330 FRC(I+1)=FF
5340 SW=1
5350 NEXT I
5360 IF SW=1 THEN 5280
5370 FOR I=1 TO NF
5380 FOR J=1 TO NF
5390 IF FRC(I)=FRD(J)THEN 5410
5400 NEXT J
5410 PRINT I;"  ";NNN(J);"/";DDD(J)
5420 NEXT I
5430 PRINT :::"PRESS 1 FOR NEXT PROBLEM"
5440 PRINT "PRESS 2 TO STOP"
5450 CALL KEY(0,K,STATUS)
5460 IF STATUS<=0 THEN 5450
5470 CALL CLEAR
5480 IF K=49 THEN 5060
5490 IF K=50 THEN 590
5500 GOTO 5450
5510 CALL SOUND(400,330,2,262,5)
5520 CALL VCHAR(4,8,136,3)
5530 CALL VCHAR(4,9,136,3)
5540 CALL SOUND(100,330,2)
5550 CALL HCHAR(7,8,144,2)
5560 CALL SOUND(100,262,3)
5570 CALL SOUND(400,330,1)
5580 CALL VCHAR(4,15,128,4)
5590 CALL VCHAR(8,15,152,2)
5600 CALL SOUND(100,330,2)
5610 CALL SOUND(100,262,3)
5620 CALL SOUND(400,330,1)
5630 CALL VCHAR(4,23,152,3)
5640 CALL VCHAR(4,24,152,3)
5650 CALL SOUND(200,392,5)
5660 CALL VCHAR(7,23,136,3)
5670 CALL SOUND(200,524,3)
5680 CALL VCHAR(7,24,136,3)
5690 CALL SOUND(400,660,1)
5700 RETURN
```

Coming in the July/August Issue . . .

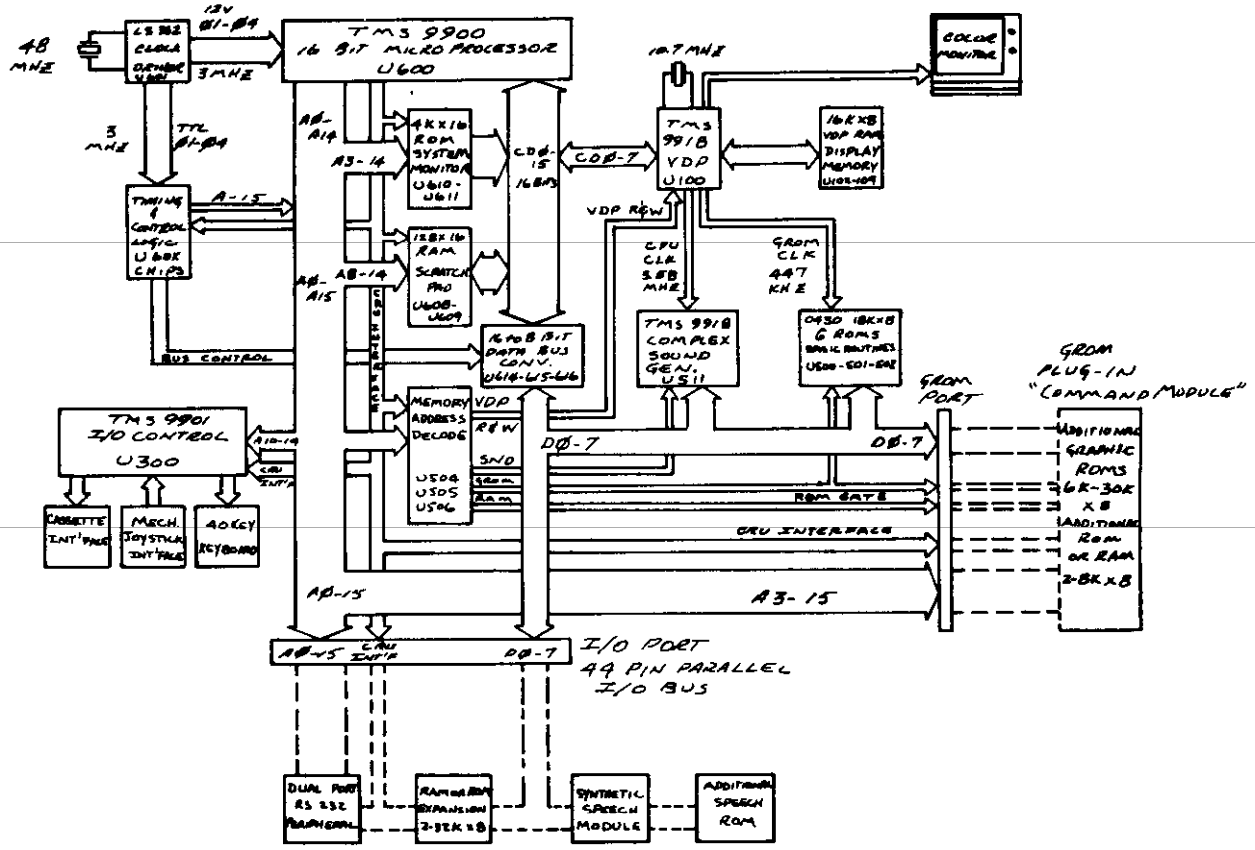
- Telecommunicating with the TI-99/4
- Micro Bartender
- CAI for Engineers
- Let's Learn Notes
- Typing Tutor
- Pre-Schoolers and the TI-99/4
- Software Language Conversions
- A Look at Bit Pads, CRTs & Modems
- Marinchip Systems' TMS9900 Boards & Super-Fast QBASIC

☆Plus All Our Regular Features

And Many Additional Surprises . . .

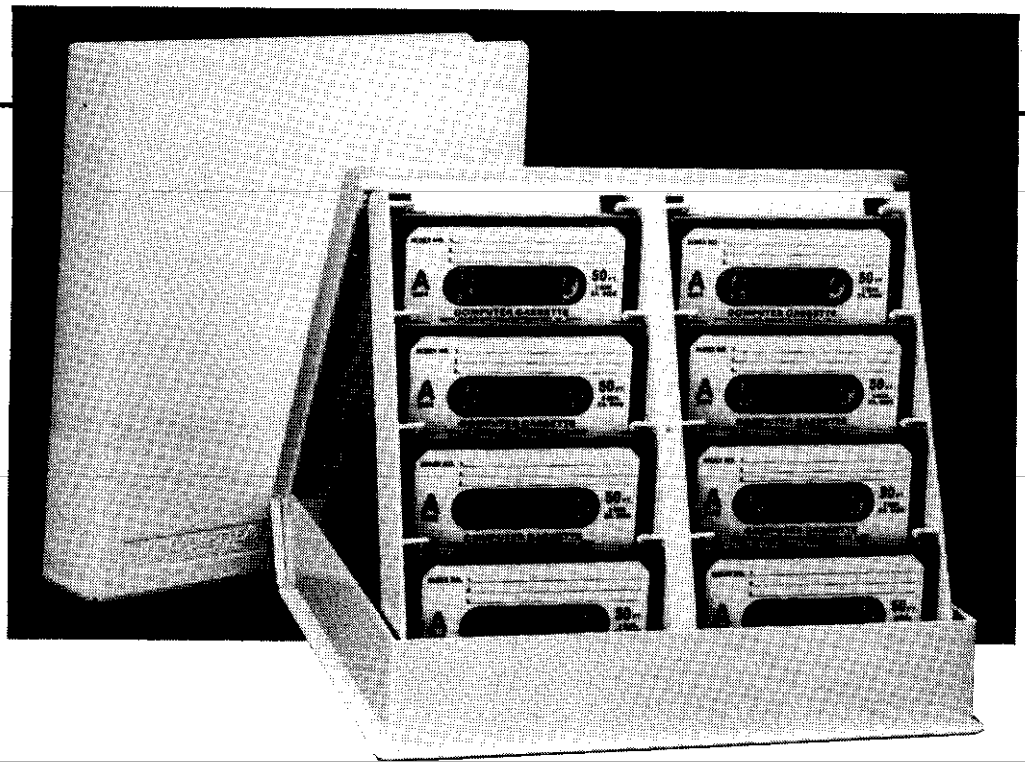
The Way It Works:

A Block Diagram of the TI-99/4



From *Texas Instruments Home Computer Technical Data*

Index to Advertisers



DIGITAL COMPUTER CASSETTES
with LIBRARY ALBUM

- High-Quality BASF Tape in a 5-Screw Cassette Housing for Data Integrity
- Secure Storage & Quick Convenient Retrieval with Protective File Album
- 8 Special 50-Foot Cassettes – Ideal for Building Up Your 99'er Program Library
- Economical As It Is Functional

☆ Only $9.95 each; or 3 for $27.00

Add $2.00 shipping and handling. Use the bind-in card in the back of the magazine for your convenience in ordering. Telephone orders accepted if charged to credit cards.

99er-ware™

INNOVATIVE PRODUCTS FOR TMS9900-BASED PERSONAL COMPUTING

P.O. Box 5537
Eugene, Oregon 97405
Tel. (503) 485-8796