The New Commodore Plus/4: A Hands-On Preview

COMPUTE!

$2.95
October
1984
Issue 53
Vol. 6, No. 10
$3.75 Canada
02193
ISSN 0194-347X

The Leading Magazine Of Home, Educational, And Recreational Computing

The Parser's Tale: How Adventure Games Work

Two Captivating Games:

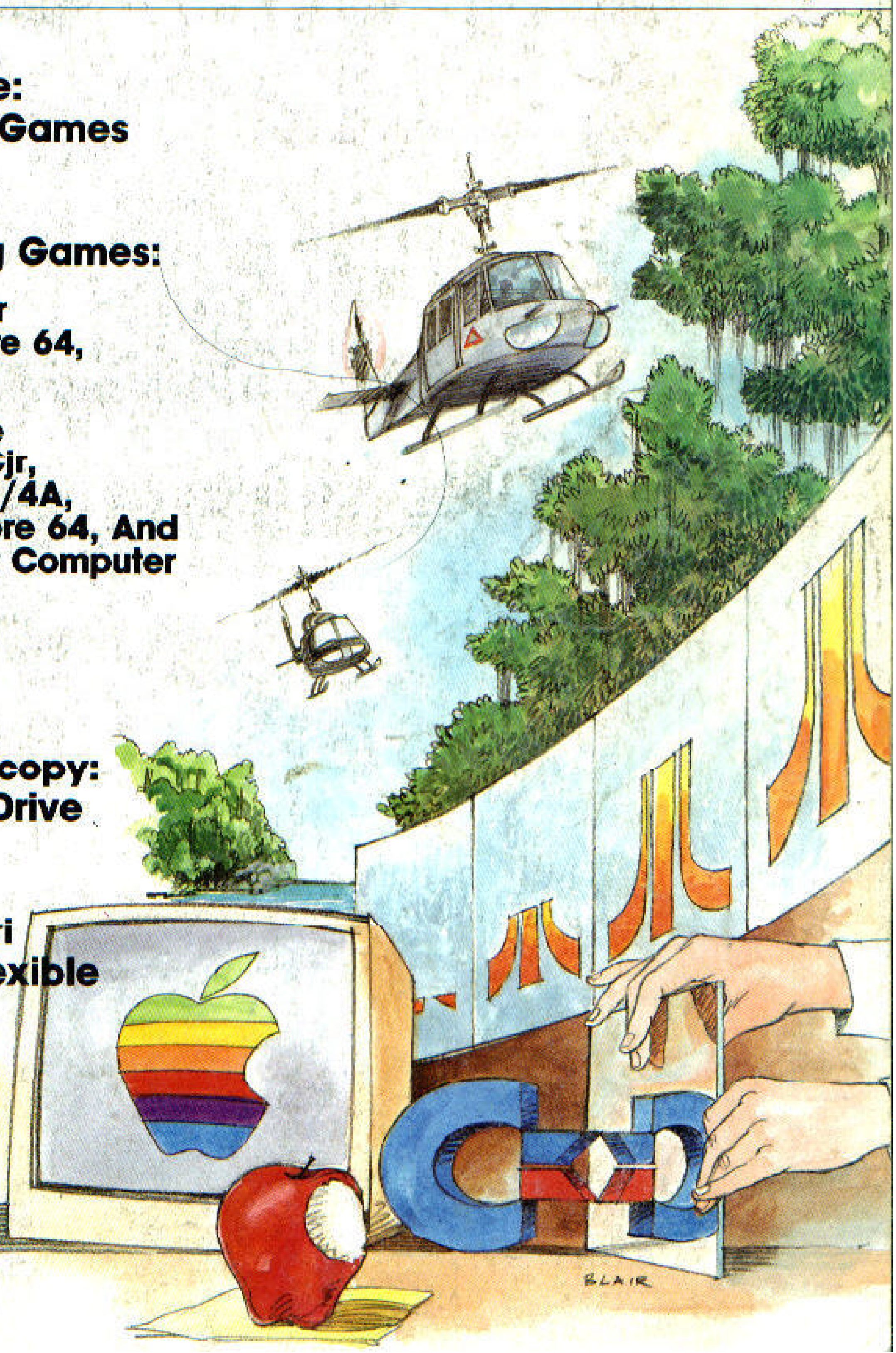
Canyon Runner For Apple, Commodore 64, VIC-20, And Atari

The Number Game For IBM PC And PCjr, Atari, Apple, TI-99/4A, VIC-20, Commodore 64, And Radio Shack Color Computer

Simple Apple Screen Dump

Commodore Unicopy: Back Up Single-Drive Disks Efficiently

Multiscreen Atari Animation: A Flexible Graphics Editor



0 71486 02193 10

October 1984 Vol. 6, No. 10

FEATURES

EDUCATION AND RECREATION

REVIEWS

COLUMNS AND DEPARTMENTS

THE JOURNAL

NOTE: See page 179 before typing in programs.

AP Apple **AT** Atari, **P** PET/CBM, **V** VIC-20, **C** Radio Shack Color Computer, **64** Commodore 64, **TS** Timex/Sinclair, **TI** Texas Instruments, **PCjr** IBM PCjr, **PC** IBM PC, **AD** Coleco Adam.
*All or several of the above.

COMPUTE! Publications, Inc.

One of the ABC Publishing Companies:
ABC Publishing, President, Robert G. Burton
1330 Avenue of the Americas, New York, New York 10019

COMPUTE! The Journal for Progressive Computing (USPS: 537250) is published monthly by COMPUTE! Publications, Inc., P.O. Box 5406, Greensboro, NC 27403 USA. Phone: (919) 275-9809. Editorial Offices are located at 324 West Wendover Avenue, Greensboro, NC 27408. Domestic Subscriptions: 12 issues, $21. Send subscription orders or change of address (P.O. form 3579) to **COMPUTE!** Magazine, P.O. Box 914, Farmingdale, NY 11737. Second class postage paid at Greensboro, NC 27403 and additional mailing offices. ISSN 0194-357X.

EDITOR'S NOTES

The Editor's Notes this month are written by Tom R. Halfhill, new editor of COMPUTE!.

Robert Lock
Editor In Chief,
COMPUTE! Publications

A New Beginning

Nearly two and a half years ago, Robert Lock hired me as features editor of COMPUTE!. At that time the editorial staff consisted of four full-time people. We occupied a few offices in an old building near downtown Greensboro, and the circulation of COMPUTE! was about 75,000.

Today COMPUTE! Publications has an editorial staff of more than 50 full-time people. Together with about 70 employees of other departments, we occupy an entire floor in a new office building, with warehouse and shipping facilities across town. COMPUTE! is approaching 400,000 circulation. Our second magazine, COMPUTE!'s GAZETTE, has gained more than 300,000 readers in just over a year of existence, and our Book Division consistently places titles on computer-book bestseller lists. In mid-1983, COMPUTE! Publications became a part of ABC Publishing, a subsidiary of the American Broadcasting Company.

Obviously, we have gone through a great many changes in the past two and a half years. Hundreds of other companies in the computer industry have experienced the same kind of phenomenal growth, of course. But now that the industry is maturing, the spectacular growth of those first few years is leveling off and is becoming more like the steady, sustainable growth common to other industries. Some companies which became accustomed to annual growth rates of 50 percent, 100 percent, or even more are suddenly finding themselves in trouble because they assumed the roller coaster would keep speeding forever. That's partly why some of those companies are cutting back, laying off, and even going out of business. In an industry where the market changes almost monthly, you have to be quick on your feet to survive.

At COMPUTE!, so far we've managed to keep pace with the changes. There have been plenty of growing pains which have demanded much from our staff, but we've always remained flexible and succeeded in pulling together.

My own path shows how fast things change around here. After less than a year as features editor, I was appointed founding editor of our second magazine, COMPUTE!'s GAZETTE. The first few months were a struggle, but with lots of hard work, together we built the GAZETTE into the most successful new magazine in the industry. Then, just as things started rolling along smoothly, I was assigned to another new project—COMPUTE!'s PC & PCjr magazine. The new IBM PCjr was arriving on the market and it seemed destined to become the success story of 1984.

As you probably know by now, things didn't quite work out that way. The PCjr didn't sell, so neither did our new magazine. We decided to stop publication with the October 1984 issue.

But that's not all bad. After more than a year's absence, I'll be returning full-time to our flagship magazine, COMPUTE!—this time as its new editor. Richard Mansfield, who has handled COMPUTE!'s daily duties for more than three years, will continue as senior editor of COMPUTE Publications, helping to supervise editorial operations for both our magazines and our Book Division.

And we have a number of improvements planned for COMPUTE! to strengthen its position as the leading magazine for home, educational, and recreational computing. For one thing, we'll be merging our IBM coverage into COMPUTE! to serve both our existing IBM subscribers and several thousands of new readers joining us next month from COMPUTE!'s PC & PCjr. More programs will be translated for the PC and PCjr, and there'll be some IBM reviews and stand-alone articles as well. We're also adding a new column next month, "IBM Personal Computing," by Donald B. Trivette.

Apple readers can expect more attention, too. With the introduction of the Apple IIc and Macintosh, plus heavily discounted prices on the Apple IIe, we've noticed a resurgence of interest in Apple coverage. More of our programs will be translated for the Apple, and we're beefing up coverage in other areas also.

If you use a Commodore, Atari, or TI, don't despair. You still make up the bulk of our readership and therefore deserve the most coverage in COMPUTE!. We won't let you down. If anything, we plan to strengthen our coverage of your computers.

You might be wondering how it's possible to increase coverage for everybody without taking something away from somebody. That's always a concern in a multimachine magazine. Our solution: We'll be reorganizing our regular columns, streamlining the articles, and taking great pains to make sure the articles and programs we publish continue to be of the highest possible quality.

For example, in coming issues you'll notice that some columns will be consolidated and new ones will be added. Programs will be translated to run on as many computers as possible. And we'll make a renewed commitment to minimize errors and publish the best computer magazine on the market.

You'll begin noticing these improvements within the next few issues—we're making them as fast as possible. That's the way things happen in the computer industry.

Erasing Cassettes

How can someone erase cassettes that already have programs on them?

Todd Butcher

Cassettes can be easily erased. The simplest way is to just record something new on top of the old information. Alternatively, if you want to erase a cassette entirely, you could put a computer cassette into an ordinary music tape player and, after turning the volume down, press the play and record buttons simultaneously. Another way to do it is to use a bulk eraser, a device that uses a strong magnetic field to erase tape. These devices are available at record and electronics stores.

Atari CLOAD Problems

I own an Atari 800 with an Atari cassette recorder. I have used my 800 to type in and save programs from COMPUTE! to cassette tape. All of these programs used to load and run properly, but recently I have not been able to CLOAD the same tapes into my computer. I have tried different Atari recorders and even my friend's computer, but have only succeeded in loading in one of the programs. When I attempt to CLOAD a program on my recorder, I get an error 138 or 143. What can I do?

James L. Jenkins

The problem you are experiencing is a very common one. Usually this happens when the Atari recorder has been in use for some time. The reason it happens is that the recorder head needs to be either cleaned or demagnetized, or both. There are several tricks that you can use to see if you might be having other problems. Try connecting the recorder directly to the computer instead of through another peripheral. If this clears up the problem, it could mean that the connection in your other peripheral (disk drive, printer, or expansion box) is soiled or loose.

You can also try completely rewinding the tape and then fast-forwarding it past the tape header. Set the tape counter to zero, and try CLOADing from there. If you still get an error, rewind again and this time try CLOADing from tape counter position 1. Keep doing this in one-step increments until the tape loads.

The last trick is to insert your computer tape into an audio cassette player and listen to it until you hear a screeching sound. Once you hear the sound, you are at the beginning of the program on the tape. Try to get as close as possible to the beginning without passing it, and then try CLOADing it on your Atari recorder. If this does not work, try demagnetizing and cleaning your recorder's head. This is an easy procedure and should be done regularly anyway. Kits are available at any record store.

Once you do manage to CLOAD your programs, you should consider LISTing them to tape instead of CSAVEing them. The advantage: LISTing, combined with the ENTER command, is a more reliable method of loading from cassette than CLOAD. The LIST command takes up more tape and is also slower, but that's a small price to pay for greater reliability.

Disk Density

I recently purchased a disk drive. The instructions specify that you should use single-density disks. However, I have some double-density disks which I would like to be able to use. Will it cause any problems?

James P. Simson

Double-density disks will not cause any problem. Using a product of higher quality than specified never hurts. However, using single-density disks on a drive that specifies double-density could cause difficulties.

Speech For VIC And Atari

I recently bought an Atari 800 and VIC-20 computer. I want to know if there is any way to generate speech on them without spending a small fortune on a speech synthesizer. If it is possible, please explain how.

Mel Barries

To our knowledge there is no easy way to program speech on the VIC or Atari. Usually, special add-ons are necessary to accomplish this task. The S.A.M. speech program works well, although the number of words you can use at any given time is limited. Many schemes have been invented to simulate speech through software, but all of them require extensive amounts of memory. One such scheme requires a microphone and a board for entering the sounds that you want the computer to mimic. Specially designed software takes volume readings thousands of times for each word, and records the readings in RAM memory. The speech software then uses these volume changes to simulate the sound through your computer's sound chip. COVOX manufactures a good implementation of this technique for the Commodore 64.

TI Peripheral Expansion Box

Could you please tell me what the peripheral expansion box is needed for and give me some advice on whether I should purchase one or not? I'm a little apprehensive about investing a lot of money in my TI to find out that no one is going to support it. Do you have any suggestions about this?

Todd M. Aube

Since Texas Instruments decided to discontinue the TI, third-party support for this computer is expected to decline. COMPUTE!, *however, will continue to support the TI with new software each month.*

If you have not bought an expansion box yet, you probably won't even be able to find one. Many people who bought TIs early on realized that they would need an expansion box, and consequently most stores have already sold out. The expansion box is required to use the TI peripherals, such as disk drives and printer.

Atari Hex-To-Decimal Conversion

The hex to decimal conversion program in "Readers' Feedback" in the July 1984 COMPUTE! by Frank Sgabellone is quite powerful. However, the modifications necessary to make it run on an Atari might not be too obvious. The following translation will work on Ataris. The value of C in line 20 can be changed, to vary the number of leading zeros.

H. Earl Hill

```
GN 10 DIM A$(16),C$(1)
JN 20 A$="0123456789ABCDEF":? "INPUT
      DEC/HEX (0-65535)":INPUT A:B=
      1:C=3:D=16^C:? A;" = $";:A=A+1
JP 30 IF A-D>1 THEN A=A-D:B=B+1:GOTO
      30
CA 35 J=1
JO 40 C$=A$(B,B+J-1):? C$;:B=1:C=C-1
      :D=16^C:IF C>-1 THEN 30
DG 50 ? "{5 SPACES}":? :GOTO 20
```

Apple Trigonometry

I have an Apple II+ on which I was hoping to be able to do some trigonometry homework. I was testing the SIN, TAN, and COS functions and discovered that when I provided a number within parentheses for these functions to evaluate, the number never matched a set of answers that I have in a chart. I looked up these functions in my Apple manual, and all it gave was an explanation of radians and other things I could not comprehend. Could you give me an understandable explanation of what these functions do?

Chuck Knakal

On computers such as the Apple, TI, Commodore, and many others, the trigonometric functions are always expressed in radians. *Radians are just another way to measure an angle. For example, instead of expressing an angle as 180 degrees, you would say it was one pi radians.*

A complete circle is 360 degrees. In radians, that would be exactly two pi radians (pi is approximately 3.1416). If what you are looking for, though, is an easy way to get answers in degrees from your computer, all you have to do is multiply the angle that you want evaluated by pi and divide that by 180. If you then input that number into the SIN, TAN, or COS functions of your computer, you should get the right answer in degrees.

For example, let's take 90 degrees. The sine of 90 degrees should give you an answer of 1, but since the computer does not work in degrees, PRINT SIN (90) will give you another answer. To get the answer in degrees, just take the 90 and multiply it by 3.1416, then divide the answer by 180. Now take the SIN of that answer and you should get 1. If your computer has a built-in key for pi, use that instead of the approximation because it will give more precise results. For example, on the Commodore VIC and 64, pressing SHIFT and the up-arrow (↑) key will print a pi symbol which can be used in expressions as a constant with the value of pi. On the Atari there's an even simpler way. You can use the DEG statement to switch all calculations to degrees.

64 Reverse Lines

I would like to display 40 reverse spaces per line. But the printed fortieth character causes a line to be skipped before the next line of text is printed. Therefore, I must leave the fortieth column unprinted to. How may I accomplish this feat in 64 BASIC without skipping a line?

Philip A. Egan

Try PRINTing 39 reverse characters on the screen per line, and then add a routine that will POKE reverse characters into the fortieth column of each line. The following should help on the 64:

```
90 FOR X= 1053 TO 2023 STEP 40: POKE X,160:
POKE X+54272,COLOR: NEXT
```

The variable COLOR can be any of the 64's colors.

Program Conversions

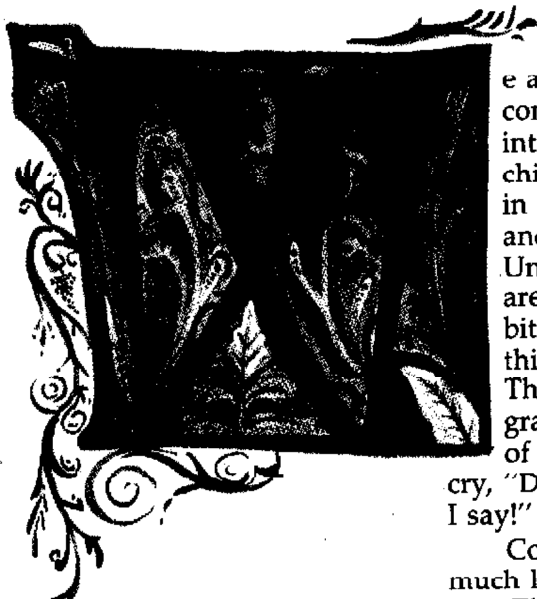
I used to own a PET Commodore Computer. Since then, I have been a subscriber to COMPUTE! magazine. COMPUTE! once published a program that would help in converting 64 and VIC-20 programs to the PET. I now own a 64 and need to convert some of my PET software into 64 format. Can you help?

Darren Storkamp

Conversions from one computer to another can sometimes become very involved because of the problems that POKEs or SYSs to machine-specific ROM routines can cause. The best way to attempt something like this might be to try to write a program on the new machine, in your case the 64, that follows the logic and flow of the old program. Even doing it this way, though, does not guarantee that the program will work properly. Some very simple

A Parser's Tale
How Adventure Games Work

Charles Brannon, Program Editor



We all know that so far computers are not truly intelligent. Like all machines, computers operate in a consistent, logical, and straightforward way. Unlike people, computers are unable to make arbitrary decisions. Everything is black and white. That's why so many programmers, bent by weeks of midnight programming, cry, "Do what I mean, not what I say!"

Computers make decisions much like your home thermostat. The thermostat does not know that it is too warm, and therefore it needs to turn on the air conditioner. Two dissimilar metals, bound together, twist as one metal expands farther than the other. The metal plates make contact with the air conditioner switch, and separate when they cool down, releasing the air conditioner. A computerized thermostat would be no more aware of its function than the mechanical one. Machines operate in a predictable fashion.

Yet adventure games appear to be smart. In an adventure, you are playing in a specialized world, created by a programmer and administered by the computer. The descriptions paint a mental picture, and as you play you get the illusion that the adventure world is a complete, though tiny, universe. The computer seems to understand what you say, as long as you use the right vocabulary. You can open doors, light lamps, fight with trolls, converse with aliens, question criminals, dig for treasure, even ask for help. While you are playing an adventure, you can remain unaware that you are solving and rearranging a complex data base.

Adventure games, sometimes described as interactive fiction, have a basic story line, characters, and a setting. The setting may be a medieval dungeon, a distant planet (in a

alaxy far, far away), an alien paceship, or even a modern hopping mall. You are usually he protagonist, but you are not here in person. Instead, you ommand your alter ego, who cts out your commands.

Your persona may be a word-wielding treasure seeker, detective, or an average, hap-ss urbanite. You control your haracter by giving it commands ke GO WEST or EAT HOUSE. fact, you are commanding the mputer to carry out your ac-ons. Some adventures let you more detailed, as in OPEN HE MANILA ENVELOPE, TAKE OUT THE LETTER, AND READ IT TO ME. In order to follow your orders, the computer must break the sentence into subcommands by checking for commas, periods, and conjunctions. Words like IT must be replaced with the most recent object. Adjectives and articles should be discarded. The sentence would become OPEN ENVELOPE/[REMOVE] LETTER/READ [LETTER].

Parsing

The process of breaking down and interpreting your command is called *parsing*. A parser routine within the program breaks each subcommand into an action verb (such as GO, OPEN, or READ) and an object (LETTER, HOUSE). The verb is then looked up in a dictionary of commands and replaced by a number. Most adventure games offer several synonyms. EXAMINE and LOOK are assigned the same number. The number causes the computer to jump to a specific subprogram that handles that action.

The object of a sentence is also turned into a number. This is a little more difficult. For example, the adventure might describe a room with an "old brown bag on the table." The player might say OPEN BAG, or GET THE BROWN BAG, or even TAKE OLD SACK. But BROWN BAG, BAG, and OLD SACK are all reduced to the same number. The GET/TAKE/PICK UP routine then uses that number to handle the request. For example, EAT POISON and EAT TREE must be handled in very different ways.

The more advanced parsers have even more to deal with. If you entered SEE WHAT'S IN E BAG, the adventure could k it down to SEE BAG. SEE nonymous with EXAMINE. EXAMINE routine checks its data base to see just what a BAG is, noting qualities such as the fact that a bag must be opened to see what's inside. You may then be told to open the bag first, or the adventure could assume that's what you've implied.

The Game Data Base

In a way, an adventure is an application like a disk operating system (DOS). In DOS, you use commands to manipulate files, for example, ERASE TEST. An adventure is no different, except you are manipulating the adventure's data base. An adventure data base consists of a map which describes how rooms or locales are linked together, objects such as treasure, and the status of various objects and situations.

You may be right next to another room, but unless the map allows direct movement to it (through a door, window, or transporter beam), you have to take another route. A room description includes legal exits and where the exits lead to, what objects the room contains, and room status, such as whether it's dark or lit. When you remove an object from a room, the room "forgets" that object. When you drop an object, the object is added to the room's description. Your player's status also has to be updated when you pick up an object, lose an object, gain powers, or get hurt. Some realtime adventures (where the clock keeps ticking and action keeps happening while you are deciding what to do) even take into account player fatigue. Your alter ego must sleep to regain energy.

Objects must be monitored. A lamp has a certain fuel supply, which is used up over time. The lamp can be either on or off, which temporarily affects the room description, lighting up a dark room. Some objects are incomplete in themselves, and must be assembled with other objects. For example, you could separately collect a bottle, some string, and some cooking oil. If it occurred to you to MAKE LAMP, the three items would become a crude lamp. A new object has been created, replacing the three separate ones. Don't think that you can make anything you want, though. Unless the programmer planned ahead to specifically allow you to create a lamp, you couldn't assemble one, even if you had all the necessary parts.

> "It can be disturbing when you penetrate the illusion and realize you are the one being programmed."

There are also variables for global status, such as the time of day. In a space adventure, there may be a status for the entire ship, like fuel and shields remaining. In more complex adventures, other people are like independent objects, with their own characteristics and descriptions. All these qualities, though, are numbers, and these numbers let a computer make arbitrary decisions. You can't GO NORTH if there is no north exit. If you have no sword, you can't fight with it. Unless you have a lamp, you cannot see within a dark room. The computer does not decide these things as a person would. It just understands statements like:

```
IF OBJECT=10 THEN PRINT
"YOU CANNOT SEE HERE."
```

Anticipated Actions

The most difficult part of designing an adventure is not creating the basic plot and world, but in anticipating the actions the player might take. Again, there is nothing open-ended in an adventure. Every possible action you may try has to have been predicted and programmed for. Some players become frustrated by the illusion, and don't understand why they can't get the computer to do what they want. Certain synonyms just aren't in the adventure dictionary. You may be faced with a locked door, but without a key. "But, aha!," you say, "I have a crowbar that worked on another door." You try the crowbar, and it doesn't work. The programmer either forgot about the crowbar, or never intended it to be that easy.

It can be disturbing when you penetrate the illusion and realize you are the one being programmed. The adventure may have many solutions, but you are just trying to figure out one of the predetermined actions planned for you. No action you take could be described as creative or innovative, since the programmer already knew that you would try it. An adventure tries to make you feel that you are participating and affecting the outcome of the adventure, but you are really just solving a complex puzzle or maze. If you realize this, the frustration may disappear, and you can concentrate on cracking the programmer's schemes. You aren't really playing against the computer, but trying to unravel a cleverly contrived mystery. ©

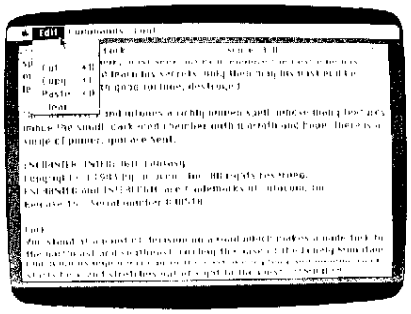
from a catalyst in reading development to a therapeutic tool in the treatment of a suicidal youngster.

On To Qyntarr

Sir-Tech Software's next game, says Woodhead, will be an all-text adventure called *The Mines of Qyntarr,* in the tradition of the original *ZORK* created by Infocom.

"It's an extremely complex and involved adventure game. And the major effort we're making right now is to make it a lot more user-friendly in terms of its command parser. [See "The Parser's Tale" in this issue.] I hope to get a command parser running that understands more complex grammar than *ZORK*," adds Woodhead.

In speaking with programmers and designers of adventure games, the name Infocom almost invariably comes into the conversation. It's widely acknowledged that the Cambridge, Massachusetts, based company is the uncontested leader in the production of sophisticated, all-text adventure games.



Infocom's adventures are considered the standard for quality in all-text formats. Here, on the Macintosh, a pull-down menu can be seen in the upper left corner.

ZORK Forever!

ZORK, for example, has a command parser vocabulary in excess of 600 words, allowing significant variety in the kinds of sentences that the game can understand. Infocom's new release, *Sorcerer,* a sequel to another Infocom game, *Enchanter,* has a vocabulary in excess of a thousand words.

Infocom has spent its time and efforts developing the plot, the writing, the puzzles, and the parsing rather than on sound and graphics—the latter two of which Infocom vice president and master programmer Marc Blank calls "bells and whistles."

And the results have demonstrated the popularity of well-done all-text games. The *ZORK* series, which Blank coauthored, has already well surpassed the quarter million mark in number of disks sold. A now-defunct *ZORK* User Group (ZUG) boasted more than 20,000 members nationwide. *ZORK* T-shirts, bumper stickers, posters, and special clue books have all flourished. Infocom's games are available in versions for most personal computers.

When game historians give credit for the development and the legitimization of the term *interactive fiction* as applied to a certain type of computer adventure game, it will be Infocom which will get the laurels.

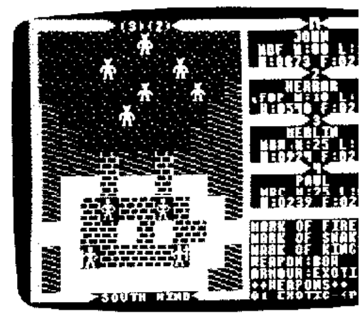
Seeing The Movie Vs. Reading The Book

But what about the future of all-text adventure games as computers become powerful enough to have sophisticated parsers *and* colorful graphics all at the same time?

"For the next year or so there'll still be a market for the incredibly well-done text adventures," says Richard Garriott, coauthor of Origin Systems' *Ultima* series of fantasy role-playing games, available for Apple, Atari, and Commodore machines. "Anybody other than the Infocom style cannot succeed at this point. Infocom has put together a very, very so ticated parser, and the non-player characters within the game actually have some in gence to their movements.

"I really think that as v develop better computer sys tems that surely this same k of technique—if not the qua that Infocom is putting into their games—can also then the added feature of the realtime graphics and anim put on top as well," he say

"The standard argumen



Origin Systems' Exodus: Ultim *offers graphic images of the ga left while game information is played at right.*

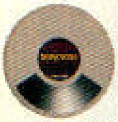
that the game with graphi like going to see the movi the game with text is like ing the book," adds Garric "Some people will still ha some preference between two. But the vast majority marketed products will alr have to turn to graphics b of the demand of the publ

"You Can Do Anyt You Want"

Garriott, 23, has been wri computer fantasy-adventu games since his sophomo in high school. He comple fantasy games while still i school, learning more abo genre with each attempt.

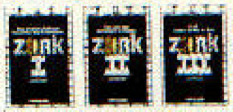
Origin Systems' *Ultim* tasy role-playing series is tament to the strength of Garriott's game-designing ents. "The key to the *Ulti*

THE REAL TRICK IS GETTING OUT.



Expect the unexpected the first time you experience Infocom's interactive fiction. Because you won't be booting up a computer game. You'll be stepping into a story.

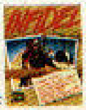
You'll find yourself at the center of an exciting world that continually challenges you with surprising twists, unique characters (many of whom possess extraordinarily developed personalities) and original, logical, often hilarious puzzles.



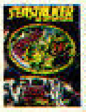
Communication is carried on just as it is in a book—in prose. And interaction is easy—you type in full English sentences.

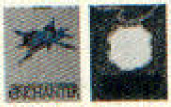
But if you think getting inside a story is a pretty neat trick, just try getting out.

The most remarkable thing about Infocom's interactive fiction is that you become almost inextricably involved with it. That's not our opinion—it's the testimony of our customers. They tell us their pulse rates have skyrocketed and their palms have sweated as they've striven to solve the mysteries of our tales. And even when they've paused in the course of their adventures to attend to their everyday lives, their minds have continued to churn away at what the next step should be, how to alter strategy, where the ultimate solution lies.

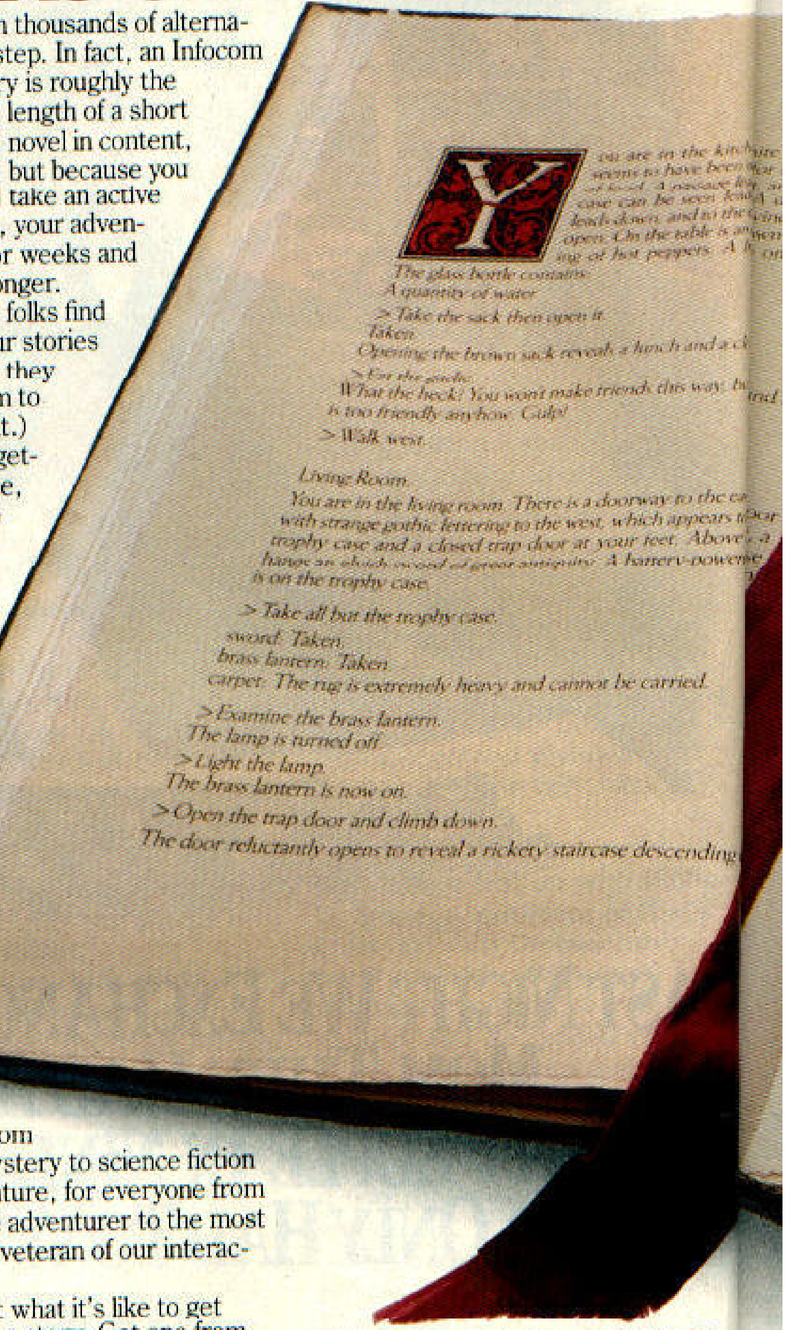


Obsessions? Yes, but magnificent ones. For the first time, you can be more than a passive reader—you can become the story's main character and driving force. You can shape its course of events by what you choose to do. And you enjoy enormous freedom in your choice of actions—you have hundreds, even thousands of alternatives at every step. In fact, an Infocom interactive story is roughly the length of a short novel in content, but because you take an active role in the plot, your adventure can last for weeks and months. (Or longer. Frankly, some folks find being inside our stories so fascinating, they just don't seem to want to get out.)





As hard as getting out may be, though, we've made it easy for everyone to get into Infocom's interactive fiction.



We write everything from fantasy to mystery to science fiction to high adventure, for everyone from the first-time adventurer to the most experienced veteran of our interactive stories.

So find out what it's like to get inside a story. Get one from Infocom. Because with Infocom's interactive fiction, there's room for you on every disk.

Ease into interactive fiction with our Can't-Lose Sampler Offer! Watch for Infocom's new sampler disk at your favorite dealer—and get your first taste of interactive fiction for just $7.95 (suggested retail price). You've got nothing to lose (in fact, you can make a shiny new nickel in the bargain) because your Infocom sampler disk even comes with a coupon entitling you to $8 off the purchase of your first complete Infocom story, plus coupons for other exciting bonuses!

INFOCOM™

55 Wheeler St., Cambridge, MA 02138

For your: Apple II, Macintosh, Atari, Commodore 64, CP/M 8", DECmate, DEC Rainbow, DEC RT-11, HP 150 & 110, IBM PC* & PCjr, KAYPRO II, MS-DOS 2.0*, NEC APC, NEC PC-8000, Osborne, TRS-80 Color Computer, Tandy 2000, TI Professional, TI 99/4A, TRS-80 Models I & III.

*Use the IBM PC version for your Compaq and the MS-DOS 2.0 version for your Wang, Mindset, Data General System 10, GRiD and many others.

Horse Racing

Robert Onufer

Watch your favorite pony win (or lose) in this detailed, effective simulation of race-track betting. Versions included for the TI-99/4A with Extended BASIC, the Commodore 64, VIC-20, Apple II+/IIc/IIe, and IBM PC/PCjr.

"Horse Racing" is a multiplayer game in which you must wager on horses. Up to nine players may play the game, and each starts the game with $500. There are five races. The player with the greatest amount of money after the fifth race is the winner. To make the simulation more accurate, the program recalculates the odds at the beginning of each race. That means that the favorite will always pay lower odds. And you will always know these new odds because they are posted just before the race begins.

Some of the most exciting horse races occur when the track conditions vary. The reason for this is that long shots often have a better chance of winning on slippery tracks because the track could cause some of the better horses to fall or not get a good footing for speed. Horse Racing varies the track conditions from race to race and gives a slight advantage to one horse for each particular track condition. This advantage is taken into account when the initial odds are calculated, making a horse the favorite very often, but not always. In the TI version of Horse Racing, you can change the advantage by changing the value of AD(T) in line 1030. (For other versions, see "Programmer's Notes.")

Track Graphics

After the final odds are displayed, the track is drawn using custom characters. These are drawn on the screen transparently and then lit up all at once in either line 710 or line 720, depending on track conditions. Using the powerful graphics capabilities of TI Extended BASIC, the horses are magnified sprites drawn on a 16 × 16 grid. The animation effect is created by alternating each horse through two different patterns, making the horses appear to move. Speed is randomly updated in lines 750 through 790.

When the program determines that a horse has crossed the finish line, the position of each horse is checked. The victory is given to the horse furthest across the finish line. The track is then erased, payoffs are made or monies deducted, and a summary appears on the screen. After the last race, you may choose to play again by pressing the 1 key or to exit the game by pressing the 2 key.

Program 1: TI Horse Racing

Extended BASIC required.

```
100 DIM NOTE(26),DUR(26)
110 FOR I=1 TO 26 :: READ NOTE(I)
    ,DUR(I):: NEXT I
120 DATA 294,30,392,30,494,30,587
    ,45,587,15,587,30,494,45,494,
    15,494,30
130 DATA 392,30,494,30,392,30,294
    ,90,294,30,392,30,494,30,587,
    45,587,15,587,30
140 DATA 494,45,494,15,494,30,294
    ,30,294,30,294,30,392,90
150 IMAGE HORSE ## :  ### TO 1
160 A$="00000171 0F0F0F18204080000
    000000000589C3FF8E0C0784422 11
    0000000000"
170 B$="00000171170F0F0E040201000
    000000000589C3FF8F8F030101070
    0000000000"
180 C$="0000000000C0BFBF3F1019010
    00000000000046371FDE3E1C3C54D
    2300000000"
```

```
190 CALL CHAR(128,A$)
200 CALL CHAR(132,B$)
210 CALL CHAR(136,C$)
220 CALL CLEAR :: CALL SCREEN(3):
    : DISPLAY AT(12,9):"HORSE RAC
    ING"
230 GOSUB 990 :: FOR DELAY=1 TO 3
    00 :: NEXT DELAY
240 CALL CLEAR :: CALL SCREEN(5):
    : K=0
250 FOR I=0 TO 14 :: CALL COLOR(I
    ,16,1):: NEXT I
260 DISPLAY AT(8,4):"NUMBER OF PL
    AYERS ?"
270 ACCEPT AT(8,26)SIZE(1)VALIDAT
    E(DIGIT)BEEP:N :: IF (N=0)THE
    N CALL HCHAR(8,28,32,2):: GOT
    O 270
280 FOR I=1 TO N :: CASH(I)=500 :
    : NEXT I
290 DISPLAY AT(10,1):"EACH PLAYER
     STARTS WITH $500"
300 DISPLAY AT(14,2):"HORSES ARE
    NUMBERED FROM"
310 DISPLAY AT(16,8):"BOTTOM TO T
    OP"
320 FOR D=1 TO 600 :: NEXT D
330 K=K+1
340 IF (K>5)+(FL=1)THEN FL=0 :: G
    OTO 1460
350 FOR I=1 TO 5 :: AD(I),AM(I)=0
     :: NEXT I
360 GOSUB 1000 !TRACK COND.
370 GOSUB 1080 !DETERMINE ODDS
380 GOSUB 1150 !PLACE BETS
390 CALL CLEAR
400 GOSUB 560 !DRAW TRACK
410 DISPLAY AT(4,9):"HANOVER DOWN
    S"
420 PAT=128 :: PAT2=132 :: PAT3=1
```

Programmer's Notes For VIC, 64, IBM, and Apple Versions

Patrick Parrish, Programming Supervisor

The VIC-20, Commodore 64, IBM, and Apple versions of "Horse Racing" are designed to capture the excitement of going to the races. As many as nine players can play the game by betting on one of five horses (six horses in the IBM version). Five hundred dollars is awarded to each player to start the game.

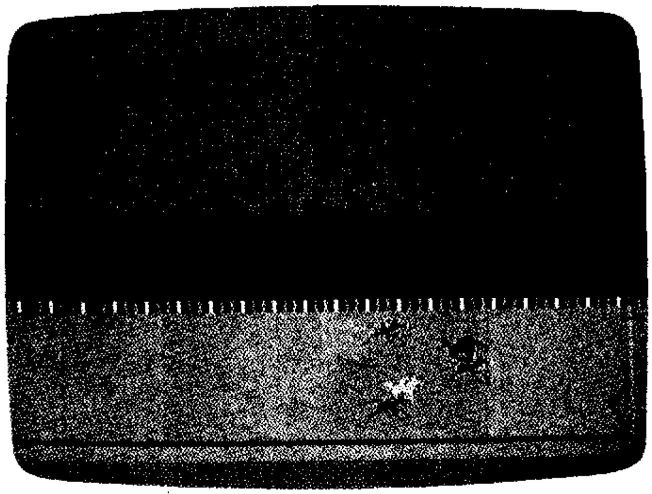
Winning odds are based on the wagers made before a race. When betting, bear in mind that each horse favors a different track condition. The advantage gained by a horse running under optimum conditions is determined by the variable AD(T) located in lines 50, 550, 730, and 380 in the VIC, 64, IBM, and Apple versions, respectively. If you want to add to the advantage given to a particular horse under specific track conditions, increase the value assigned to this variable.

The VIC version of Horse Racing runs on the unexpanded VIC with a few bytes to spare. The 64 version uses multicolor sprites to define the horse and riders. A short ML routine to move the sprites is loaded in from the DATA statements beginning at line 1350. The IBM version requires BASICA and a color/graphics adapter for the PC, or a PCjr with Cartridge BASIC. The race track is depicted on graphics screen 1 with the horse and riders drawn from DATA stored in lines 290–500.

The Apple version of Horse Racing runs on all Apple IIs with DOS 3.3 or ProDOS. Since the program uses the secondary text page (at 2048, where the BASIC program normally resides), a series of POKEs is required to relocate the BASIC program. These POKEs are done by Program 5, which serves as our loader program. It locates Program 6 (which must be saved as "HORSERACE") at location 24576 by POKEing 104 and 103 (the high- and low-byte pointers to the start of the BASIC program) with 96 and 0, respectively (256*96+0=24576).

Program 6 defines the horses as high-resolution shapes with shape table DATA stored from line 790 on. The movement of the horses is animated by use of a high-resolution page-flipping routine in lines 190–210. This routine lets you view the horses on one high-resolution screen while drawing them further along the track on a second high-resolution screen. After the shapes have been placed on the second screen, this screen is viewed and drawing is done on the first screen. This sequence continues until the race is won.

A series of POKEs enables us to page flip in Program 6. By alternately accessing locations −16300 and −16299, either high-resolution screen 1 or 2 is displayed. POKEing location 230 with 32 or 64 causes the shapes to be drawn on high-resolution screen 1 or 2, respectively.



"Horse Racing" for the TI home computer.

```
    36 :: X=4 :: CALL SCREEN(8)
430 CALL MAGNIFY(3)
440 CALL SPRITE(#1,128,7,156,5,#2
    ,128,16,148,5,#3,128,5,140,5,
    #4,128,2,132,5,#5,128,14,124,
    5)
450 GOSUB 990 !OPENING SONG
460 CALL MOTION(#1,0,SP1,#2,0,SP2
    ,#3,0,SP3,#4,0,SP4,#5,0,SP5)
470 CALL POSITION(#1,Y1,X1,#2,Y2,
    X2,#3,Y3,X3,#4,Y4,X4,#5,Y5,X5
    )
480 IF X2>230 OR X1>230 THEN 810
490 IF X4>230 OR X3>230 THEN 810
500 IF X5>230 THEN 810
510 PAT=PAT+X :: PAT2=PAT2-X :: X
    =-X
520 CALL PATTERN(#1,PAT,#2,PAT2,#
    3,PAT,#4,PAT2,#5,PAT)
530 FOR DELAY=1 TO 8 :: NEXT DELA
    Y
540 GOSUB 740 !UPDATE MOTION
550 GOTO 460
560 CALL CHAR(97,"FF")!DRAW RACE
    TRACK
570 CALL CHAR(96,"FFFFFF666666666
    6")
580 CALL CHAR(120,"FFFFFFFFFFFFFF
    FF")
590 FOR I=9 TO 12 :: CALL COLOR(I
    ,1,1):: NEXT I
600 CALL CHAR(104,"8080008080008080
    80")
610 CALL CHAR(112,"0101030307CFEF
    FF")
620 CALL CHAR(114,"C0F0F8FCFCFFFF
    FF")
630 CALL CHAR(113,"FFFFFFFFFFFFFF
    FF")
640 CALL HCHAR(15,1,96,32)
650 FOR I=16 TO 22 :: CALL HCHAR(
    I,1,120,32):: NEXT I
660 CALL HCHAR(23,1,97,32)
670 CALL VCHAR(16,31,104,7)
680 FOR I=11 TO 14 :: CALL HCHAR(
    I,1,113,32):: NEXT I
690 FOR I=1 TO 31 STEP 2 :: CALL
    HCHAR(10,I,112):: NEXT I
700 FOR I=2 TO 32 STEP 2 :: CALL
    HCHAR(10,I,114):: NEXT I
710 IF T<5 THEN CALL COLOR(9,2,12
    ,10,16,12,11,13,1,12,12,1)
720 IF T=5 THEN CALL COLOR(9,2,4,
    10,16,4,11,13,1,12,4,1)
730 RETURN
740 RANDOMIZE !SPEED OF HORSES
750 SP1=INT(5*RND+AD(1))
760 SP2=INT(5*RND+AD(2))
770 SP3=INT(5*RND+AD(3))
780 SP4=INT(5*RND+AD(4))
790 SP5=INT(5*RND+AD(5))
800 RETURN
810 ATEM=MAX(MAX(X1,X2),MAX(X3,X4
    ))
820 A=MAX(ATEM,X5)
830 IF A=X1 THEN WIN=1 :: GOTO 87
    0
840 IF A=X2 THEN WIN=2 :: GOTO 87
    0
850 IF A=X5 THEN WIN=5 :: GOTO 87
    0
860 IF A=X3 THEN WIN=3 ELSE WIN=4
870 FOR I=1 TO 5
880 IF I=WIN THEN 900
890 CALL DELSPRITE(#I)
900 NEXT I
910 CALL MAGNIFY(4)
920 CALL MOTION(#WIN,0,0):: CALL
    LOCATE(#WIN,150,124):: CALL P
    ATTERN(#WIN,PAT3):: FOR DELAY
    =1 TO 100 :: NEXT DELAY
930 DISPLAY AT(4,7):"THE WINNER I
    S #";WIN
940 CALL SOUND(1000,392,5)
950 CALL SOUND(1000,332,5)
960 CALL SOUND(1000,262,5)
970 GOSUB 1340 !PAYOFF
980 GOTO 330
990 FOR I=1 TO 26 :: CALL SOUND(D
    UR(I)*3.5,NOTE(I),5):: CALL S
    OUND(30,40000,5):: NEXT I ::
    RETURN
1000 RANDOMIZE !TRACK COND
1010 T=INT(5*RND)+1
1020 TR$(1)="FAST" :: TR$(2)="GOO
     D" :: TR$(3)="SLOW" :: TR$(4
     )="MUDDY" :: TR$(5)="TURF"
1030 AD(T)=.4 :: AM(T)=500
1040 CALL CLEAR :: CALL SCREEN(8)
1050 FOR I=0 TO 8 :: CALL COLOR(I
     ,2,1):: NEXT I
1060 DISPLAY AT(8,12):"RACE";K
1070 DISPLAY AT(12,3):"TRACK COND
     ITION:{3 SPACES}";TR$(T):: F
     OR DELAY=1 TO 300 :: NEXT DE
     LAY :: RETURN
1080 RANDOMIZE !INITIAL ODDS
1090 MT=0
1100 FOR I=1 TO 5 :: M(I)=INT(100
     0*RND)+0.1+AM(I):: MT=MT+M(I
     ):: NEXT I
```

```
1110 FOR I=1 TO 5 :: OD(I)=INT(MT
     /M(I))
1120 IF OD(I)>20 THEN OD(I)=20
1130 NEXT I
1140 RETURN
1150 CALL CLEAR :: CALL SCREEN(7)
     !PLACE BETS
1160 DISPLAY AT(1-(N<5),9):"CURRE
     NT ODDS"
1170 FOR I=1 TO 5 :: DISPLAY AT(1
     +I-(N<5)*2,3):USING 150:I,OD
     (I):: NEXT I
1180 FOR I=2 TO 2*N STEP 2
1190 IF CASH(I/2)<1 THEN AMT(I/2)
     =0 :: GOTO 1270
1200 DISPLAY AT(5+I+(9-N)/2,3):"P
     LAYER";I/2;"BETS - HORSE?"
1210 ACCEPT AT(5+I+(9-N)/2,26)VAL
     IDATE(DIGIT)BEEP SIZE(1):H(I
     /2)
1220 IF H(I/2)>5 THEN CALL HCHAR(
     5+I+(9-N)/2,29,32,3):: GOTO
     1210
1230 DISPLAY AT(6+I+(9-N)/2,3):"A
     MOUNT?" :: ACCEPT AT(6+I+(9-
     N)/2,24)SIZE(3)VALIDATE(DIGI
     T)BEEP:AMT(I/2)
1240 IF AMT(I/2)>CASH(I/2)THEN 12
     30
1250 M(H(I/2))=M(H(I/2))+AMT(I/2)
1260 MT=MT+AMT(I/2)
1270 NEXT I
1280 CALL CLEAR :: CALL SCREEN(14
     )
1290 DISPLAY AT(5,11):"NEW ODDS"
1300 FOR I=1 TO 5 :: OD(I)=INT(MT
     /M(I))
1310 IF OD(I)>20 THEN OD(I)=20
1320 DISPLAY AT(7+I*2,4):USING 15
     0:I,OD(I):: NEXT I
1330 FOR DELAY=1 TO 1000 :: NEXT
     DELAY :: RETURN
1340 FOR I=1 TO N
1350 IF H(I)=WIN THEN CASH(I)=CAS
     H(I)+INT(AMT(I)*INT(MT/M(WIN
     )))
1360 IF H(I)<>WIN THEN CASH(I)=CA
     SH(I)-AMT(I)
1370 NEXT I
1380 CALL DELSPRITE(ALL):: CALL C
     LEAR
1390 DISPLAY AT(3,12):"SUMMARY"
1400 P$="RACES" :: IF K=1 THEN P$
     ="RACE"
1410 DISPLAY AT(5,9):"AFTER";K;P$
1420 FOR I=1 TO N :: DISPLAY AT(8
     +I,3):USING 1510:I,CASH(I)::
      NEXT I
1430 FOR DELAY=1 TO 1500 :: NEXT
     DELAY
1440 FL=1 :: FOR I=1 TO N :: IF C
     ASH(I)>0 THEN I=N :: FL=0
1450 NEXT I :: RETURN
1460 CALL DELSPRITE(ALL):: CALL C
     LEAR :: CALL SCREEN(16)
1470 DISPLAY AT(12,10):"GAME OVER
     " :: DISPLAY AT(21,3):"PRESS
      1 TO PLAY AGAIN" :: DISPLAY
      AT(23,4):"PRESS 2 TO END GA
     ME"
1480 CALL KEY(0,KEY,S)
1490 IF KEY=49 THEN 240
1500 IF KEY<>50 THEN 1480
1510 IMAGE PLAYER ##:   $######
1520 CALL CLEAR :: END
```

Program 2: 64 Horse Racing

Refer to the "Automatic Proofreader" article before typing this program in.

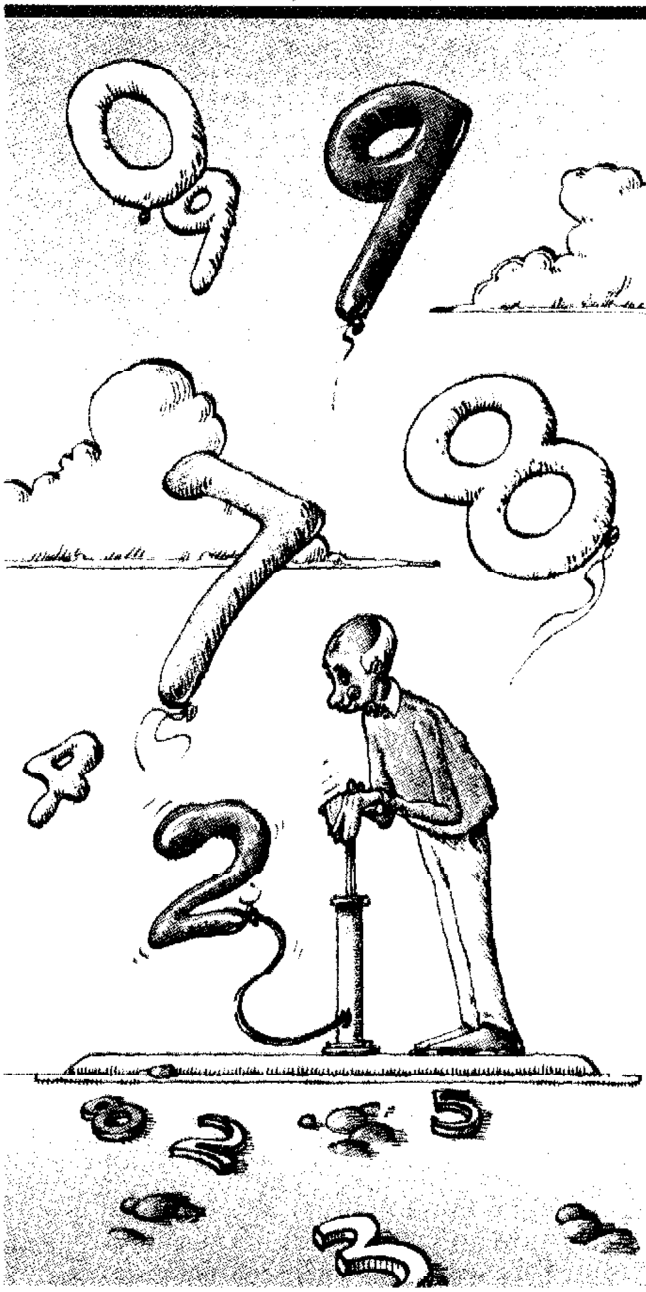
Translation by Jeff Hamdani, Editorial Programmer

```
10 POKE53280,6:DIMHF(28),LF(28),DR(28)
                                   :rem 71
20 PRINT"{CLR}{9 DOWN}"TAB(14)"{YEL}{RVS}
   HORSE RACING{OFF}{WHT}"        :rem 172
30 PRINT"{9 DOWN}{WHT}"TAB(5)"LOADING DAT
   A.......PLEASE WAIT":V=53248   :rem 143
40 FORI=12288TO12414:READA:POKEI,A:NEXT:P
   OKEV+28,31                     :rem 136
50 POKEV+37,0:POKEV+38,9:FORI=1TO5:POKE20
   39+I,192:POKEV+38+I,6-I:NEXT   :rem 210
60 FORI=0TO8STEP2:READA:POKEV+I+1,A:NEXT:
   FORI=1TO5:READCR(I):NEXT         :rem 6
70 FORI=1TO26:READHF(I),LF(I),DR(I):NEXT:
   I=0:CT=0                       :rem 233
80 IFPEEK(49523)=212ANDPEEK(49524)=96THEN
   120                              :rem 5
90 I=I+1:READA:CT=CT+A:IFA=256THEN110
                                  :rem 225
100 POKE49151+I,A:GOTO90          :rem 129
110 IFCT<>45269THENPRINT"{CLR}ERROR IN RE
    ADING DATA IN.":END           :rem 218
120 PRINT"{CLR}":S=54272:FORL=STOS+24:POK
    EL,0:NEXT                     :rem 211
130 POKES+24,15:POKES+5,18:POKES+6,245
                                  :rem 206
140 POKE53280,15:POKE53281,15:PRINT"
    {10 DOWN}{BLU}"TAB(14)"HORSE RACING":
    GOSUB1060                     :rem 177
150 PRINT"{3 DOWN}"TAB(7)"NUMBER OF PLAYE
    RS (1-9)? ";:HR=0              :rem 94
160 GETZ$:N=VAL(Z$):IF(N<1ORN>9)THEN160
                                  :rem 109
170 PRINTZ$:FORI=1TO200:NEXT:FORI=1TON:CH
    (I)=500:NEXT:PRINT"{CLR}{7 DOWN}"
                                  :rem 177
180 PRINTTAB(6)"EACH PLAYER STARTS WITH $
    500."                         :rem 170
190 PRINTTAB(3)"{DOWN}WHEN A PLAYER LOSES
     ALL OF HIS/HER"              :rem 253
200 PRINTTAB(10)"{DOWN}MONEY, THE GAME EN
    DS."                            :rem 4
210 PRINT"{2 DOWN}HORSES ARE NUMBERED FRO
    M BOTTOM TO TOP."              :rem 81
220 FORI=1TO4000:NEXT              :rem 18
230 HR=HR+1                        :rem 98
240 FORI=1TON:AD(I)=0:AM(I)=0:NEXT:rem 51
250 GOSUB530:REM TRACK CONDITION  :rem 233
260 GOSUB590:REM CALCULATE ODDS   :rem 140
270 GOSUB630:REM PLACE BETS        :rem 99
280 GOSUB400:REM DRAW TRACK       :rem 111
290 POKES+5,17:POKES+6,24:POKES+2,4:POKES
    +3,5:POKEV+(2*T-2),24+AD(T):SYS49152
                                   :rem 35
300 FORI=0TO8STEP2:A=PEEK(V+I):IFA=65THEN
    WN=(I+2)/2                    :rem 140
```

Software for children often benefits from large display characters and numbers. Here's a method for creating big numbers on the TI-99/4A, with a simple example program—a number recognition game which uses the larger digits. Includes versions for Commodore VIC and 64, Atari, Apple, IBM PC/PCjr, and the Color Computer.

The Number Game

Lou Tylee

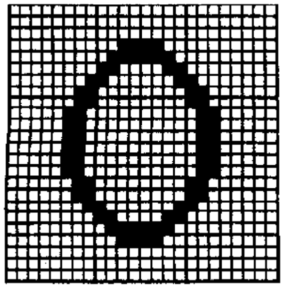


The built-in number characters on the Texas Instruments 99/4A Home Computer are too small to really grab a child's attention. Using the character definition capabilities of the 99/4A, representations of the numbers 0 through 9 can be developed which are three times taller and wider than these built-in digits. And these larger number characters can be used in your own programs.

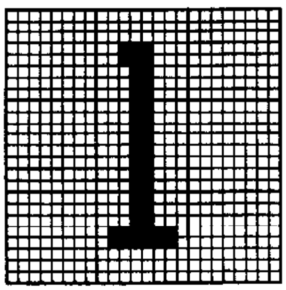
Magnifying The Numbers

In the May 1983 issue of COMPUTE!, C. Regena wrote a tutorial on the use of TI graphics ("Programming The TI: Graphics"). Regena explains that each character on the display screen is an 8 × 8 grid of 64 dots. When you press a number key on the TI keyboard, that number is displayed using one such character. By employing the CALL CHAR statement in TI BASIC to turn dots on and off within a particular 8 × 8 grid, custom characters can be defined. The larger numbers here each use 9 custom characters in a 3 × 3 array. The figure shows these numbers and corresponding hexadecimal codes for defining characters.

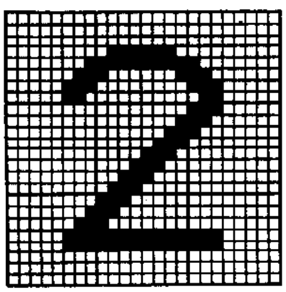
Examining the figure, you may wonder if these numbers could perhaps be defined in a simpler manner. For example, it is possible to represent each number by straight line segments only, such as are used on digital watches. Certainly, this would work, but it may not be advisable for teaching young children, because children learn numbers for the first time in a pattern recognition mode, trying to match similar objects. Hence, the numbers in the figure are designed to mimic (as closely as possible) the TI keyboard depictions of the numbers. For older children, who are used to seeing numbers written in different ways, the digital watch approach to number display would be fine.



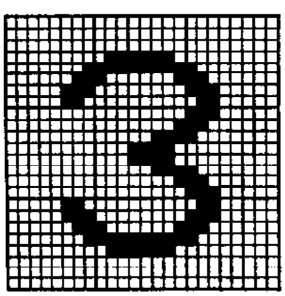
Row	Column	Index	Hex Code
1	1	0	0000000000000103
1	2	1	0000003C7EC38100
1	3	2	00000000000080C0
2	1	3	0306060606060603
2	2	53	0000000000000000
2	3	4	C0606060606060C0
3	1	5	0301000000000000
3	2	6	0081C37E3C000000
3	3	7	C080000000000000



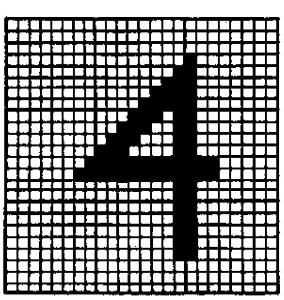
Row	Column	Index	Hex Code
1	1	53	0000000000000000
1	2	8	0000003838181818
1	3	53	0000000000000000
2	1	53	0000000000000000
2	2	9	1818181818181818
2	3	53	0000000000000000
3	1	53	0000000000000000
3	2	10	1818187E7E000000
3	3	53	0000000000000000



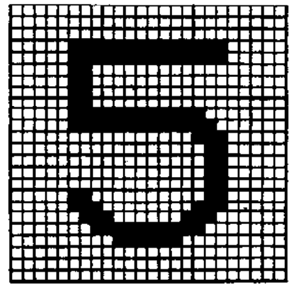
Row	Column	Index	Hex Code
1	1	45	0000000001030706
1	2	46	0000007EFF810000
1	3	47	0000000080C0E060
2	1	53	0000000000000000
2	2	11	000103070E1C3870
2	3	12	E0C0800000000000
3	1	13	0001030707000000
3	2	14	E0C080FFFF000000
3	3	15	000000E0E0000000



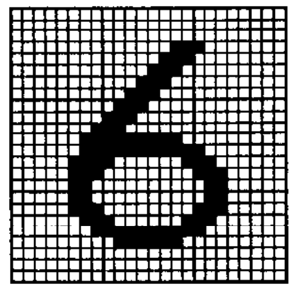
Row	Column	Index	Hex Code
1	1	45	0000000001030706
1	2	46	0000007EFF810000
1	3	47	0000000080C0E060
2	1	53	0000000000000000
2	2	16	0001071E1E070100
2	3	42	E0C080000080C0E0
3	1	48	0607030100000000
3	2	43	000081FF7E000000
3	3	44	60E0C08000000000



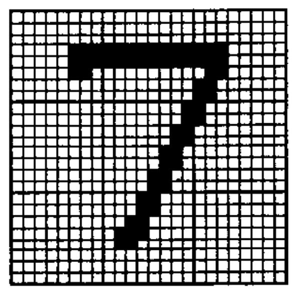
Row	Column	Index	Hex Code
1	1	53	0000000000000000
1	2	17	000000000103070F
1	3	18	0000008080808080
2	1	19	0000000001030000
2	2	20	1D3971E1FFFF0101
2	3	21	80808080E0E08080
3	1	53	0000000000000000
3	2	22	0101010101000000
3	3	23	8080808080000000



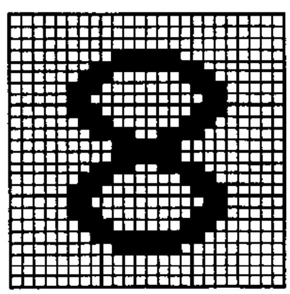
Row	Column	Index	Hex Code
1	1	24	0000000707060606
1	2	35	000000FFFF000000
1	3	25	000000E0E0000000
2	1	26	0607070000000000
2	2	27	00FFFF0000000000
2	3	28	0080C0E060606060
3	1	48	0607030100000000
3	2	43	000081FF7E000000
3	3	44	60E0C08000000000



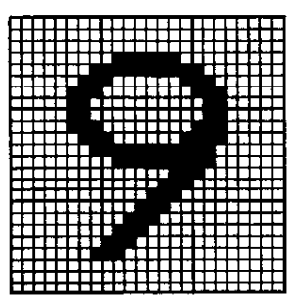
Row	Column	Index	Hex Code
1	1	53	0000000000000000
1	2	29	00000003070E1C38
1	3	30	0000008000000000
2	1	31	0000010303070706
2	2	32	70E0C0FEFF810000
2	3	33	0000000080C0E060
3	1	48	0607030100000000
3	2	43	000081FF7E000000
3	3	44	60E0C08000000000



Row	Column	Index	Hex Code
1	1	34	0000000707060000
1	2	35	000000FFFF000000
1	3	36	000000E0E060C0C0
2	1	53	0000000000000000
2	2	37	0101030306060C0C
2	3	38	8080000000000000
3	1	53	0000000000000000
3	2	39	1818307060000000
3	3	53	0000000000000000



Row	Column	Index	Hex Code
1	1	45	0000000001030706
1	2	46	0000007EFF810000
1	3	47	0000000080C0E060
2	1	40	0703010000010307
2	2	41	0081E77E7EE78100
2	3	42	E0C080000080C0E0
3	1	48	0607030100000000
3	2	43	000081FF7E000000
3	3	44	60E0C08000000000



Row	Column	Index	Hex Code
1	1	45	0000000001030706
1	2	46	0000007EFF810000
1	3	47	0000000080C0E060
2	1	48	0607030100000000
2	2	49	000081FF7F03070E
2	3	50	60E0E0C0C0800000
3	1	51	0000000001000000
3	2	52	1C3870E0C0000000
3	3	53	0000000000000000

Using The Magnified Numbers

Now that we have the character definitions, we need to efficiently incorporate them into a program. Ten digits, defined by nine characters each, is a total of 90 characters. Of these 90 characters, however, only 54 are distinct. Lines 150–290 of Program 1 assign each of these distinct characters to an index number in the array CI$. These 54 character indices fill an array N which is used to define the ten large digits. In lines 310–480 of Program 1, I is character N's row and J is character N's column within the 3 × 3 array used to define digit K. For example, character 17 (000000000103070F) defines the first row and second column of the number 4 (see the figure). So we can write N(4,1,2)=17.

Next, we need to relate the two arrays CI$ and N to allow drawing large numbers on the display screen. One way to accomplish this is to load each of the 54 distinct characters into character codes 106 through 159 using CALL CHAR:

```
FOR I=0 TO 53
CALL CHAR(I+106,CI$(I))
NEXT I
```

An alternative which eliminates the need for a CI$ array, is to read the character definitions directly from DATA statements

```
FOR I=0 TO 53
READ C$
CALL CHAR(I+106,C$)
NEXT I
DATA ...
DATA ...
```

where the DATA statements are identical to those used earlier to define CI$. Then, to draw digit K starting at row R and column C on the screen, we use:

```
FOR I=R TO R+2
FOR J=C TO C+2
CALL HCHAR(I,J,106+N(K,I-R+1,J-C+1))
NEXT J
NEXT I
```

This will work fine, yet it has one drawback. It requires the use of 54 custom characters. This does not leave many characters available for other graphics use. We can use another technique that only requires, at most, nine characters for each digit to be displayed on the screen at one time. So, if our application only displays two digits at any one time, just 18 characters must be defined.

Dynamic Character Definition

That technique, used in Program 1, can be called dynamic character definition. That is, character codes are redefined and reused as each number is displayed. Lines 1500–1580 draw digit K starting at row R, column C, and character code CC.

This approach requires that the contents of the CI$ array have already been assigned, as shown in lines 150–290. If we are using two digits at most, good choices for starting character codes are CC=126 for one digit and CC=135 for the other. This leaves many codes available for other graphics. As long as we require six or fewer different digits to be displayed, this method of dynamic character definition uses fewer character codes than the previous method.

The large numbers developed here have many applications. Math flash card drills, counting games, and guess-the-number games are just a few. As a sample application, the programs provide a preschool game to teach number recognition. In Program 1, which runs in either TI console BASIC or Extended BASIC, the computer randomly picks a number from 0 to 9 and displays it at the center of the screen. The child is then asked to find that number on the keyboard and press it. A correct response wins a snappy tune and a like number of blocks are drawn. An incorrect answer gets an "uh-oh" and the child is asked to try again. Since this program displays only one number at a time, dynamic character definition (CC=135) is employed for display.

Program 1: TI Number Game

```
100 RANDOMIZE
110 CALL CLEAR
120 CALL SCREEN(8)
130 PRINT TAB(8);"...PLEASE WAIT"
140 REM LOAD CHARACTER CODE ARRAY
150 DIM CI$(53)
160 FOR I=0 TO 53
170 READ CI$(I)
180 NEXT I
190 DATA 0000000000000103,0000003C7
    EC381,000000000000080C,030606060
    6060603,C06060606060600C
200 DATA 0301,0081C37E3C,C08,000000
    3838181818,1818181818181818
210 DATA 1818187E7E,000103070E1C387
    ,E0C08,0001030707,E0C080FFFF
220 DATA 000000E0E,0001071E1E0701,0
    00000000103070F,0000008080808080
    ,0000000000103
230 DATA 1D3971E1FFFF0101,8080808E
    0E0808,0101010101,808080808,000
    0000707060606
240 DATA 000000E0E,060707,00FFFF,00
    80C0E06060606,000000003070E1C38
250 DATA 0000008,000001030307070 6,7
    0E0C0FEFF81,00000000080C0E06,000
    000070706
260 DATA 000000FFFF,000000E0E060C0C
    ,0101030306060C0C,808,181830706
270 DATA 0703010000010307,0081E77E7
    EE781,E0C080000080C0E,000081FF7
    E,60E0C08
280 DATA 0000000001030706,0000007EF
    F81,00000000080C0E06,06070301,00
    0081FF7F03070E
```

```
290 DATA 60C0E0C0C08,0000000001,1C3870E0C,
300 REM LOAD CHARACTER INDEX ARRAY
310 DIM N(9,3,3)
320 FOR K=0 TO 9
330 FOR I=1 TO 3
340 FOR J=1 TO 3
350 READ N(K,I,J)
360 NEXT J
370 NEXT I
380 NEXT K
390 DATA 0,1,2,3,53,4,5,6,7
400 DATA 53,8,53,53,9,53,53,10,53
410 DATA 45,46,47,53,11,12,13,14,15
420 DATA 45,46,47,53,16,42,48,43,44
430 DATA 53,17,18,19,20,21,53,22,23
440 DATA 24,35,25,26,27,28,48,43,44
450 DATA 53,29,30,31,32,33,48,43,44
460 DATA 34,35,36,53,37,38,53,39,53
470 DATA 45,46,47,40,41,42,48,43,44
480 DATA 45,46,47,48,49,50,51,52,53
490 REM DEFINE BLOCK CHARACTER
500 CALL CHAR(119,"")
510 CALL COLOR(11,16,16)
520 REM TITLE SCREEN
530 CALL CLEAR
540 CALL SCREEN(12)
550 PRINT TAB(6);"LEARN THE NUMBERS": :
560 GOSUB 1320
570 PRINT "THIS IS A PRE-SCHOOL NUMBER"
580 PRINT "RECOGNITION GAME. THE COMPU-"
590 PRINT "TER DISPLAYS A NUMBER AND"
600 PRINT "YOU MUST FIND AND PRESS THAT"
610 PRINT "KEY ON YOUR KEYBOARD.": :
620 PRINT "IF CORRECT, THAT NUMBER OF"
630 PRINT "BLOCKS IS DRAWN AND YOU ARE"
640 PRINT "GIVEN ANOTHER NUMBER. IF NOT"
650 PRINT "CORRECT, THE COMPUTER WILL"
660 PRINT "ASK YOU FOR ANOTHER ANSWER.": :
670 PRINT "TO STOP THE GAME, PRESS THE"
680 PRINT "SPACE BAR WHEN ASKED FOR AN"
690 PRINT "ANSWER.": : :
700 PRINT "PRESS ANY KEY TO PLAY."
710 CALL KEY(0,KEY,S)
720 IF S=0 THEN 710
730 CALL SOUND(100,1000,3)
740 REM PLAY GAME
750 CALL CLEAR
760 CALL SCREEN(14)
770 M$="LEARN THE NUMBERS"
780 XM=8
790 YM=3
800 GOSUB 1440
810 RANDOMIZE
820 K=INT(RND*10)
830 IF TM=K THEN 820
840 TM=K
850 CC=135
860 R=8
870 C=15
880 GOSUB 1500
890 M$="PRESS THIS NUMBER .."
900 XM=6
910 YM=13
920 GOSUB 1440
930 REM CHECK ANSWER
940 CALL KEY(0,KEY,S)
950 IF S=0 THEN 940
960 IF ((KEY<48)+(KEY>57))*(KEY<>32)THEN 940
970 CALL SOUND(100,1000,3)
980 IF KEY=32 THEN 1000 ELSE 1020
990 REM GAME ENDS
1000 CALL CLEAR
1010 STOP
1020 CALL HCHAR(13,27,KEY)
1030 IF (KEY-48)<>K THEN 1270
1040 REM CORRECT ANSWER
1050 GOSUB 1320
1060 M$=STR$(K)&" BLOCKS:"
1070 IF K<>1 THEN 1090
1080 M$="1 BLOCK:"
1090 XM=6
1100 YM=15
1110 GOSUB 1440
1120 IF K=0 THEN 1220
1130 BC=4
1140 FOR NB=1 TO K
1150 CALL HCHAR(18,BC,119,2)
1160 CALL HCHAR(19,BC,119,2)
1170 CALL SOUND(100,330,3)
1180 FOR D=1 TO 200
1190 NEXT D
1200 BC=BC+3
1210 NEXT NB
1220 FOR D=1 TO 2000
1230 NEXT D
1240 CALL HCHAR(8,15,32,370)
1250 GOTO 810
1260 REM INCORRECT ANSWER
1270 CALL SOUND(100,392,3)
1280 CALL SOUND(100,330,2)
1290 CALL HCHAR(13,27,32)
1300 GOTO 940
1310 REM PLAY TUNE
1320 FOR I=1 TO 2
1330 CALL SOUND(300,349,3,262,3,220,3)
1340 CALL SOUND(150,349,3)
1350 CALL SOUND(150,349,3)
1360 NEXT I
1370 CALL SOUND(300,440,3,349,3,262,3)
1380 CALL SOUND(150,523,3)
1390 CALL SOUND(150,523,3)
1400 CALL SOUND(300,440,3,349,3,262,3)
1410 CALL SOUND(300,349,3,262,3,220,3)
1420 RETURN
1430 REM TEXT PRINT SUBROUTINE
1440 FOR I=1 TO LEN(M$)
1450 C=ASC(SEG$(M$,I,1))
1460 CALL HCHAR(YM,XM+I-1,C)
1470 NEXT I
1480 RETURN
```

```
1490 REM NUMBER DRAWING SUBROUTINE
1500 L=CC
1510 FOR I=R TO R+2
1520 FOR J=C TO C+2
1530 CI=N(K,I-R+1,J-C+1)
1540 CALL CHAR(L,CI$(CI))
1550 CALL HCHAR(I,J,L)
1560 L=L+1
1570 NEXT J
1580 NEXT I
1590 RETURN
```

Program 2: 64 Number Game

Refer to the "Automatic Proofreader" article before typing this program in.

```
10 POKE53281,1:FORI=1TO12:READA:NEXT:GOSU
   B980                                  :rem 57
20 POKE53281,1:CO=2:LL=54272:FORI=LLTOLL+
   24:POKEI,Ø:NEXTI                      :rem 26
30 POKELL+5,1:POKELL+6,241:POKELL+24,15
                                         :rem 48
40 L8=INT(RND(1)*1Ø)+47                 :rem 232
50 GOSUB 43Ø                            :rem 124
60 GOSUB 14Ø                            :rem 123
70 POKE198,Ø                            :rem 148
80 GET A$:IFA$=""THEN8Ø                 :rem 243
90 IFA$=" "THENPOKE198,Ø:SYS198          :rem 21
100 IFASC(A$)<>L8THENGOSUB94Ø:GOTO8Ø
                                        :rem 21Ø
110 GOSUB 33Ø                           :rem 168
120 GOSUB 11ØØ                          :rem 213
130 FORI=1TO3ØØØ:NEXT:GOTO4Ø            :rem 232
140 RESTORE:FORI=7TO27STEP5              :rem 2Ø
150 CO=CO+1:IFCO>15THENCO=2             :rem 13Ø
160 PRINT"{HOME}{11 DOWN}"               :rem 54
170 POKE646,CO                           :rem 37
180 POKE249,I:READA:POKE25Ø,A           :rem 224
190 SYS828:PRINT"{UP}":NEXT              :rem 8Ø
200 FORI=1ØTO25STEP5                    :rem 219
210 CO=CO+1:IFCO>15THENCO=2             :rem 127
220 PRINT"{HOME}{15 DOWN}"              :rem 119
230 POKE646,CO                           :rem 34
240 POKE249,I:READA:POKE25Ø,A           :rem 221
250 SYS828:PRINT"{UP}":NEXT              :rem 77
260 FORI=12TO22STEP5                    :rem 224
270 CO=CO+1:IFCO>15THENCO=2             :rem 133
280 PRINT"{HOME}{19 DOWN}"              :rem 193
290 POKE646,CO                           :rem 4Ø
300 POKE249,I:READA:POKE25Ø,A           :rem 218
310 SYS828:PRINT"{UP}":NEXT:RETURN
                                        :rem 1ØØ
320 DATA 16,18,5,19,19,2Ø,8,9,19,11,5,25
                                        :rem 167
330 LL=54272:FORI=LLTOLL+24:POKEI,Ø:NEXT
                                          :rem 1
340 POKELL+24,15:POKELL+5,36:POKELL+6,132
                                        :rem 155
350 POKE54284,129:POKE54285,132         :rem 2Ø1
360 FORI=1TO8:READLB,HB                 :rem 172
370 POKELL,LB:POKELL+1,HB:POKELL+4,33
                                         :rem 32
380 POKE54279,LB:POKE5428Ø,HB:POKE54283,3
    3                                   :rem 178
390 FORK=1TO275:NEXT:POKELL+4,32:POKE5428
    3,32                                :rem 231
400 FORL=1TO15:NEXT:NEXT                 :rem 48
410 DATA 152,5,152,5,152,5,152,5,12,7,48,
    4,12,7,152,5                         :rem 24
420 RETURN                              :rem 118
430 PRINT"{CLR}";:POKE56334,PEEK(56334)AN
    D254:POKE1,PEEK(1)AND251            :rem 145
440 CO=CO+1:IFCO>15THENCO=2             :rem 132
450 POKE646,CO:L8=L8+1:M=53247+8*L8
                                        :rem 219
460 FORM1=M+1TOM+7:X=PEEK(M1):FORL=1TO7:C
    =146:X=X*2                          :rem 135
470 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334
    )OR1                                :rem 138
480 POKELL+4,16:Q=INT(RND(1)*4Ø):POKELL+1
    ,Q+(M1-M)*8                         :rem 253
490 IFX>255THENX=X-256:C=18:POKELL+4,17
                                         :rem 91
500 PRINTTAB(16)CHR$(C)CHR$(32);         :rem 56
510 POKE56334,PEEK(56334)AND254:POKE1,PEE
    K(1)AND251:NEXT:PRINT:NEXT          :rem 112
520 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334
    )OR1                                :rem 134
530 POKELL+4,16:RETURN                  :rem 1Ø7
540 POKE646,CO:PRINTTAB(16)CHR$(C);
                                         :rem 76
550 PRINT"{DOWN}{1Ø SPACES}PLEASE WAIT A
    {SPACE}MOMENT"                      :rem 124
560 T=Ø:FORJ=688TO7Ø3:READK:T=T+K:POKEJ,K
    :NEXT                               :rem 192
570 IFT<>3Ø78THENPRINT"ERROR IN DATA STAT
    EMENTS":STOP                        :rem 142
580 T=Ø:FORJ=828TO1ØØ6:READK:T=T+K:POKEJ,
    K:NEXT                              :rem 235
590 IFT<>2Ø3Ø6THENPRINT"ERROR IN DATA STA
    TEMENTS":STOP                       :rem 185
600 POKE249,Ø:RETURN                    :rem 218
610 DATA32,188,19Ø,226,172,225,191,251
                                        :rem 133
620 DATA187,255,161,236,162,254,252,96
                                        :rem 145
630 DATA 169,2Ø8,133,ØØ4,173,Ø24         :rem 37
640 DATA 2Ø8,41,2,24Ø,4,169              :rem 43
650 DATA 216,133,4,169,Ø,162             :rem 94
660 DATA 3,6,25Ø,42,2Ø2,2Ø8              :rem 38
670 DATA 25Ø,24,1Ø1,4,133,4              :rem 33
680 DATA 165,25Ø,133,3,173,14           :rem 145
690 DATA 22Ø,41,254,141,14,22Ø          :rem 184
700 DATA 165,1,41,251,133,1              :rem 31
710 DATA 169,Ø,133,25Ø,169,5             :rem 97
720 DATA 133,2,16Ø,Ø,177,3              :rem 241
730 DATA 133,5,23Ø,3,177,3              :rem 246
740 DATA 133,6,23Ø,3,198,2              :rem 25Ø
750 DATA 24Ø,28,162,Ø4,169,Ø             :rem 95
760 DATA 6,6,42,6,6,42                   :rem 55
770 DATA 6,5,42,6,5,42                   :rem 54
780 DATA 164,25Ø,153,48,2,23Ø           :rem 147
790 DATA 25Ø,2Ø2,2Ø8,232,24Ø,21Ø         :rem 27
800 DATA 165,1,9,4,133,1                :rem 144
810 DATA 173,14,22Ø,9,1,141              :rem 35
820 DATA 14,22Ø,16Ø,Ø,166,249           :rem 14Ø
830 DATA 24Ø,8,169,29                    :rem 18
840 DATA 32,21Ø                         :rem 218
850 DATA 255,2Ø2,2Ø8,25Ø,169,4          :rem 2ØØ
860 DATA 133,6,185,48,2,17Ø              :rem 53
870 DATA 189,176,2,133,5,41              :rem 56
880 DATA 64,24Ø,5,169,18,32              :rem 57
890 DATA 21Ø,255,165,5,41,191           :rem 151
900 DATA 32,21Ø,255,169,146,32          :rem 195
910 DATA 21Ø,255,2ØØ,198,6,2Ø8          :rem 195
920 DATA 221,169,13,32,21Ø,255          :rem 19Ø
930 DATA 192,16,2Ø8,196,96               :rem 18
940 POKELL+4,33:POKELL+1,1Ø:POKELL,143
                                          :rem 4
```

Robert Alonso, Assistant Editor

Logical Dreams

"If" is one of the most useful words in our vocabulary; it allows us to test situations and make appropriate decisions based on the test. Likewise, the BASIC command IF is one of the most useful words in the computer's vocabulary, and for the same reasons. Computer programs always rely on logical decisions to produce a result. Everything from data processing applications to arcade-style games relies on IF-THEN testing. The format is pretty simple: *if* something is true, *then* do something in response. For example, IF *joystick is pushed up*, THEN *move spaceship up*.

Using The Right IF

The IF-THEN statement can often be replaced with the statement IF-GOTO. Before mixing the statements or replacing one with the other in your programs, you must first understand the nature of each. IF-THEN is the most convenient and safest to use of the two. The reason for this is that almost any instruction placed after the THEN will be executed without any problems. You can place a line number after the THEN and the program will go to that line number, or you can place an expression such as A=A+1 and the program will execute it. The IF-THEN statement can thus be a very powerful and useful part of your programs. IF-GOTO is not as versatile as IF-THEN because it can only execute line numbers after the GOTO. If you tried placing an expression such as the previously mentioned A=A+1, the computer would flag it as an error.

IBM's Double GOTO

The Apple, Atari, and Commodore computers all flagged Program 1 as having an error in the line containing the IF-GOTO. The only two computers tested that did not flag it as an error were the IBM PC and PCjr. The IBM computers allowed the expression A=A+1 after an IF-GOTO and also allowed the following line:

```
20 IF A=1 GOTO GOTO 40
```

The double GOTO was allowed only if a line 40 had been entered and only after the IF statement. A program with a line number followed by double GOTOs and a target line number resulted in an error. This kind of rule bending is atypical of IBM. Just for reference you should know that the second edition (May 1982) of the BASIC manual by IBM and Microsoft states: "If the expression is true (not zero), the THEN or GOTO clause is executed. THEN may be followed by either a line number for branching or one or more statements to be executed. GOTO is always followed by a line number."

Although IBM may let you get away with an expression after an IF-GOTO, you should try to avoid such a construction within your programs. It is not standard and can produce errors and plenty of confusion. It is probably better to use only the IF-THEN statement because it allows either an expression or a line number and works the same on all the tested computers.

Sometimes an IF-THEN construction alone is not enough. In some situations, a structure called IF-THEN-ELSE can be useful. This structure is quite similar, but allows you to specify two THEN outcomes (one that's triggered by the IF and one that's triggered by an implied "IF NOT"). In other words, if the condition following the IF is true, whatever follows the THEN is carried out. If it is false, whatever follows the ELSE is carried out.

The Missing ELSE

However, this IF-THEN-ELSE construction is almost never used in programs published in magazines and books. The reason for this is not that there is something better, but that many home computers (Apple, Atari, and Commodore) do not have an ELSE command as part of their BASIC. IBM is one of the few that do allow IF-THEN-ELSE. The TI *Extended BASIC* cartridge also allows it.

There is a way to mimic IF-THEN-ELSE. Let's say that you want to test if a variable is equal to 100 and you want the THEN to end the program if it is. Otherwise, you want an ELSE to add 1 to the variable and let the program continue. Program 2 is an example of a routine that will do just that, without the ELSE command.

Imitating IF-THEN-ELSE

The IF-THEN-ELSE construction is in lines 30 and 40. The reason this works is that if the IF-THEN in line 30 is false, program execution "falls through" to line 40. The line following an IF-THEN can thus be used for the ELSE. There are some extra precautions that you should take. If the IF-THEN in line 30 had an expression (like A=A+1) instead of the END instruction, the program would execute the expression and then go to the next line and execute the line which you are using as an ELSE. This must be avoided or your program will not work properly. Program 3 is an example of how to properly mimic an IF-THEN-ELSE.

Take a look at the differences between Programs 2 and 3. The GOTO 50 in line 30 of Program 3 prevents the program from going on to line 40 when the IF condition is true. You should always include a GOTO with a target line number at the end of your IF-THEN if you are going to create the IF-THEN-ELSE construction. It is the only way to insure that the ELSE condition will not be executed haphazardly.

Program 1: IF-GOTO Error Demo

```
10 A=1
20 IF A-1 GOTO A=A+1
30 PRINT A
```

Program 2: IF-THEN-ELSE Construction

```
10 A=0: B=0: REM INITIALIZE
20 B=B+A: REM EXPRESSION
30 IF A=100 THEN END: REM IF THEN
40 A=A+1: GOTO 20: REM ELSE
```

Program 3: Better IF-THEN-ELSE

```
10 A=0: B=0: REM INITIALIZE
20 B=B+A: REM EXPRESSION
30 IF A=100 THEN PRINT B: GOTO 50: REM IF THEN
40 A=A+1: GOTO 20: REM ELSE
50 END
```

©

SEGA
THE ARCADE WINNERS

Arcade Classics Come To The Commodore 64

STAR TREK
STRATEGIC OPERATIONS SIMULATOR
THE VIDEO GAME

• Official arcade version.

Now you can control the Starship Enterprise as you fight Klingons before they destroy your bases. Full 3-D view as well as overhead radar lets you know what's going on around you. Use your joystick to control warp drive, impulse power, photon torpedoes and phasers. But watch your shields, photon supply and warp power. Fantastic graphics and sound make this a must for everyone. List $39.95. **Sale $29.95** (cartridge).

Congo Bongo

• Official arcade version • 3-D graphics
• Two screen displays • One or two players.

The famous arcade game featuring the coconut throwing gorilla, monkeys, hippos, sharks and rhinos now comes to the Commodore 64. Superb reproduction of the arcade machine challenges you to destroy the gorillas lair. List $39.95. **Sale $29.95** (cartridge).

ZAXXON

• Official arcade version • Stunning 3-D scrolling graphics • Multiple screen displays.

Now the famous invasion of robot Zaxxon's lair can be done in your own home. Invade the fortress, get past the fighters then through Zaxxon's home and finally destroy Zaxxon himself. Just like the arcade game. This one's fantastic. List $39.95. **Sale $29.95** (cartridge).

BUCK ROGERS
PLANET OF ZOOM

• Official arcade version • 3-D color graphics
• Four Galactic screen displays.

Just like the arcade version. Steer your ship through deadly electron posts as you fight off enemy space saucers and hoppers. Finally you meet the enemy source ship. If you get through all this you start again only this time the enemies have more powers and surprises. Fantastic graphics and sound. List $39.95. **Sale $29.95** (cartridge).

Add $3.00 for shipping, handling and insurance. Illinois residents please add 6% tax. Add $6.00 for CANADA, PUERTO RICO, HAWAII, ALASKA, APO-FPO orders. Canadian orders must be in U.S. dollars. WE DO NOT EXPORT TO OTHER COUNTRIES.
Enclose Cashiers Check, Money Order or Personal Check. Allow 14 days for delivery, 2 to 7 days for phone orders. 1 day express mail!
VISA — MASTER CARD — C.O.D.
No C.O.D. to Canada, APO-FPO.

PROTECTO ENTERPRIZES WE LOVE OUR CUSTOMERS
BOX 550, BARRINGTON, ILLINOIS 60010
Phone 312/382-5244 to order

C. Regena

Algebra Tutorial

Part 1

We have examined educational software in previous columns and discussed how to construct drill programs. Now let's create a tutorial program. There have been a lot of requests for an educational program for algebra so here is the first part of a tutorial program on multiplying binomials.

"Algebra Tutorial," assumes the student has some knowledge of algebra and understands terms usually introduced before binomial multiplication. This program only covers multiplication of one binomial (numeric expression of two terms) by another binomial—such as (x + 5) times (x + 4). Additional related units could include multiplying polynomials, dividing polynomials by binomials, and factoring trinomials.

The program uses PRINT statements to avoid DATA statements with lots of numbers. If you prefer to prevent scrolling, you can use the graphics method of CALL HCHAR and CALL VCHAR to print problems on the screen.

Redefining Characters

Lines 160 and 170 redefine two characters for use in printing the problems. Character 94 is ordinarily the caret or exponentiation symbol, but is redefined here as a 2, which will be used as the superscript for a number squared. To type the program in, use SHIFT 6 to get the ^ symbol in lines such as line 400.

The underline is also redefined. Character 95 is ordinarily the underline, but several underlines together yield a dotted line, and we want a solid line. Lines 230 and 270 are examples of the underline in the listing. To type the underline, press the FCTN key and the U. As you type the listing, you will see the regular symbols, but when you run the program, you will see the redefined characters.

When learning algebra, it is important to understand that you can work with letters using the same rules and methods that are used with regular numbers. Lines 190–300 print a screen showing a comparison of binomial multiplication in algebra with a numeric multiplication problem. Lines 310–460 show the general form of the multiplication problem and its answer.

Generating A Random Problem

Lines 470–950 present a problem for the student to try. A and B are two random numbers chosen for the second terms of the binomials. This problem is the simple case using X plus a number from 1 to 3. The computer goes through the problem step by step, and the student presses a number where prompted. Correct numbers must be entered to continue.

CALL KEY is used rather than INPUT, so the student just needs to press a key for the answer. If you use INPUT, there is a greater chance for user error or for the program to crash. Avoid INPUT in tutorials so the student can use the program as easily as possible.

The tutorial adds new information a little at a time. Lines 960–1110 present a screen showing numeric coefficients for the first term. Lines 1120–1180 (and the subroutine starting at line 1960) give the student a problem of this type. Lines 1190–1300 present a screen of information about using positive and negative numbers.

```
110 CALL CLEAR
120 PRINT "  BINOMIAL MULTIPLICATIO
    N"
130 PRINT ::"THIS PROGRAM DISCUSSES
    "
140 PRINT :"MULTIPLICATION OF BINOM
    IALS"
150 PRINT :"SUCH AS (X+5) TIMES (X+
    3).":::::
160 CALL CHAR(94,"0000304808102078")
170 CALL CHAR(95,"00000000000000FF")
180 GOSUB 1530
190 CALL SCREEN(8)
200 PRINT "COMPARE":"  ALGEBRA TO":
    "  REGULAR MULTIPLICATION:"
210 PRINT ::"{3 SPACES}12";TAB(21);
    "X + 2"
220 PRINT :"{3 SPACES}23";TAB(21);"
    X + 3"
230 PRINT "  ___";TAB(20);"______"
240 PRINT :"{3 SPACES}36";TAB(20);"
    3X + 6"
250 PRINT TAB(16);"^"
260 PRINT "  24";TAB(15);"X  + 2X"
270 PRINT "  ___";TAB(15);"________
    "
280 PRINT TAB(16);"^"
290 PRINT "  276";TAB(15);"X  + 5X
    + 6"::::
300 GOSUB 1530
310 CALL SCREEN(4)
320 PRINT "IN GENERAL,"
330 PRINT :TAB(15);"X  +  A"
340 PRINT :TAB(15);"X  +  B"
350 PRINT TAB(12);"__________"
360 PRINT :TAB(14);"BX  + AB"
370 PRINT TAB(7);"^"
380 PRINT TAB(6);"X  +{4 SPACES}AX"
390 PRINT TAB(6);"________________"
400 PRINT TAB(7);"^"
410 PRINT TAB(6);"X  + (A+B)X + AB"
420 PRINT :::"THE FIRST TERM IS X*X
    "
430 PRINT "THE LAST TERM IS A*B"
440 PRINT "THE MIDDLE TERM COMBINES
    "
450 PRINT "A AND B MULTIPLIED BY X"
460 GOSUB 1530
470 CALL CLEAR
480 CALL SCREEN(8)
490 PRINT "NOW YOU MULTIPLY:"
500 RANDOMIZE
510 A=INT(3*RND)+1
520 B=INT(3*RND)+1
530 F=0
540 PRINT :TAB(22);"X +";A
550 PRINT :TAB(22);"X +";B
560 PRINT TAB(21);"______"
570 PRINT :B;"TIMES TOP";TAB(21);"?
    X +"
580 C=23
590 GOSUB 1620
600 IF K=48+B THEN 630
610 GOSUB 1580
620 GOTO 590
630 C=28
640 GOSUB 1620
650 IF K=48+A*B THEN 680
660 GOSUB 1580
670 GOTO 640
680 PRINT TAB(17);"^"
690 PRINT " X TIMES TOP";TAB(16);"X
    +  X"
700 C=23
710 GOSUB 1620
720 IF K=48+A THEN 750
730 GOSUB 1580
740 GOTO 710
750 PRINT TAB(16);"___________"
760 PRINT TAB(17);"^"
770 PRINT " ADD";TAB(16);"X  +  X +"
780 GOSUB 1620
790 IF K=A+B+48 THEN 820
800 GOSUB 1580
810 GOTO 780
820 C=28
830 GOSUB 1620
840 IF K=A*B+48 THEN 870
850 GOSUB 1580
860 GOTO 830
870 GOSUB 1690
880 IF F=0 THEN 910
890 GOSUB 1530
900 GOTO 470
910 PRINT :::"CHOOSE:  1 ANOTHER PR
    OBLEM"
920 PRINT TAB(10);"2 CONTINUE PROGR
    AM"
930 CALL KEY(0,K,S)
940 IF K=49 THEN 470
950 IF K<>50 THEN 930
960 CALL CLEAR
970 CALL SCREEN(12)
980 PRINT "THERE MAY BE COEFFICIENT
    S"
990 PRINT "OF THE FIRST TERM,"
1000 PRINT "BUT THE RULES DON'T CHA
     NGE."
1010 PRINT ::"FOR EXAMPLE,"
1020 PRINT :TAB(15);"2Y + 5"
1030 PRINT :TAB(15);"3Y + 1"
1040 PRINT TAB(15);"______"
1050 PRINT :TAB(15);"2Y + 5"
1060 PRINT TAB(10);"^"
1070 PRINT TAB(8);"6Y  + 15Y"
1080 PRINT TAB(8);"_____________"
1090 PRINT TAB(10);"^"
1100 PRINT TAB(8);"6Y  + 17Y + 5":::
1110 GOSUB 1530
1120 CALL SCREEN(8)
1130 T=1
1140 SD=1
1150 SD$="+"
1160 SE=1
1170 SE$="+"
1180 GOSUB 1960
1190 CALL CLEAR
1200 CALL SCREEN(4)
1210 PRINT "BINOMIALS MAY CONTAIN"
1220 PRINT :"+ OR - NUMBERS."
1230 PRINT ::"MULTIPLY THE NUMBERS."
1240 PRINT :"AND REMEMBER THE RULES"
1250 PRINT :"FOR THE SIGNS."
1260 PRINT ::"{3 SPACES}+ * + = +"
1270 PRINT :"{3 SPACES}+ * - = -"
1280 PRINT :"{3 SPACES}- * + = -"
1290 PRINT :"{3 SPACES}- * - = +"
1300 GOSUB 1530
```

Next month, we'll present the remainder of the program.

TI Disassembler

James Dunn

Since information on the operating system and BASIC interpreter used by the TI-99 is scarce, "TI Disassembler" will come in handy if you want to try your hand at programming in TI-9900 machine language.

A disassembler converts the jumble of numbers that actually constitute a machine language program into a more readily understandable form. For each machine language instruction (called an opcode), TI has established a one- to four-letter representation called a *mnemonic.* This disassembler decodes the contents of memory into standard TI mnemonics, making ML programs less difficult to understand. However, this program will not teach you machine language programming. To use this program, you must have at least an elementary understanding of TI machine language and a familiarity with TI's standard format for ML assemblers. Refer to any of the several books on this subject for further information.

This Disassembler is written in Extended BASIC. However, it can be easily translated for the *Mini Memory* or *Editor/Assembler* cartridges. All that is necessary is to unstack the lines so that there is only one statement on a line. All the commands can be found in console BASIC except the PEEK command which is in Extended BASIC, and also available when the *Mini Memory* or *Editor/Assembler* cartridge is installed.

Printer Output

Depending upon your printer setup, you may have to modify line 110 or the subroutine starting on line 860, which prints to the screen. It might be wiser to leave that routine as is and just add the extra lines necessary to output to your printer.

Notice that all computations and input are in decimal. If you want hexadecimal numbers, you can modify the program to add conversions. Be warned, however, that this will slow down the program. When you are disassembling 16K blocks, that can be something to think about.

The Disassembler does an excellent job on machine language programs; however, it has one weakness. It cannot tell if the area of memory you ask it to disassemble contains data, text, or jump tables. It will attempt to disassemble these as if they were legitimate opcodes. To tell if this is happening, watch for the BYT output, which indicates that the area you are disassembling contains something other than machine language.

Where You Can't PEEK

The Disassembler can only look into the CPU address space. This is a fault of the architecture of the computer itself. Since the 16K RAM area used by console BASIC is not connected to the CPU, but rather to the VDP (Video Display Processor), the Disassembler cannot access it. Also unreadable are the GROMs which contain the GPL. If you have expansion memory, it is accessible, as are the command modules. Both the *Mini Memory* and the *Editor/Assembler* cartridges provide PEEK and POKE commands which can access these areas.

In order to be consistent with TI machine language conventions, the Disassembler uses the same field symbols and addressing mode symbols used in the TI *Editor/Assembler* package. In case you don't have that package, Tables 1 and 2 show the symbols.

Explanation Of Program

30–110	Initialization and input.
120	Start of main loop.
140	PEEK locations.
150–260	Determines the format of the opcode and sends program to appropriate line number for decoding.
270–370	Decodes Format VIII opcodes.
380–420	Decodes Format VI opcodes.
430–450	Decodes Format V opcodes.
460–530	Decodes Format II opcodes.
540–590	Decodes Format IV opcodes.
600–680	Decodes Format III and IX opcodes.
690–780	Decodes Format I opcodes.
790–810	Decodes Format VII opcodes.

820	If not one of the above, byte is not a valid opcode.
840	Optional sound signal and hold when no opcode found.
860	Print to screen routine.
900	Subroutine to READ DATA and pick out mnemonic.
930	Subroutine to decode the Ts address mode.
1000–	DATA statements which contain mnemonics listed according to their Format.

Variables Used

A	Start address
A1	Temporary variable to cover quirk of PEEK statement
A$	Opcode
B	End address
B$	Source field
C$	Destination field
H	High byte of PEEK address
I	Temporary loop variable
J	Base to which value K is added
J$	Want printout
K	Displacement variable for loop
L	Low byte of PEEK address
N	Computed total of H and L
01	next bytes in order after L
02	next bytes in order after L
03	next bytes in order after L
04	next bytes in order after L
PR	Printout variable (0 = no, 1 = y)
Q$	Temporary storage for txfr to A$
R	Register number
TR	Loop indicator
Z	Number of opcodes in format type

Table 1: TI Opcode Field Symbols

CO	Count
D	Destination operand
NU	Number
S	Source operand
Td	Specific address mode of destination operand
Ts	Specific address mode of source operand
WR	Workspace register

Table 2: TI Addressing Mode Symbols

*****	means Indirect address mode
(R)	means Indexed address mode
+	after * means Auto Increment address mode
#	means Workspace Register address mode
@	means Direct address mode

TI Disassembler

```
30 REM INITIALIZE
40 TR=0 :: CALL CLEAR :: PR=0
50 PRINT "START ADDRESS (MUST BE AN
   EVEN DECIMAL NUMBER )?" :: INPU
   T A
60 IF A=0 THEN 80
70 IF A/2<>INT(A/2)THEN 50
80 PRINT "END ADDRESS ?" :: INPUT B
90 PRINT "DECODE FROM : ";A;" TO :
   ";B
100 PRINT "WANT PRINTOUT?" :: INPUT
    J$
110 IF J$="Y" THEN PR=1 :: CLOSE #1
    :: OPEN #1:"RS232",OUTPUT
120 IF A>=B THEN 50
130 A1=A :: IF A>32767 THEN A1=A-65
    536
140 CALL PEEK(A1,H,L,01,02,03,04)
150 REM TEST FOR OP CODES & ADDRESS
    MODES
160 N=H*256+L :: IF N>16383 THEN 69
    0
170 IF N>14335 THEN 600
180 IF N>12287 THEN 540
190 IF N>11263 THEN 600
200 IF N>8191 THEN 600
210 IF N>4095 THEN 460
220 IF N>2047 THEN 430
230 IF N>1023 THEN 380
240 IF N>831 THEN 790
250 IF N>511 THEN 270
260 GOTO 820
270 REM FORMAT VIII OP-CODES
280 IF (L AND 16)=16 THEN 820
290 RESTORE 1020 :: J=480 :: Z=5 ::
     K=32 :: R=(L AND 15):: N=((H A
    ND 3)*256)+(L AND 224):: GOSUB
    900
300 IF TR<>1 THEN 330
310 C$=STR$(01*256+02):: A=A+4
320 B$="R"&STR$(R)&"," :: GOTO 370
330 Z=2 :: GOSUB 900 :: IF TR<>1 TH
    EN 350
340 C$="" :: A=A+2 :: B$="R"&STR$(R
    ):: GOTO 370
350 Z=2 :: GOSUB 900 :: IF TR<>1 TH
    EN 820
360 B$=STR$(01*256+02):: A=A+4 :: C
    $=""
370 GOSUB 860 :: TR=0 :: GOTO 120
380 REM FORMAT VI OP-CODES
390 N=(H*256)+(L AND 192):: J=960 :
    : Z=14 :: K=64 :: RESTORE 1000
    :: GOSUB 900
400 GOSUB 930
410 C$="" :: IF A$="B" AND B$="*R11
    " THEN C$="(SAME AS RTS)"
420 GOSUB 860 :: TR=0 :: GOTO 120
430 REM FORMAT V OP-CODES
440 N=(H AND 11)*256 :: J=1792 :: C
    O=(L AND 240):: WR=(L AND 15)::
     RESTORE 1040 :: Z=4 :: K=256 :
    : GOSUB 900
450 B$="R"&STR$(WR)&"," :: C$=STR$(
    CO):: A=A+2 :: GOSUB 860 :: TR=
    0 :: GOTO 120
460 REM FORMAT II OP-CODES
470 RESTORE 1050 :: J=3840 :: TR=0
    :: Z=13 :: K=256 :: N=H*256 ::
    GOSUB 900
480 IF TR=0 THEN 500
490 B$=STR$(2*L+2):: GOTO 520
500 Z=3 :: K=256 :: GOSUB 900
510 B$=STR$(L)
520 C$="" :: A=A+2 :: IF A$="JMP" A
    ND B$="2" THEN C$="(SAME AS NOP
    )"
530 GOSUB 860 :: TR=0 :: GOTO 120
540 REM FORMAT IV OP-CODES
550 IF (H AND 252)=48 THEN A$="LDCR
    " :: GOTO 580
```

```
560 IF (H AND 252)=52 THEN A$="STCR
    " :: GOTO 580
570 GOTO 820
580 GOSUB 930 :: NU=((H AND 3)*4)+(
    (L AND 192)/64)
590 C$=","&STR$(NU):: GOSUB 860 ::
    TR=Ø :: GOTO 12Ø
600 REM FORMAT III & IX OP-CODES
610 RESTORE 1070 :: J=7168 :: N=((H
     AND 60)*256):: D=((H AND 3)*4)
    +((L AND 192)/64):: Z=3 :: K=10
    24 :: GOSUB 900
620 IF TR<>1 THEN 650
630 C$=","&"R"&STR$(D)
640 GOSUB 930 :: GOSUB 860 :: TR=Ø
    :: GOTO 120
650 IF N<>11264 THEN 670
660 A$="XOP" :: C$="XOP OP #"&STR$(
    D):: GOTO 640
670 J=13312 :: Z=2 :: K=1024 :: GOS
    UB 900 :: IF TR<>1 THEN 820
680 GOTO 640
690 REM FORMAT I OP-CODES
700 RESTORE 1080 :: J=12288 :: N=(H
     AND 224)*256 :: Z=12 :: K=4096
     :: GOSUB 900
710 TD=(H AND 12):: D=((H AND 3)*4)
    +((L AND 192)/64):: GOSUB 930
720 IF TD=Ø THEN C$=","&"R"&STR$(D)
    :: GOTO 770
730 IF TD=4 THEN C$=","&"*R"&STR$(D
    ):: GOTO 770
740 IF TD=12 THEN C$=","&"R"&STR$(S
    )&"+" :: GOTO 770
750 IF (TD=8)AND(D=Ø)THEN C$=","&"@
    "&STR$(O1*256+O2):: A=A+2 :: IF
     TS=32 THEN C$=","&"@"&STR$(Ø3*
    256+O4)
760 IF (TD=8)AND(D<>Ø)THEN C$=",@"&
    STR$(O1*256+O2)&"(R"&STR$(S)&")
    " :: A=A+2 :: IF TS=32 THEN C$=
    ",@"&STR$(O3*256+O4)&"(R"&STR$(
    S)&")"
770 GOSUB 860 :: TR=Ø :: GOTO 120
780 IF (TD=8)AND(D=Ø)THEN C$=",@"&S
    TR$(O1*256+O2):: A=A+2 :: IF TS
    =32 THEN C$=",@"&STR$(Ø3*256+O4
    )
790 REM FORMAT VII
800 N=(H*256+L):: A=A+2 :: B$="" ::
     C$="" :: Z=6 :: J=800 :: K=32
    :: RESTORE 1100
810 GOSUB 900 :: GOSUB 860 :: TR=Ø
    :: GOTO 120
820 REM NOT OP-CODE
830 A$="BYTE" :: B$=STR$((H*256)+L)
    :: C$=CHR$(H)&"  "&CHR$(L):: A=A
    +2 :: GOSUB 860
840 REM CALL SOUND(800,400,Ø):: ACC
    EPT Q4$ :: GOTO 97
850 GOTO 120
860 REM PRINT ROUTINE
870 PT$=STR$(A1)&"   "&A$&"  "&B$&C$
    :: PRINT PT$ :: A1=A
880 IF PR=1 THEN PRINT #1;TAB(10);P
    T$
890 RETURN
900 REM FIND OP-CODE FROM DATA
910 FOR I=1 TO 7 :: J=J+K :: READ Q
    $ :: IF N=J THEN A$=Q$ :: TR=1
920 NEXT I :: RETURN
930 REM SUBROUTINE TS ADDRESS
940 S=(L AND 15):: TS=(L AND 48)::
    IF TS=Ø THEN B$="R"&STR$(S):: A
    =A+2 :: RETURN
950 IF TS=16 THEN B$="*R"&STR$(S)::
     A=A+2 :: RETURN
960 IF TS=48 THEN B$="*R"&STR$(S)&"
    +" :: A=A+2 :: RETURN
970 IF (TS=32)AND(S=Ø)THEN B$="@"&S
    TR$(O1*256+O2):: A=A+4 :: RETUR
    N
980 IF (TS=32)AND(S<>Ø)THEN B$="@"&
    STR$(O1*256+O2)&"(R"&STR$(S)&")
    " :: A=A+4 :: RETURN
990 BREAK
1000 DATA BLWP,B,X,CLR,NEG,INV,INC,
     INCT,DEC
1010 DATA DECT,BL,SWPB,SETO,ABS
1020 DATA LI,AI,ANDI,ORI,CI
1030 DATA STWP,STST,LWPI,LIMI
1040 DATA SRA,SRL,SLA,SRC
1050 DATA JMP,JLT,JLE,JEQ,JHE,JGT,J
     NE
1060 DATA JNC,JOC,JNO,JL,JH,JOP,SBO
     ,SBZ,TB
1070 DATA COC,CZC,XOR,MPY,DIV
1080 DATA SZC,SZCB,S,SB,C,CB,A,AB
1090 DATA MOV,MOVB,SOC,SOCB
1100 DATA IDLE,RSET,RTWP,CKON,CKOF,
     LREX
```

©

```
230 IF X1 THEN POKEH1,X1:POKEL1,Y1:POKEV1
    ,17
240 IF X2 THEN POKEH2,X2:POKEL2,Y2:POKEV2
    ,33
250 IF X3 THEN POKEH3,X3:POKEL3,Y3:POKEV3
    ,17
```

For each instrument: If its pitch is not zero, set the pitch and hit the note. You will see that we make the note sound by adding 1 to the waveform value. Compare these values with the ones shown in line 200, above.

```
260 T=T+S
270 IF T>TI GOTO270
280 GOTO200
```

We calculate the note's timing, and wait until the proper amount of time has passed. Then we go back and get the next note.

```
290 FOR J=L1 TO 54296:POKE J,Ø:NEXT J
295 PRINT CHR$(154):END
```

Finally, we clear all the SID music registers, change the printing color back to light blue, and stop.

Here come the DATA statements to play the music and write the words. Note that whenever a word ends with a period or comma, it will be printed and then a new line will be started.

```
300 DATA 40,"{2 SPACES}HAP",34,75,Ø,Ø,Ø,Ø
310 DATA 20,"PY",34,75,Ø,Ø,Ø,Ø
320 DATA 60," BIRTH",38,126,28,214,5,185
330 DATA 60,"DAY",34,75,28,214,Ø,Ø
340 DATA 60," TO",45,198,38,126,5,185
350 DATA 60," YOU",43,52,30,141,4,73
360 DATA 60,",",Ø,Ø,Ø,Ø,Ø,Ø
370 DATA 40,"{2 SPACES}HAP",34,75,Ø,Ø,Ø,Ø
380 DATA 20,"PY",34,75,Ø,Ø,Ø,Ø
390 DATA 60," BIRTH",38,126,30,141,6,108
400 DATA 60,"DAY",34,75,30,141,Ø,Ø
410 DATA 60," TO",51,97,34,75,4,73
420 DATA 60," YOU",45,198,28,214,5,185
430 DATA 60,",",Ø,Ø,Ø,Ø,Ø,Ø
440 DATA 40,"{2 SPACES}HAP",34,75,Ø,Ø,Ø,Ø
450 DATA 20,"PY",34,75,Ø,Ø,Ø,Ø
460 DATA 60," BIRTH",68,149,22,227,5,185
470 DATA 60,"DAY",57,172,25,177,Ø,Ø
480 DATA 60," DEAR",45,198,28,214,7,53
500 DATA 60," AN",21,154,30,141,7,163
510 DATA 60,"DREW",19,63,30,141,Ø,Ø
520 DATA 60,",",Ø,Ø,Ø,Ø,Ø,Ø
530 DATA 40,"{2 SPACES}HAP",61,126,Ø,Ø,Ø,
    Ø
540 DATA 20,"PY",61,126,Ø,Ø,Ø,Ø
550 DATA 60," BIRTH",57,172,34,75,8,147
560 DATA 60,"DAY",45,198,28,214,Ø,Ø
600 DATA 60," TO",51,97,30,141,4,73
610 DATA 60," YOU.",45,198,28,214,2,220
620 DATA Ø
1000 PRINTS$;:IF RIGHT$(S$,1)<"Ø"THENPRIN
     T
1010 RETURN
```

Finally, we see a subroutine at 1000 to print the word or part word, and to test if it ends in a nonalphabetic character. If so, a new line will be started. Be sure to include the semicolon after the PRINT statement in line 1000.

CAPUTE!

Modifications Or Corrections To Previous Articles

TI Jackpot

Our lister program garbled characters in several graphics definition lines of the TI-99/4A version of this program from the August issue (p. 83). Several readers have noted that lines 660, 680, and 690 should read as follows:

```
660 DISPLAY AT(12,2)SIZE(25):"w"&CH
    R$(133)&CHR$(134)&"wwwwwwwwwwst
    ststwwwwww" :: DISPLAY AT(13,2)
    SIZE(25):"wJJw~w~w>w2wwuvuvuvw>
    w14w"
680 DISPLAY AT(15,2)SIZE(25):"wJJJJ
    w~w>w5ww¦}¦}<=w>w18w" :: DISPLA
    Y AT(16,2)SIZE(25):"wdede:;wwww
    wwz{z{z{wwwwww"
690 DISPLAY AT(17,2)SIZE(25):"wfgfg
    <=w>10ww¦}¦}¦}w>w18w" :: DISPLA
    Y AT(18,2)SIZE(25):"wdededewwww
    ww:;:;:;wwwwww"
```

Also, the space near the end of the string in line 440 (between the characters 1F and 1F) should be omitted.

VIC Lightsaver

The machine language for this program from the September issue (p. 96) is correct, but there are bugs in the version of "Tiny MLX" (p. 151) to be used to enter it. Lines 100 and 210 of Tiny MLX do not contain the proper values for "Lightsaver." Also, a change is necessary to line 763 to allow you to use BASIC's standard LOAD and RUN commands to activate Lightsaver. The corrected lines are as follows:

```
100 POKE 55,30:POKE 56,25:CLR:POKE 788,19
    4                                :rem 21
210 S=6430:E=7677                   :rem 135
763 POKE 780,1:POKE 781,DV:POKE 782,Ø:SYS
     65466                           :rem 68
```

64 Devastator

Readers using the "Automatic Proofreader" to check the BASIC portion of the 64 version of this game from the August issue (Program 7, p. 79) have noticed a problem with line 60. The error does not affect the operation of the program, but if you'd like the checksum for line 60 to match the one which appears in the magazine, add {7 RIGHT} after the {12 DOWN} in that line.