New Software, New Hardware Computer Languages, and Games

Jerry Pournelle
c/o BYTE Publications
70 Main St
Peterborough NH 03458

"Read any good books lately?" asked my mad friend Mac Lean.

Having just come back from an autograph party for *King David's Spaceship*, and the day before sent off the copyedited manuscript for the new Niven-Pournelle *Oath of Fealty*, I knew what to say. "Haven't had time. But I've *written* some good books lately...."

"Yeah, well, I wasn't talking about science fiction," said Mac Lean. "I meant good books on computers. I've got a dilly." He held up *PL/I : Structured Programming*, by Joan K Hughes.

"Hey, I know her," I said.

"Well, you tell her for me she's written a really top book. Good index. Clear English. Stand-alone chapters, so you don't have to thumb back and forth to find out what's going on."

"I'll do better than that," I said. "I'll tell my readers."

"Yeah. Sure. When?"

"Uh, real soon now...."

* * *

My apologies for taking so long in finishing this column, and my thanks to all of you who've written encouraging letters. Things do get hectic here at Chaos Manor, and the last few months have been something to see: books to get done, articles to write, and I built a new wing on the office, which meant moving everything around like Chinese Checkers, which meant that I lost the documentation to half the software sent me for review, which—

I am also, for my sins, Chairman of the Citizens Advisory Council on National Space Policy, which involves chairing meetings and editing papers and writing summaries and flying to Washington. The result was that for a good while I had no time to play with Ezekial, my Z80, but things are a bit caught up now, and maybe we can get onto a schedule. Just last week I had a surgeon remove the telephone from my ear.

One reason we got caught up was Spellguard.

Every now and then you find programs that do things right and have documentation that tells you what to do and how to do it—programs that are a joy to use. Spellguard is like that. (See the reviews of Spellguard, Microspell, the Word, Microproof, and Wordsearch on pages 434-448.) Spellguard finds spelling errors. That's all it does. It doesn't wash dishes, or set your clock, or do your taxes, but wow! can it find spelling errors in standard ASCII text files. And it doesn't care what editor you used to create the files.

Using Spellguard is simplicity itself. All you do is tell it what file to look at. It does the rest. It does *not* do it as fast as the ads say—at least it doesn't with Zeke, but then he's only running at 2 MHz, and my ancient iCom drives, while utterly reliable, are pretty slow. Help is on the way: I've got a pair of Lobo's 8-inch drives, and they've promised an S-100 double-density controller next month. If the controller is as fast and reliable as their drives, things are going to hum around here. We've been using Lobo drives (both 8-inch and 5¼-inch) on the boys' TRS-80 for months now, and we love them.

Meanwhile, Spellguard runs fine under Speed, a rather strange program available from the CP/M User's Group. Speed trades memory for rapid disk I/O, and if you have programs that don't use a lot of memory but have a lot of disk operations, Speed can cut running time dramatically. For Spellguard the saving is about 50%; the saving in posting my journals to ledgers is even more dramatic—70%.

Speed, with auxiliary programs to get it running properly, is available on Volume 38 of the CP/M User's group (CPMUG) disks. Those with access to a computer net may be able to find it in an on-line file. You need a bit of patching with DDT to install SPEED, but the documentation (also on CPMUG Disk 38) is clear enough.

One warning, though: Speed maintains the disk directories in RAM (random-access memory), and if you change disks without rebooting (in Speed that's control-B, rather than CP/M's control-C) the result is an unmitigated disaster!

But back to Spellguard. When you run Spellguard, it first gives you a table, telling you how many words it has read, the number and percentage of unique words, the number and percentage that it can't find in the dictionary, and finally a changing column that tells you the percentage of proofreading the program has done.

Many years ago, when I was young and impressionable, I fell under the spell of a science fiction writer named A E Van Vogt; later he became a friend, neighbor, and colleague. Van Vogt was (and still is) interested in a rather hard-to-define field of study called general semantics, and through him I was led to Alfred Count Korzybski and a strange book called *Science and Sanity*. That led me to Wendell Johnson at the University of Iowa. Professor Johnson's interests spanned everything from classical linguistics and speech therapy to general semantics.

One of Johnson's research interests was identifying text: how could you tell if an anonymous work had been written by a particular author? Dr Johnson used a number of quantitative measures, two of the most important being the type/token ratio and the verb/adjective ratio. Type/token means the ratio of unique words to total words; verb/adjective is self-explanatory. I remember going nearly blind counting total words and making tables of unique words in, for example, Marlowe's *Duchess of Malfi*; the idea was to find out if Marlowe had written any of the Shakespearean plays. (As best we could calculate, he hadn't.)

Now Spellguard gives you automatic type/token ratios, and if you really want to, you can make up separate verb and adjective dictionaries, thus finding the ratio merely by typing in the text, which, believe me, is a great deal easier than doing it in teams of two with pencil and paper. Spellguard, with its efficient search algorithms, could be extremely useful in linguistic research projects.

Anyway, if you work with text at all, you'll love Innovative Software's Spellguard. It is an example of excellent software: a program that does one thing, does it very well, and has documentation to match.

Datebook and Milestone

Organic Software of Livermore, California, has also produced two excellent programs that I recommend, although I'll probably never use them.

The first, Datebook, can keep track of about six months' worth of appointments for three people. We had no trouble getting it to run, and it seems to be easy to use. The main drawback is that you have to *want* to run it; Datebook requires both disk drives and all of your microcomputer's memory. What I want is Calendar, a program that I may have to write myself; it would come up when I turn on my system, insist that I give it the date (the way Lobo's LDOS operating system does), and then natter at me about what I have to get done. But it's fairly obvious that I can't write *that* program until I have hard disks with multimegabyte storage.

Calendar would solve my problem: I forget to look at my appointment book until it's too late. And that raises a question: is Datebook really better than an appointment book of the kind used by many physicians and lawyers? I can't answer that, but my guess is that I wouldn't buy a computer for that *alone*. Datebook has various search patterns, so you can look for appointment openings of stated lengths, and the program offers up to nine candidates—but you can do that by glancing at a book, too.

The value of Datebook is that it will keep three schedules simultaneously, so you can work appointments with your partners (it searches for times when you're all free). It will also search through and find all the appointments you've made with a particular person—a task that is more difficult and certainly more tedious if you must rely on visual inspection of a book (especially if there are a lot of entries). And of course Datebook can make hard copy, and update that often. All in all, if I worked in a business where I had lots of appointments and schedules, I'd probably use Datebook, but then I'm gadget-oriented—and I *have* a computer.

Organic's other interesting program is Milestone, and people who need it will like it a lot. Milestone is a PERT-chart generator. It performs critical-path analysis for jobs

with up to 300 tasks; it computes milestones (critical events), monthly manpower levels, monthly costs—in short, it handles most of the details we used to have to include in the management-plan portion of a research proposal.

Milestone isn't easy to use. In fact, it's almost impossible to use on my system because I don't have a 24 by 80 terminal. I use a memory-mapped 16 by 64 video display, in part because I still use the old-fashioned Electric Pencil as a text editor, and Pencil requires it, in part because I like the smooth scrolling and fast response, and in part because I'm lazy and have never gotten around to buying a terminal. That's going to change one day.

Anyway, Milestone isn't easy to use, not because the directions for the *program* aren't clear (they're only fair), but because PERT charting and critical-path analysis are more arts than sciences, and not easy jobs. Milestone can make the jobs easier, and if I ever again have to generate research proposals, I'll certainly use Milestone.

One of my pet peeves is documentation without examples. I can't imagine why people write instructions on how to use a program but fail to include specific illustrations of what command to issue and the results.

Let me illustrate. Suppose I am opening Milestone after not looking at it for a couple of weeks (as indeed I am). I turn to page 36, and there I find:

> "B(egin) and E(nd) work
> These two values define when the normal working day begins and ends. They must be even hours as defined on a 24-hour clock. Follow the rules for entering integers rather than times."

Now I ask you, what does this mean? Presumably, I once knew, but I confess I've forgotten. Indeed, some of us go days on end without even thinking about computers!

I don't want to be too hard on Organic; Milestone, in fact, comes with a set of example cases. One is Dr Victor Frankenstein III's PERT for creating a monster. Event One, "Fanatic Desire to Create Life," is a milestone; it has no duration, but you don't start without it. We proceed to Task Two, "Move to ancestral castle," and continue searching for Grandfather's notes, hiring a linguist, etc. As it happens, Organic ran this example as part of a demonstration for me at the San Francisco Computer Faire, and I can testify that Milestone is both fast and accurate. I thought up the need for a linguist to translate the notes. That broke into subtasks: advertise, interview, hire. I watched Burns VanHorne of Organic enter the new tasks. Milestone thought for a moment, then rearranged itself, because this information became part of the critical path. (Not long ago they sent a revision of Milestone and I notice that the tasks I suggested are now in the case study.)

So they have examples, in the sense of worked-out problems for their program, but they should have in-

cluded specific examples all over their manual—and so should every other publisher. Please?

Debate on Languages

My last column, on languages, generated a lot of correspondence. Some was predictable: I wasn't sufficiently respectful of LISP, the LISt Processing language written in 1956 by my friend John McCarthy, and since improved and expanded by McCarthy and my collaborator, Marvin Minsky, and used by my friends and associates who wrote ZORK, and—in other words, please, I don't dislike LISP users.

What I said was that LISP was fine for special purposes, but it wasn't among the candidates for replacing BASIC.

For those offended, my apologies, but I remain unrepentant. LISP may indeed be a great language for professional programmers, as it certainly is for those working in artificial intelligence. Furthermore, if you're someplace where you can learn LISP easily—say MIT or Stanford—then by all means grasp the opportunity.

Most of us, though, don't have that opportunity. Even if you have access to MIT's LISP-teaching programs, it's going to take time—lots of time—to learn. The ideal way to learn LISP is to use it; it does have the great feature of being an interactive language (which is BASIC's great advantage). But the LISPs available for microcomputers are very limited (Minsky himself wasn't able to do much with the one I have), and it isn't likely that they'll get better—not with our present hardware. Come the revolution, when 32-bit machines with 256 K bytes of active memory and 50 megabytes of disk storage can be bought for $2000, LISP may then be the best thing available, but not now, unless you have very specialized needs.

In any case, this remains the *User's Column*, directed in large part toward nonprofessionals who are trying to make their small systems do useful things, and for those readers I don't recommend LISP. It can be fun to play with, and I'm glad Microsoft published it, but I doubt that microcomputer users will ever do more than play with LISP.

But the problem of languages has yet to be solved.

In theory, BASIC is an inadequate language. Listen to the hackers: they'll tell you that BASIC programs "are a maze of GOTOs" or that "you can't do structured programming in BASIC."

That isn't true. A good modern BASIC—say Microsoft's BASIC-80, or Software Systems' CBASIC—has *do while*, *if-then-else*, *case*, and nearly all the features Pascal has, plus string features that are a *lot* better than any Pascal I've seen, and decent I/O, which Pascal doesn't have at all. True, there are problems in BASIC that are easiest to solve with judicious use of GOTO statements, but it's certainly possible to write good BASIC programs without a single GOTO, and even easier to tame the GOTO so it never refers to anything outside a local modular block.

You *can* write top-down structured programs in BASIC. Best of all, you can write interactively, testing each step of the way, then test the program logic until it's working, after which you turn it over to Microsoft's BASCOM compiler, and wham. And for maybe 80% of the jobs you want a microcomputer to do, that's the best approach. It's almost certainly the fastest.

So what's wrong with it?

Plenty. First, BASIC still has a fatal flaw—no truly local variables. Passing parameters to a subroutine is hard, and controlling side effects (making sure you don't do something you didn't intend) isn't easy. You can reserve I,J,K,L as indices and set up "declarations" in remarks up at the top of the program, and with the new cross-reference programs you can *usually* find the side effects. Having done that, you still have trouble passing parameters, and if the program gets big, so that you'd like to compile it in chunks, you're out of luck. BASCOM doesn't permit decent chaining of programs, nor does it allow true compilation in parts.

In fact, the Microsoft BASCOM is not really the same language as its BASIC-80. In addition to the chaining problems, you can't use computed array sizes or common statements.

"But," protests the BASIC enthusiast, "that can be

fixed. In fact, I bet you somebody at Microsoft is working on it right now."

True. And maybe, one day, they'll really fix the BASIC/BASCOM system. I hope so, because I find BASIC programs fairly easy to read and write. I expect you'd be able to buy a lot of programs in compiled BASIC right now if it weren't for Microsoft's disastrous policy of demanding royalties for every program compiled with BASCOM. Competition will take care of that; meanwhile, I use BASIC-80 with BASCOM for routine jobs, and a lot of my most useful programs are written in CBASIC.

There's so much investment in BASIC software—take Joan Hughes, author of the PL/I book that so impressed Mac Lean. Joan runs Execudata, one of the first of the small-systems houses that have sprung up everywhere. She sells turnkey systems for small businesses. The hardware she favors is the Vector Graphic S-100 bus Z80 with its "mindless terminal," which is really memory-mapped video. The software comes from all over. The editor is Vector's, and I'm tempted to buy a Vector machine just to get it; it's a lot like Electric Pencil but with most of the bugs out. Execudata writes other software or buys it commercially. Like all first-rate systems houses, they support everything they sell, and their customers seem downright enthusiastic.

The interesting angle is that the author of the best book we've seen on PL/I sells software largely written in CBASIC.

Why? Execudata has been around several years. When Joan first started the company, there wasn't a PL/I for microcomputers and CBASIC was the only BASIC that allowed long variable names (remember the horrible days when variables were "A$" and "B1" and you hadn't the foggiest what they referred to?) and structured concepts. She may change over, now that we have PL/I from Digital Research.

Digital's PL/I documentation is, if nothing to brag about, at least readable. PL/I-80 is a fairly healthy subset of the ANSI General Purpose (Subset G) PL/I, and it runs.

"Yeah, and so what," asked a friend. "Who'd use it?" "PL/I is unwieldy and inefficient," says a recent article in a magazine. On the other hand, Mac Lean has spent the last couple of weeks learning it and he loves it.

Meanwhile, I have—at last!—gotten not one but two Pascals that run, and I've been wading through Peter Grogono's *Programming in Pascal* (see references) and reading everything I can find on Pascal.

I recommend Grogono's book—for that matter, I recommend the Pascal/MT+ implementation of Pascal. It works, and unless Mike and Nancy Lehman have become liars, which I doubt, Pascal/MT+ is a full implementation of the Standard Pascal, along with a few much-needed extensions.

Their manual is improved, too. (They sent me an early copy, and apparently my anguished screams were too much for them.) They've added a number of sections, and, according to Nancy, "That's all because of you...."

I also have the Sorcim Pascal/M up and running. Pascal/M compiles to an intermediate code. It's slower than Pascal/MT+, but it's also more compact and more portable. It, too, has extensions to standard Pascal, and I recommend it.

What I don't recommend is Pascal itself. Not yet, anyway. When I first looked at Pascal several years ago, I thought it was the nicest thing I'd ever seen. But the more I look at it, the more misgivings I have. I certainly could be wrong, and I'll know more by next column time. And by then, Mac Lean will have done things with PL/I so we can compare notes.

Meanwhile, don't throw away your CBASIC; *they've* made some improvements, also (Compiler Systems' President Gordon Eubanks says) as a result of my reviews. The needless limit on the line printer width has been fixed, and the COMMON and CHAIN features improved. I think the latest version has some other improvements, too, but I've mislaid the new manual. Although I've said it before, it's worth mentioning again: the CBASIC documentation is excellent.

Gordon also tells me that by the time this is printed there'll be a new version of CBASIC that allows nested IF

statements (fixing one of Joan Hughes's pet complaints), and before the end of the year they'll have parameter passing and local variables, and they're working on speeding it up.

The language situation isn't our only dilemma. Let's face it, our microcomputers are becoming obsolete. In one sense that's silly: we have more computing power than the government did ten years ago. The machines work reliably. And there's no point in replacing our machines just when we're finally getting good software.

But that's the problem: we're beginning to see hardware limitations on the new software.

Take Spellguard for instance. It's a good program. I use it often. But it can't touch *real* spelling programs like those that run on the big MIT computers. MIT's Spell (written, incidentally, in LISP) not only finds misspelled words—it shows you the word along with context, shows a menu of words it thinks might have been meant, offers you a chance to enter the new word in your permanent dictionary (if that's refused, offers the opportunity to put it in a dictionary kept just for this job), and, finally, lets you input the proper spelling, which it inserts into the text. It does all this at blinding speeds, searching dictionaries of 35,000 to 50,000 words. No 8-bit machine running floppy disks can touch that job. [**Editor's note:** *Don't be so sure, Jerry. See the review of spelling programs in this issue.* ...PL]

There are other limits. One controversy over text editors hinges on a simple forced choice: do you limit the amount of text you can work on to what can be held in memory (as Pencil does), or do you keep part of it on the disk (as WordMaster, WordStar, and Magic Wand do)? If you keep part on disk, you can't conveniently change disks: that can be a serious limitation if you want to bring in a chunk of an old file or make a quick copy of something you can't afford to lose. If you keep all the text in memory, not only are you confined to 10,000 words or less (with my system, anyway), but you encounter real problems if you want windows and multiple buffers or if you'd like the machine to do some computing on values in the text.

Obviously, I prefer Pencil's limits. I have Wand and WordStar, but I don't use them—Star because it natters at me (when was the last time you wanted to know line and column number every time you typed a letter?) and Wand because of the disk operations. (Word Master, on the other hand, is the best programming editor I know of.) None of the available editors, not one, can do what I'd really like them to—for instance, let me put equations in my text, solve them, and have the answers available as I write. (What I usually do is leave the text on the screen and turn to my programmable TI-59, and that's silly.)

Yet what I want isn't impossible—for big machines. MIT's MACSYMA can solve intricate equations, and its EMACS editor is inherently more powerful than anything we can implement on a microcomputer. I haven't seen them combined, but it wouldn't be that different if the memory was available.

New Machines

For a few years we'll exploit what we have, but it isn't going to be long before Zeke becomes the world's smartest terminal (two memory-mapped screens, a 20 K-byte PROM monitor to control printing and I/O, etc), while a new machine does the work. Only—what will the new machine be?

Two years ago everyone would have said "the Z8000." Now we just don't know.

Next, when we get the new machine, what will we use for an operating system?

Again, two years ago everyone would have said "UNIX, of course." And if we had UNIX, which is a fairly complex tree-structured operating system developed by Bell Laboratories, we'd also have answered the language question, since UNIX contains a C programming-language compiler and is written in C.

Now, again, we just don't know.

What went wrong?

First, there weren't any reliable Z8000 chips. Now that problem's fixed—but there aren't many available Z8000 computers, are there?

There is one. Onyx has a working Z8000, with UNIX and C, hard disks, and 128 K bytes of memory. They're being shipped as fast as they're being made, and according to their customers they work and work well. There's a good chance I'll have an Onyx running here in a few weeks and I'll be able to tell you a lot more.

Meanwhile, there's a serious rival to the Z8000 and UNIX: the dual 8085/8088, working on an S-100 bus and running Digital Research's operating system that looks a lot like CP/M and allows you to bring over most of your CP/M files. The 8085/88 won't run Z80 code, but then neither will the Z8000. It does execute 8080 code.

Bill Godbout sells an 8085/88 system, and Tony Pietsch, the engineer who built my system, recommends it, along with hard disks and dual-sided double-density floppies to back up the hard-disk system. Tony is building a Godbout S-100 system and there's a chance I'll get one for review. The Lehmans use a Godbout S-100 8085/88 system for their Pascal/MT+ and they're very happy with it, while Sorcim's president Richard Frank says they've got half a dozen running constantly and that you can pitch the Godbout box out a second-story window without hurting it.

In other words, Z8000 with UNIX has respectable supporters, and so does 8085/88. Prices aren't quite comparable: the Onyx system costs more, largely because of the UNIX software, which isn't cheap. Performance isn't comparable either: the Z8000 Onyx is much faster than the 8085/88 machines.

Fortunately, we don't have to make any decisions just yet. It may be that the best policy is to skip all the 16-bit machines entirely and wait for a new generation of 32-bit monsters. Certainly if you're contemplating buying your first system, go ahead and get one: I still recommend the S-100 bus and Z80. You can upgrade to dual-density floppies and hard disks without *too* much risk, but if you want a system to *use* rather than play with, be conservative and leave systems development to the Tony Pietsches of this world. (Notice that if I get new systems in here, they'll be in addition to Zeke; him I don't touch.)

And stand by. Exciting things are happening in microland.

* * *

I like computer games. I'm fond of ZORK and the various Automated Simulation games, and now they've got a new one called Star Warrior that's guaranteed to use up more time than you thought you'd give it, especially if you're a fan of Heinlein's *Starship Trooper*. The game has a lot of the same appeal as that book. It's played in real time, and you jump about in space armor trying to wreak havoc on the dastardly enemy while avoiding death. And it's balanced: you usually get out alive, but just barely.

I'm a Star Warrior addict. The boys like ASI's Hellfire Warrior, which is an extension of their famous Temple of Aphsai. They also like ASI's Rescue at Rigel, which is Dungeons and Dragons in spacesuits. I expect I'd like them too, if I had time to play them.

However, I'm not enough of a masochist to play Scott Adams's Adventure International games. There are those who like Scott's games a lot. Certainly, they aren't easy. Unfortunately, for me at least, they're just too tough. For example, he's got one called Balrog Sampler (it also has the title Maces and Magic #1 on it), a vastly complex game requiring two 5¼-inch disk drives. Like Automated Simulation's Temple of Aphsai, when you enter the game it generates a random character for you, after which you buy weapons, armor, and the like, before setting out.

Unlike ASI's games, Adams's game gives you no choice. You *have* to accept the characters randomly generated for you and a scruffy lot they are. But don't worry—if you're lucky enough to get one who's strong as Superman and rich as Croesus, he's still going to get killed about the time you get interested in the game. *Everybody* gets killed in Adams's dungeon, and often unfairly. In Zork and Adventure there's some logic to the puzzles, but in Balrog various things pop up and kill you before you can run away. There's one room in his dungeon where you find an attendant and a wheel of fortune. If you don't play, the bouncer comes and kills you. If you do play, you lose.

There are other "puzzles" in Balrog, but all of them are inexplicable—at least to me—and I always get killed. So do the boys, and one of them is an experienced (and fanatical) Dungeons and Dragons player, so it isn't just my amateur status.

It wouldn't be so bad if you could just play, get killed, and start over, but since the game is in BASIC (with encrypted messages and all kinds of kludges to keep you from analyzing it) it takes forever to initialize, load, create a character, and go through the ritual of choosing weapons and armor, just so you can get killed two minutes later. The dungeon keeps score and the program alters itself after every expedition so that when next you play, the dungeon proudly displays the number of players who have entered and the number killed. Ours is up to about 25 for 25 now, and nobody in the neighborhood wants to see the game again.

Meanwhile, Workman & Associates continues to sell the original Crowther and Woods Adventure for CP/M systems (see my review, December 1980 BYTE, page 230), only now they've added a twist.

"Only with our Build-an-Adventure Kit can you truly build an Adventure," the Workman copy reads, and I guess that's nearly true. The "kit" consists of an Adven-

ture data base and instructions for altering it or, if you like, starting over with everything new.

The data base contains a travel table (description of each room, where you can go from there, and what commands get you where); the vocabulary (five-letter or shorter words that the program will recognize); initial conditions for various rooms (light, water, oil, etc); messages, objects, including their initial location and whether or not they can be moved; treasures, which are a kind of object; and various other stuff.

It's amazing what you can do with it. For example, you can specify that to get from a room, say BEDQUILT, to another, say SWISS CHEESE, you will be successful 25% of the time if you go East, but always if you go Up, or that the travel will fail if you are not carrying the skull, or that it will fail if you *are* carrying the skull. You can automatically transfer from one room to an identical room (identical to the player), except all commands work backward in the second room, or there are extra objects and doors.

In the actual Adventure game this is done a lot: for example, there's a room with a live dragon standing on a carpet and another with a dead dragon lying near a carpet. Most players think it's the same room, since it's the same location, but in fact it's not. In the first room the carpet isn't real: it's part of the room description. If you tried to take it, the program would say "I see no carpet here" if the program wasn't instructed to issue messages about the dragon only. In the second room, the carpet is *real*, and so is the dead dragon, except that you can't "take dragon" successfully because he's an immovable object. Once you've killed the dragon, however, you can't get back to the room in which he's still alive.

Workman's implementation of Adventure includes a "wizard" routine that lets you move freely through the dungeon. He doesn't tell you how to use the routine unless you buy the "Build-an-Adventure Kit."

To build an adventure, you use a standard editor, such as Wordmaster, to alter the game data base (or create a new one), after which you invoke a BUILDER program. The program takes your logically ordered data base and makes a work file out of it, while simultaneously patching other chunks into the ADVENT.COM file that controls the game.

You can construct very complex dungeons and when you're done you have a game that recognizes vocabulary. You don't have to specify the permissible moves the way you must in BASIC adventure games.

I asked Workman how many of the people who buy the game intend to market adventures and he chuckled. He isn't sure, but he suspects that most copies are sold to people who bought Adventure and then couldn't solve the puzzles and mazes. "We give them the original Adventure data base with the Adventure Kit," Workman said. "That contains the Adventure map. We also tell them how to move through the dungeon by magic. I know that's why one guy bought the kit. He kept writing letters begging for the map. When we put out the kit, he bought the first copy."

Coming Up: In the next column I will continue to discuss languages, and I've also collected a raft of data bases I want to talk about. And I just finished letting Zeke do my taxes.... ■

References

Grogono, Peter. *Programming in Pascal* (revised edition). Reading MA: Addison-Wesley, 1980. $13.95.

Hughes, Joan K. *PL/I—Structured Programming*, second edition. New York: John Wiley & Sons, 1979.

Additional Information

Innovative Software Applications POB 2797, Menlo Park CA 94025		
Spellguard	*CP/M*	*$295*
Organic Software, 1492 Windsor Way, Livermore CA 94550		
Milestone	*CP/M, Apple Pascal, UCSD Pascal, CP/M-86*	*$295*
Datebook	*CP/M, Apple Pascal, UCSD Pascal, CP/M-86*	*$295*
Workman & Assoc, 112 Marion Ave, Pasadena CA 91106		
Adventure	*CP/M*	*$29.95*
Build-an-Adventure	*CP/M*	*$97.50*
CP/M Users Group, 1651 Third Ave, New York NY 10028		
Speed	*Volume 38*	
Each volume	*CP/M 8-inch disk*	*$8*
	North Star 5¼-inch disk (depending on the number of disks needed)	*$12*
Catalog		*$6*
Adventure International, POB 3435, Longwood FL 32750		
Balrog Sampler	*CP/M*	*$29.95*
Automated Simulations Inc, Dept SW7 Box 4247 1988 Leghorn St, Mountain View CA 94040		
Star Warrior	*Apple II, TRS-80, Atari*	*$ 39.95*
Digital Research, POB 579, Pacific Grove CA 93950		
PL/I-80, MAC, RMAC, LINK-80	*CP/M*	*$500*
MT Microsystems, 1562 Kings Cross Dr, Cardiff-by-the-Sea CA 92007		
Pascal/MT+	CP/M	$475
Sorcim, POB 32505, San Jose CA 95152		
Pascal/M	*CP/M*	*$225*