FAREWELL, WHITE SCARF? GOODBYE, GOGGLES?

Does history indeed repeat itself? We learned to fly back in the good old days, in an open-cockpit biplane, when the flight line of a misty morning was a scene out of a World War I movie in which the hero-pilots all wore (what else?) boots, white scarves, and goggles. At 18, we couldn't resist boots, a white scarf or goggles (girls could not resist them either).

Then some darn engineer invented the closed cockpit and shut out the wind (and the rain), stuffed the cockpit with gauges, crammed in an autopilot and other high tech gear--obsoleting the pilot's universal repair kit (pliers, a wrench, a siphon, tire tape, and lots of wire). Bureaucrats by the bushel began to direct by radio where you could go, what you could do, and where you could do it. They issued fat rule books and timetables. No longer could you fly, high and free, where you wanted. Ten years later, we discovered we were a glorified truck driver, sans goggles, sans boots, sans white scarf; that the fun and the glory were gone, and said to hell with it--for a while, until a genius created, from tubes and cloth, the ultra-light aircraft, and again gave us a critter we could saddle and ride wherever we wanted in the vasty canyons of the air.

AMIGA REVIEW ISSUE

See page 217

We flew (and fly) for two distinct reasons. First, for the pleasure of flying, for the feeling of mastery of the element and of the machine. Second, to get to Peoria as quickly as we can (when we board an airliner and do not necessarily expect to enjoy the trip).

Until the present time, the personal computer has served in both roles; it has given us the pleasure of learning to master it in its own element, and it has 'gotten us to Peoria' when we used it for business. All of us have enjoyed the white scarf and goggle days of personal computing because these conflicting views--business and pleasure--were reconciled in relatively simple machines which, with reasonable study, almost any computer pilot could master and fly.

The good old days are about over in personal computing. Say farewell to your white scarf; bid goodbye to your goggles. History indeed does repeat itself. As you'll see later in this issue, the new generation of personal computers has been designed to 'get you to Peoria...' but personal mastery is just about gone.

If you're the type to enjoy doing snap rolls or inverted Immelmans by pushing the right button, you'll love them. If, like ye ancient editor, you'd sometimes prefer to execute your own snap rolls or to invert an Immelman your own way, you may well find that you can't.

The new computers are as complicated as the airliners which replaced the old biplanes. Professional programmers and circuit designers can understand them; few others will. Most users will now be limited to professionally written programs, to high-level languages, or to a simplified user interface.

We may use the new computers as we'd use an airliner; we'll get to Peoria, and swiftly. We may, in a lugubrious metaphor, snap roll a Boeing 747 whenever we please. But the software is the master of that maneuver; we aren't--and some of the fun in computing is gone.

Do not take what we say as dislike for the new computers. You may love the old while liking the new. Once upon a time we laid hands on an mint Stutz Bearcat, owned--literally--by a little old lady in tennis shoes, who had stored it on blocks when her husband died. We never have driven a car which gave us a greater sense of freedom, speed, and delight in the road--even though it smoked, roared, and steered like a truck. Yet who would choose a Bearcat today if offered a Porsche? (Well, on Tuesdays only, we would.)

Which leads us to speculate that the ultra-light aircraft may have a parallel in personal computers one day--a lean, mean machine, fast as scat, without windows, mice, multi-tasking, or nearly half a megabyte of code for an operating system. We also suspect the C-128 and the Apple II (and SuperPET) will live a lot longer than expected--because you can control and fly them all by yourself. If you own one, you can still wear your boots, your white scarf, your goggles...and as the total master of your machine, fly it however and wherever you please.

ONCE OVER LIGHTLY
Miscellany

Lous Mittelman, Jr. tells us that those who have rare and unusual early computer items may contact Dr. Utah C. Merzbach, Curator of the Division of Mathematics, National Museum of American History, Smithsonian Institution, Washington D.C. 20560. Or call 202 357-2392. (See Once Over, last issue, re contribution of early computer stuff). The attic is sorta full; don't send anything unless you call or write.

NOW MEANS NEXT NOVEMBER??? In the August/September issue, we offered SuperPET schematics to those who ordered NOW, and said we would not stock the item after orders were filled. Yet orders keep trickling in at the rate of about one per week. As we write this on November 30, 1985, we have another one. Folks, you do NOT stock 55-page schematics unless you own a warehouse and are rich enough to tie up your money in documents no one may ever order. Instead, we waited three weeks for orders, added them up, printed the lot, and shipped them. We even ordered ten extra sets for those who procrastinated. The extra sets are gone. So please--when we offer an ORDER NOW item, order it now, or don't order...

REPAIR AT COMMODORE : CHANGES: In II, Issue 6, p. 155, we noted a new arrangement for getting in touch with Commodore for repairs. Things have changed. There's a new number to call for information. The old toll-free service number of 1-800 247-9000 has been discontinued. You must call 215 436 4200. To get specific direction as to where to send equipment for repairs, call 215 431-9235. If nobody answers, let it ring. When an obviously tired but polite service type answers, tell him where you live. He'll tell you where to send the gear.

For those in the central U.S.: Try the National Repair Center, 3354 Winbrook Drive, Memphis, Tennessee 38116 [1-800 248-2983]. They assume you own a 1541 drive if you mention drives, so be sure they know you have an 8x50 or a 4040. Minimum charge is $50; rates are $50 per hour to a maximum of three hours, plus parts. We haven't tried the service but are about to. They accept VISA and Mastercard, cashier's and personal checks (after a delay on the latter).

Ship by Federal Express or UPS. Enclose in the shipping container a precise written description of the problem, your name and address, and a statement authorizing use of a specific card number with expiration date. They're supposed to call you about charges if repairs will be extensive.

OOPS, WOOPS AND DANG DEPT. Before we started this rag, we couldn't understand why each issue of any computer magazine carried a big errata section for the

previous issue. Hmmmph. It's easy to catch errors if both editor and author proofread a draft, right? Well, er..., wrong. As anybody who has searched for a program error knows, you can read a mistake ten times and never see it. Which is why we prefer programs on disk; we copy 'em into text only after we've run 'em successfully. We resolutely refuse to hand-copy programs because mistakes seem almost inevitable...

Even when we get programs on disk, errors creep in. APL is a problem. We print every issue in one printer pass, changing printwheels between APL and ASCII as our formatting program tells us to. We limit ASCII printing to 80 characters per line, but that won't work for APL if authors happen to slip us a passel of overstruck characters. Every such character parses out to a character, a backspace, and a second character--which increases the character count by 2. If we have a printed line of 80 characters, four of which are overstruck, the character count is 88, not 80. We lose the last eight characters when we print. Reg Beck tries hard to avoid this; we try to check carefully, but in Volume II, p. 83, we mutilated the APL compiler just this way. The end of line [9], near the top of the page, should end with: LOC (not with L). Sigh. Please correct your copy. Thank Al Coombs for reporting the error.

NEED A BACKUP MACHINE? Mid-Kansas Computers, PO Box 506, Newton, Kansas 67114 has a number of SuperPETs available for sale at $200 each for the bare computer, without disk drives. Check on manual availability if you are interested. (316) 283 0208. Ask for N. M. Patton.

WORKAROUNDS IN SPMON1 SPMON1, Terry Peterson's powerful monitor, lets you COMPARE two sections of memory and DIVERT the output to printer. Assume, for an example, that you want to compare two variations on the same program and find the addresses of all differences. You load both programs; one at $1000, the second at $2000 in RAM. Assume further that you must compare the first $60 bytes of code in the two programs. The 'compare' command at left outputs the addresses of all differences in code to the screen. Because we don't want to hand-copy all those addresses, we have the option to DIVERT the printout to printer, using the second set of commands at left. We assume the printer here is 'ieee4'; we give the command to divert before we do the COMPARE. And it's right here that some folks run into problems.

```
c 1000.60 2000
```

```
o ieee4
c 1000.60 2000
```

When Terry wrote SPMON1, he didn't know what printer you'd use or how wide it is (80 columns or a lot more). So SPMON does no carriage returns; it simply outputs data to printer. IF your printer doesn't do an automatic CR at right margin, you print one line and then hammer the rest of the output in one place, at right margin. Almost every printer we've ever seen can be set to do an automatic CR when the printhead gets to right margin. Set your printer this way before you do an Output Divert on a long COMPARE. Last, if you find the printhead of your printer hasn't returned to left margin when COMPARE is finished, simply do an: o <RETURN> on screen; this again diverts output back to the screen, sends a CR to the printer, and closes the printer file.

In the above, we assume you won't use the screen dump in SPMON1, which you trigger with a simple 'p printer' (or serial or ieee4) because the output list may be quite long. If the list is short, stick with screen output and dump it.

Problem Two The simple assembler in SPMON uses the number sign '#' to mean both 'decimal' and 'immediate value.' In the instructions, we recommended that

you always enter numeric values in hex in SPMON to avoid confusion--such as the problem at left. The first line is an attempt to offset DECIMAL 12 bytes into the value in X register; it fails. Use either of the alternatives shown, which work well. It is simplest to use hex.

```
LEAS #12,X  [Attempt in decimal fails]
LEAS +#12,X [Uses decimal 12 as value]
LEAS $c,X   [Done in hex. All okay]
```

If you must convert from decimal to hex, remember that SPMON1 will convert for you. You merely enter the decimal value on screen, prefaced with #, hit RETURN, and are given the value in hex and decimal on the next screen line. You can cursor out of the assembler at any time, convert, cursor back up and continue assembly.

ANOTHER GOLDURN OCEAN! Last issue we asked if Canada was now bounded on the south by a new ocean, because Canadians keep sending us $25 for overseas dues. Now the Texans are doing it! We already knew that Texas was a separate nation, to which the rest of the U.S. is joined only by reluctant permission--but dagnab it, we didn't know the borders of Texas were now bright bounding sea. For those whose borders have not been inundated by sea-waters and fishes, ISPUG dues (both Canada and the U.S.) are $15 per year. Overseas dues are $25. U.S.

GOSSIP, NEWS, and AMIGA: We thank all those who responded to our note last issue asking if anybody was interested in a new publication about Amiga. It seems you want more news and plain gossip in the Gazette about microcomputers in general--and Amiga--even if you have no plans to buy a new machine soon. We'll try to oblige if you'll all play Deep Throat and send us all leaks, rumors, gossip, findings, speculations and Mother-Grundy-stuff which comes your way.

TWO ARE NOT ENOUGH Those who have received ISPUG disks know we ship them protected by three layers of corrugated paper, which a high-school lad slices up neatly for us from otherwise wasted boxes. Two of the corrugated slices run in one direction; on the third slice, the corrugations are set at 90 degrees to the first two to keep disks from being bent and ruined by some Hercules in the Postal Service.

Unfortunately, when folks send disks back, we usually find only two layers of cardboard (the two on which corrugations run in the same direction). Such packages are easily bent in handling, and not a few disks sent back have been crammed into a PO box bent double, despite two layers of cardboard. Please use the three layers we always send; in three years we've had nary a disk so packed damaged in the slightest.

PASSING PROCEDURE/FUNCTION NAMES in PASCAL

Marvin Cox of Oak Lawn, Illinois, has been evaluating the TCL Pascal compiler we briefly noted in II, p. 123, and has found a number of problems and anomolies. Whilst demonstrating that the compiler would allow you to pass procedure or function names as parameters, Marv discovered that our microPASCAL also supports procedure and function names as parms when the keywords "function" or "procedure" are omitted in the parms. As Marv notes, the Waterloo mPASCAL users guide states on p. 132 that "passing procedure or function names as parameters is not supported." Marv, as with the rest of

```
program heading (input, output);

function product(twoformal,threeformal:real):real;
  begin
  product := twoformal * threeformal;
  end;
```

```
function two : real;
  var localtwo : real;
  begin
    localtwo := 2.0;
    two := localtwo
  end;

function three : real;
  var localthree : real;
  begin
   localthree := 3.0;
   three := localthree
  end;

begin
  writeln(product(two,three):6:4)
end.
```

us, took this as gospel, only to discover a way around the limitation, demonstrated in the program at left.

The approach works also with procedures. If "function product", left, above, is replaced by a procedure, we begin:

```
procedure  times(twoformal, threeformal,
                answerformal:real);
   begin
   end;
```

and then add a "function answer", below:

```
function answer: real;
   begin
   answer := two*three
   end;
```

And, last we revise the main program:

```
begin
times(two,three,answer);
writeln(answer:6:4)
end.
```

It's pretty clear that function and procedure names are being passed as parms in headings in both examples. Marv notes that each program runs in both mPASCAL and the TCL compiler. Sorry to say we still can't get permission to distribute it.

[Ed. Note: We received an overwhelming response to our question, last issue, on interest in Amiga amongst ISPUGgers. Without exception, those who wrote asked for a full report.

Terry Peterson, John Toebes, Joe Bostic, and ye ed all have Amigas plus the Developers' manuals. No, they weren't loaned by Commodore-Amiga; we paid cash from our own pockets. As we learn more about the Amiga and as Commodore-Amiga issues new versions of software, our views may change. As this issue went to press, we received the first cut at a new Editor for Amiga from Joe Bostic, and are encouraged by his progress.]

HANDS ON AMIGA : A First and Tentative Report

As you read this, allow for the writer's bias on graphics. When color television first came to market, we refused to buy one, on the basis that garbage in black and white would be even more odious in living color--a view which time has confirmed. For the generations raised and educated via the tube, this is heresy. Words often are much more powerful and evocative than pictures. "Spring," for example, draws in the mind an infinity of springtimes-- the carpet of blossoms on the arctic tundra, the carpet of mud in Vermont, the Azelea bursts of pink and crimson in Virginia, the incredible green of English May. Now, draw us a picture of spring...

For the specific and the dramatic, pictures have no peer. Matthew Brady's photographs of Antietam evoke that grisly battlefield across a century of time as

words never could. Words and graphics communicate well when they are properly intermixed to do what each does best.

With this bias, we see too much emphasis on pretty colors and graphics; too little on use of words--plus clumsy word handling. Suppose you create seventeen text files in Notepad (a simple little text processor). You then see seventeen icons representing seventeen pieces of paper, together with seventeen filenames. It almost fills a screen... What happens when you have fifty files? Icons are the current fad. We'll see a reasonable admixture of graphics and words when the fad has cooled. In the end, we'll probably have one "paper" icon and seventeen or fifty filenames in English--far swifter and easier to use.

We'll say little more about graphics in this review. No "draw" programs are yet on the market; we've not had time to try the graphics features of ABasiC; the next version of "Workbench", the chief icon-based user program (now both limited in capability and buggy) is due soon, and we expect it to be significantly improved. We hope to report on these matters next issue.

Despite our caveat on graphics, Amiga is a splendid machine. Most of what we find wrong is piddling, nickle-and-dime detail, and almost all of that is in the software. Almost all can be (and probably will be) corrected. We now look at specific features of Amiga, one by one.

THE HARDWARE : SETTING UP

We must compliment Commodore on the design of the physical parts of Amiga. From the packing and shipping cases through the cabling and arrangement of parts, much care and forethought are evident. We had Amiga set up and running within 20 minutes of its arrival. The cabling connections are excellent, secure, and unmistakably marked--a great improvement over previous Commodore machines. The care with detail even runs to the mouse--which has a little cover you may slip off whenever the interior needs cleaning.

The system unit contains one disk drive, the motherboard, 256K of user RAM, plus 256K of read-only RAM for the operating system (as a substitute for ROM). We bought an additional 256K of RAM, which installs under a cover on the front of the system unit in about five minutes, giving us a total of 512K of user memory plus 256K for the operating system (OS).

Commodore's 1070 RGB monitor sits atop the system unit. The ON/OFF switches for both are no longer obscene, being up front where you can find them. Monitor controls are behind a small door and very handy. So far, the hardware in both units has performed flawlessly. The monitor has a large screen (13.25 inches diagonally as compared to 11.25 inches for SuperPET), and is most easy to read. Color and control are superb.

We installed an additional 3.5-inch external drive, a tiny thing (6 x 8 x 2.75 inches). For some mad reason, Commodore made the cable so short it is impossible to place the drive on the left side of Amiga, where our desk space requires it to be. If you should get one, insist on a longer cable! You cannot install more than one additional 3.5-inch drive without an external power supply. The single extra drive we have draws its power from the system unit.

The Teeny Drive Drive operation is fast, a backup of a full disk requiring a few seconds over one minute. Single file COPIES are, however, relatively slow-

er, though considerably faster than COPY on an 8050. In languages, programs LOAD in a flash. Our SPET and its drives are comparative snails.

We regret to report that our drives are much noiser than any 4040 or 8050 (except for the knock-knock on old drives when resetting). Both our drives wheeze, moan, sing and mutter whilst operating. Unless a drive holds a disk, it feels for one with a periodic, maddening thump, so we always stuff a scratch disk in any unused drive. Reports from others indicate our drives are noiser than most.

Disks pop into and eject from the drives easily. Amiga senses either action and always knows the names of the current disks. Once in a while, a disk will not seat properly on the drive spindle when inserted; you can pop it out and put it back in without harm (so far).

Commodore warns you emphatically not to remove a disk while the drive operating light is on; if you do, a disk probably will be ruined. We wondered what would happen if the Mickey Mouse P&L let power fail during a disk operation--and soon found out. We crashed a disk. Because Amiga does a lot more writes to disk than most systems, you absolutely must back up your work from time to time.

The Keyboard: We love it. It is indeed a standard Selectric layout, with a light crisp touch, low enough to let you rest your wrists on the desktop while you type. SPET's keyboard now feels rubbery and stiff. Unfortunately, Commodore goofed on several features: first, the two ALT keys (which give you the alternate characters among other things) are placed exactly where the edge of either hand may press them accidentally whilst you type. All of the ALT characters are visible--except one. That is ALT space, which has the code of $A0, and which is entirely invisible. We spent four frustrating hours finding a compiler error in a C file caused by a single ALT space... Software simply must be revised to print ALT space in reverse field or color!

We finally built an aluminum guard which won't let us touch the ALT keys unless we really mean to press them. Problem solved.

Second, we guess the designers of Amiga still think there's a carriage return spring in there somewhere. You cannot TAB right to left (with SHIFT TAB) to go back even one lousy tabstop to the left. You must either cursor back or TAB through the entire set of tabstops, as on 1898 Underwoods and SPET. Fortunately, this can be cured by software.

Third, the INSERT key is gone. DEL, shifted or unshifted, simply deletes. Can you cursor back and overstrike for a correction? Nah. Every editor in Amiga goes into INSERT mode when you try to overstrike. If you by error type "Jsmes", cursor back to "s" and hit "a", the result is "Jasmes." You must DEL the "s". We find it maddening, because wrong characters are much more frequent than missing characters. As we'll later see, the folks who wrote the editors must be ancient mainframe, MS DOS and CP/M types, well versed in the obsolete art of editing on teletype machines.

The Mouse Mice have never impressed us, our desk space being limited (no matter how big a desk we have, it is always filled with computers, printers, copy holders, disk containers, copy, books, notepads, pencils, letters, files, coffee cups, mailing labels, lists...). We have no cause to change our mind after using Amiga's mouse. The two-button gadget is reliable and easy to use--if we could find space for it.

Amiga lets you magnify mouse motion so that a small mouse move becomes a large move on the screen. At maximum setting, we still must reserve a patch of desktop 6 x 6 inches for the mouse (at minimum setting, you need 12.5 x 16 inches!). We never have it. The mouse either sits under or over our notepad, or runs into our coffee cup, or stumbles over a pencil... Oh, the mouse works very well, but we'd give a pretty to have a big trackball on the keyboard to move the mouse pointer and a couple of convenient keys to click instead of the mouse keys.

You can control the mouse pointer on screen from the keyboard, but it's hard. The right and left Amiga keys (to either side of the spacebar) are supposed to be substitutes for the mouse keys, but we find them hard to use. To change from plain to italic printing in Notepad, for example, you must press the right Amiga key and "i". Take a look at SPET's keyboard. Try to press a key just to the right of the spacebar and the the "i" key at the same time with your right hand. Contortionists and octopi may manage it.

We say again that a big trackball on or near the keyboard with two keys above it would beat any mouse ever invented, and we stand by it. Some folks with clean desks will like the mouse, but when someone offers us a trackball rig, so we can roll the mouse pointer with the palm of a hand, we'll snap it up.

DISKS AND THE OPERATING SYSTEM

A part of the OS is loaded at boot from a disk called "Kickstart." Amiga then asks graphically for the "Workbench" disk, which loads more of the files and (as received) loads the main user interface program, "Workbench." Magazine reports on the size of the Amiga OS say that it occupies 193K. Dead wrong! 256K of RAM are reserved for the OS. There is 193K more on disk. If you load all of the DOS commands (from COPY to MAKEDIR) from disk into user RAM, the OS occupies, physically, 256K of write-protected RAM plus 193K of direct DOS commands plus the directories, libraries, and files the DOS commands refer to while executing. We haven't sorted this out at all, so we most tentatively say it appears you need somewhere above 200K of files to handle all the DOS commands.

Because you can't use a RAMdisk that large on a 512K machine, every Amiga DOS command is (pretty swiftly) loaded from disk before it is executed. There are two serious disadvantages: it slows down execution, and external DOS takes up a lot of disk space. We stripped one disk down as far as we dared (tests will let us strip it more, we're sure) and 52% of the disk was filled, leaving us with a working disk capable of holding about 420K bytes of our own programs.

There are a several reasons to dislike extrinsic, disk-based DOS. If you use DOS (and you will use it, either in Workbench or from Amiga DOS itself), you must have at least one disk with all required DOS files in one drive or another. Now, suppose you want to copy files from a disk friend John sent you to a disk of your own. Neither your disk nor John's disk hold the DOS extrinsic files. How do you copy the files John sent you?

Well, you make a special BATCH file, which loads the required DOS commands and supporting files into a RAMdisk; you tell Amiga to look to the RAMdisk for all its DOS commands; you take your DOS disk out, stuff in John's disk and your copy-to disk, and tell the RAMdisk DOS commands to copy from one disk to the other. When you get through, you re-insert your DOS disk, tell Amiga to forget the special RAMdisk, and resume operations. The process is slow; it irks us.

You shouldn't conclude that disk-based DOS commands are all bad. As Joe Bostic wisely notes, "To put the disk-based DOS commands into ROM would be a mistake as I see it. It would limit Amiga DOS to the commands somebody thought of at the start and be inconsistent with Amiga's promised expandibility and adaptibility. Suppose that DOS commands are later put in ROM. If Amiga DOS would still look to the system disk for commands it could not find in ROM, we would get faster, simpler operation and yet retain the ability to add or revise routines." Joe has a very good point. We aren't at all sure we ever want DOS in ROM, which, after all, is only one form of memory. Why not stuff it in RAM, and preserve the option to improve DOS as time goes on?

It appears the best solution is to buy another 512K of memory if you can afford it and to stuff all of the DOS commands and system files into a RAMdisk when you boot if 1) you want superswift DOS operations and 2) you don't want to encumber your disks with DOS. Amiga DOS includes a startup sequence BATCH file you can amend easily to do this.

There is a price for every capability in a computer. Obviously, if you want a DOS you can revise, expand, and improve, you'd damn well better not depend totally on DOS in ROM. And for this, you pay the price of extra RAM or suffer the consequences of a disk-based DOS as outlined above. Nothing comes free.

RAMdisk Well, of course, you noted above that Amiga allows a RAMdisk--which is most efficient, easy to set up and use, and fast as dazzled scat. Would you believe you can file a five-page document from an editor to RAMdisk in the literal blink of an eye? You'd better. No fixed amount of user memory is tied up for any RAMdisk, which never is larger than the files which you assign to it. If you delete a file, RAMdisk is smaller by that amount. Making a RAMdisk is easy. You issue the commands at left to create a RAMdisk directory called "work" and to copy a file to it. You may copy out of RAMdisk to either drive just as easily. We love it (except when we forget to copy our work to a floppy before we shut down, woe, alas and alack).

```
makedir ram:work
copy name to ram:work
```

USING AMIGA DOS

Commodore makes it easy to switch from the icon route to a command-oriented DOS at any time. And, on the whole, we love the DOS. It is powerful, easy to learn, easy to use. And in English! Those who want an excellent summary of how to use it should get a copy of the January 1986 COMPUTE! ("Introduction to AmigaDOS"). SPETters who buy an Amiga will feel very much at home, though they have much to learn. The Amiga DOS User's Manual is pretty well done--and indexed.

We've always been cool to the idea of multi-tasking (concurrency, or doing many things at one time), but Amiga is convincing us we were wrong. Here's an example of how handy it is. Last night, we were debugging a C program which employed pointers (woe, woe!) and on the Amiga DOS screen had a list of (sob!) some 12 errors from the compiler. ED, the screen editor, was running in another window, and held the code we'd just compiled. In DOS, we looked at the first error report by line number, clicked over to ED, found and corrected the error, clicked back to DOS, checked the next error, clicked back to ED and fixed it...and then ran into an error we couldn't figure out. But we remembered another program on disk in which we'd had the same problem. So we asked DOS to make another window (a new CLI, or command-line interface), and printed the old file to screen. Aha! The solution!

We turned off the new CLI, clicked back to ED, fixed the error, and then clicked back to DOS for the next error. It was fast, easy, simple, and oh-so-convenient!

On any other machine, we'd have been forced to make hard copy of the list of errors, to load ED, and to sequentially load and look at all those files. Then we'd have had to reload the faulty program and fix it. Not on Amiga! At the tips of your fingers are as many processes and files as RAM can hold; you can flip between them with a click of the mouse button or from the keyboard.

How many concurrent programs can you run? We haven't tried for a limit, and frankly don't care. The ability to copy a disk file while you EDit while you send a file to your printer while you flip back and forth between DOS and your source files has sold us on concurrency, which is absurdly easy to handle. We have noticed that our typed output to the screen in ED is a bit jerky when we have several processes running together, but we can live with that.

But--alas--concurrency has its price, too. That price is a big OS, able to manage multiple programs and multiple windows, and so complex that few computer pilots will ever be able to write programs outside of high-level languages which will run on Amiga. Part of the price is paid in speed. Our lead article this issue concerns itself with the result.

Would You Believe 30-Character Filenames? You got 'em. MS DOS and CP/M may limit you to eight characters plus a three-character extension, but at long last we have an OS which lets you identify your files in English. We swoon!

You may retrieve directories fully sorted (with DIR) or merely listed (with LIST or LIST QUICK). If you want 'em sent to disk or printer, it's easy to redirect output to printer (first example at left) or to a disk file (second example), where ">" means "redirect output to". Unfortunately, the loading of directory data and the sort for DIR listings is slow. DIR requires ten seconds to load and to sort a directory of 29 files. Our very own BEDIT produces a sorted list of 29 files from an 8050 in four seconds. A DI call from BEDIT (a list without a sort) requires three seconds from an 8050. On Amiga, a LIST QUICK requires 11.5. We suspect a complex directory arrangement is the problem, because the time does not change a whit if DIR and SORT are executed from RAMdisk rather than floppy (and the directory printed out is a floppy directory).

```
dir > prt:
dir > listing
```

You may get a selective directory listing by using a substring to define what you want (dir s .sam, for example, shows only the files containing ".sam"), or you may select them with a pattern. The SORT which alphabetizes DIR files is separately available to sort any text file.

We'll not say much about BATCH files, which you may EXECUTE to have Amiga automatically execute complex tasks. They're built into Amiga DOS and work well. We'd like a few additional features, particularly the ability to choose options while a file is executing. Unlike many other features of DOS, the EXECUTE facility is incompletely documented; several of its features are inscrutable.

Some Chrome Trim, Foxtails, Bells and Whistles While Commodore-Amiga did not think of everything for DOS, they did stuff in some pleasers. When you boot, Amiga asks for the time and date. If you care to, you can say: "08:09 today", or even "08:09 saturday", and if you've run the machine within the week, it will

consult its perpetual calendar and convert the "today" or "saturday" to the correct date and year. You can ask Amiga to LIST the files you have put to disk since a certain day or date, or even say since 'yesterday'!

Anything you conceivably might want to know about the filing system you can ask for--blocks occupied, size of files in bytes, disk percentage filled, which disk of what name is in what drive, and so on. Directory listings are not encumbered with this trivia, but it's available on call--a far cry from older CBM machines and from other operating systems we're too kind to mention. On Amiga, you can even ask to read your files in hex and send the output to disk, screen, or printer.

Nested Directories As you'd expect on a system designed to use hard disks, you can create as many different directories as you wish, to hold specific types or classes of files. DOS imposes no limit on the number of levels of nesting; you're well-advised, however, not to nest them very deeply or you'll have a devil of a time getting back a file. Why? Well, you must specify every directory in the list. Suppose you have a main directory "one", which holds a second direcory, "two", which in turn holds another directory, "three", which in turn holds the file you want--"file". To get your hands on "file", assuming it is on drive 1 (df1:), you must say: load df1:one/two/three/file.

For this reason, we never nest a directory inside another. Instead, we maintain as many directories as we need, as in: df1:programs, df1:text, etc. This way it is easy to create directories and yet to find the files you want. Suppose, for example, you want to work up a number of math programs. You MAKEDIR a directory named "math", and file all your programs to drive 1 easily with the identification of: df1:math/filename. With multiple directories, you may put like files neatly in one place. You may COPY all the files (and just the files) in one directory to another directory or to another disk with one command.

Yes, we've grown fond of DOS, and we expect it will be improved.

THE DOCUMENTATION

It is rare indeed this early in the life of a new computer to see a passionate effort by the manufacturer to document the machine. We think Commodore-Amiga deserves an award for valor. Most of the manuals are in plain English and done with care (typeset, no less!), far superior to anything we've previously seen from Commodore.

Yes, there are errors and omissions and ambiguities. After all, the nine manuals we received (most of them for developers only) outweigh the Manhattan telephone directory. Someone said of the Apple documentation on Macintosh, "...it is comprised of 25 chapters, any one of which you'll comprehend only if you understand the other 24." The statement holds for the Commodore developer's manuals. We expect we'll need six months to a year to wrap our mind around the package.

User Manuals The manual "Introduction to Amiga" is well illustrated if a bit confused here and there. That for ABasiC is cursed with a curious lack of examples just when you most need them, in favor of formal syntax generalities such as that at the left, which we abominate, never having been able to use them to form actual commands without hours of trial and error. If

```
box (x1,y1 [;x2,y2]) [,<fill>]
```

made king for a day, we'd exile to outer Slobbovia anyone who issued a manual without LOTS of clear and specific examples to illustrate the general forms of commands. (Most of Waterloo's manual writers, you may note, would now be senior Slobbovians.)

LANGUAGES : USED AND AVAILABLE

It is clear from the manuals that much of the OS is written in C, but likewise clear that some has been done in 68000 assembler. Pains have been taken to use and identify specific 68000 registers for specific system calls for the assembly language programmer.

We doubt Amiga could have been brought to market this year or next if the OS had been done completely in 68K assembler. But there is an apparent price for C; speed of operation. John Toebes disagrees with us on this, but we persist in the view that an OS done in C is slower than it ought to be. As time goes on, we expect that Amiga's OS code will be tightened up either by rewrite or by conversion to assembly. No, we don't complain about speed; we just want more.

Available for the programmer at this time are the following languages:

Lattice C Compiler, linker, and library. We've used it so far without any problems, though John Toebes says he has run into a couple of bugs. Lattice has a reputation for a robust, highly transportable version of C which adheres to the Kernighan-Ritchie book, and adds some features which may ultimately be found in an ANSI standard, type VOID amongst them. We've been following K&R chapter by chapter by chapter on Amiga and have no fault to find. Where Lattice differs from K&R, the differences are explicitly stated.

The Linker (provided by Commodore-Amiga), not Lattice, is another matter. Joe Bostic says it truncates long label names to eight characters. We know it is too too slow. John Toebes says it needs a complete rewrite.

68K Assembly This is a macro assembler which unhappily doesn't provide for writing structured assembly language. Why people continue to create assemblers without providing for structure we do not understand. Everybody complains about how long it takes to write assembly...but nobody does anything about it--except Joe Bostic, who has put together a number of macros which enable you to use IF...ENDIF and its companion IF...ELSE...ENDIF, GUESS...ADMIT...ENDGUESS and good old LOOP...UNTIL.

Joe reports no problems with the assembler, but is as unhappy with the Linker as those who write C. It works, but linking is much too slow. We don't know why we must deal with labels of eight characters or less when everywhere else in Amiga you may use 30-character names for variables and files. It's a holdover, we're sure, from versions done under more ancient and limiting operating systems.

ABasiC Here you sit, with 490K of user memory, and you're gonna write this great, long program in a great, new Basic which gives you control of sound, graphics, color, windows... Well, you're gonna write that program GOSUB and GOTO, friends. Woops, we forgot. They did stick in a WHILE...WEND. How kind.

Great, new, modern 1985 computer, the most powerful of its kind. Old-fashioned, sphaghetti code Basic, model 1891. One vomits. Why, in the name of Ada, are

computer makers so desperately afraid of the option of structure in Basic? Is anyone forced to use it?

ABasiC has a nice PRINT USING for handling numbers, all sorts of graphics commands, speech control, an arsenal of math functions, distinguishes upper and lower case, and is well-equipped with other functions, many of them new. It can handle both single and double precision floating point, but the accuracy is nothing to brag about (see the benchmarks).

It also has the absolute worst line editor we've seen since we used ED in CP/M. You can't overstrike or insert. You edit a line below a copy of the line itself, using commands. At left we show the command to insert (i) followed by the character to be inserted (again i). After a few more manipulations, you finally get "print." If all other features of this language were perfect, we'd still condemn it to the Gulag for the line editor. But the rest isn't perfect. Yes, you can write programs in ED, send them to disk, and load them into ABasiC, but for debugging or revising code you must have a decent, built-in editor. Why you cannot overstrike, amend, and revise as you can in all previous CBM machines we simply do not know. We suspect the folks who wrote the language are either old mainframe types who have fallen into evil ways or were drilled daily from birth on how to correct errors on a 1922 teletype machine.

```
100 prnt "Hoooo!"
      ii <RETURN>
```

Sadly, as we were revising a program to profile our printer, we forgot to stuff END at the end of our program, called for "renumber 100", and flipped into the land of Oz. We tried it again, and again met the wizard. Turns out these guys MEAN it when they tell you to END every program. You must reboot if you forget, which is carrying syntactical enforcement a hairy bit too far.

Other Languages REAL SOON NOW We were in touch with the folks at True Basic, Inc., who told us they expected to have their version of a structured Basic for Amiga in the first quarter of 1986, which probably means before the end of the year. We've heard rumors that Microsoft will produce an Amiga Basic, but, knowing Microsoft, it'll be the old GOSUBey GOTOey version. Hold not thy breath.

Borland has announced that a version of Turbo Pascal is being prepared for Amiga for issue next year. Manx says Aztec C will soon be ready. We would dearly love to see an interpreted C (three distinct C interpreters are out for the IBM PC). The shortest of C programs ("Hello, world") requires about 2 minutes and 20 seconds to compile and link; a complex program can waste hours of time while you again edit, compile and link. An interpreter could cut it down to minutes. The thought makes us drool.

BUGS, ALARUMS AND CONFUSIONS

Suppose you hook your printer to the parallel port on Amiga and use it in Workbench. Suppose it works perfectly for graphics and in draft and near-letter-quality modes.

Whilst you are still glowing, write a little program in ABasiC to profile your printer for workaday drafts--to set margins, perf skip, and so forth. Send the output to disk so you can copy this file (profile.print) to your printer (filename PRT:) any time. Then do so. Wooops! The printer just sits there. Nothing happens. Work at it for three hours. Try everything. Failure, confusion...

Oh, dear, you have just tripped over printer Esperanto, a universal language used by Amiga to handle all printers. If you employ the icon interface under Workbench to set page length, type of paper, and so forth, can you reasonably expect Workbench to contain the printer drivers for the 18,489 different printers on the market and their 1,504 different command sets? Not hardly. Amiga thus accepts only printer Esperanto (really an ANSI standard) whenever commands are sent to PRT:, and the OS looks on disk for a printer driver for the printer you have selected, interprets printer Esperanto to Printer Dialect 849, and sends the right commands to the printer, whether on the serial or parallel port.

At any time you address your printer as PRT:, this translation from a universal printer language takes place. You must transmit printer commands to PRT: in Esperanto. As soon as we revised our ABasiC program to do so, it worked well. Then John Toebes suggested we simply address our printer as PAR: (for the parallel port where it is connected). Aha! Printer commands right out of the printer manual worked. So, you do not need to learn Esperanto. You may bypass translation by using PAR: or SER: as the filename of your printer instead of using PRT:. Oh, well. Scratch one Saturday afternoon.

Fonts Typography on computer screens is both a science and an art, an art badly executed on many computers, and not well handled on Amiga. The fonts range from barely acceptable to downright ugly. We won't dwell on why most fonts are bad; those interested in screen typography may read an excellent article in BYTE by Charles Bigelow (Jan. '85). Most of the fonts on Amiga are better suited for a VIC 20 than for Amiga--which is supposed to be a class act.

In the reproductions which follow, recognize that our printer cannot match the definition of the screen. The samples suffer also from being duplicated. On the screen, fonts are much better defined than what you see below.

Typography is an ancient art, perfected over centuries both for beauty and to to match the natural frequency response of the human eye as it scans a page. The serifs, shapes and thicknesses of each stroke, and the relation of strokes to each other, have been reduced to rules which scientific test, not opinion, has shown that you break at your peril. The people at Apple followed the rules and did a first rate job of the fonts for Macintosh, which are both readable and beautiful.

A long_file_name in Topaz 8.
And a T at Underbar. Can you
see both underbars? Now we try
capitals: WHILE UNDer the bar...
Note the descenders next: qpp and
what happens to ascenders: hk...
This also occurs on screen.

Proportional Ruby 8.
Is this 'k' upper or lower?
K! Like William's ells? Is
this 'T' okay? Is P on
stilts in POMP? Is 'z' a
lower case Z or X? Is 's'
sassy or sad?

Judge the samples above for yourselves. See how the underlines "_" in the sample "long_file_name" are obscured by the capital letters on the next line, how asc-

enders and descenders (q descends, b ascends) in two lines merge. Such merges have several times caused us to completely misread screen text. For the curious: the hermaphroditic character above is supposed to be a "z". Such are the wages of designers who seek to create "something new and unique" in fonts. They do succeed--with the uniquely bad. The two fonts above are the best of a bad lot.

Luckily, the fonts are on disk. We fervently hope Commodore redesigns all. We can live, if uneasily, with Topaz. The rest make us ill.

Clumsy, Clumsy Editing Previous Commodore machines possessed the simplest and best screen editing capability of any computers, bar none. Amiga doesn't. We've so far used five different editors in Amiga, from that in Amiga DOS (CLI), to that in ABasiC. Every one is different, responding to a largely different command set. All are clumsy. Most evidently, the folks who wrote Amiga's software simply don't know how to write editors. In CLI, for example, where you issue DOS commands, you may not cursor back to revise a command. You backspace (and erase) to the error and then retype the rest of the command. You may not cursor up, revise, and re-issue a previous command. You type it all over again. Curses!

The editor in Notepad (an icon-based, simple text processor) obeys only two commands: CONTROL L to clear the screen, and BACKSPACE to delete characters or to join lines. If you cursor back and overstrike, you INSERT a character. The DEL key does nothing. RETURN splits a line at the cursor as it does in all editors except that in CLI. It maketh a strong man weep.

ED, a screen editor, uses CONTROL keys for major functions. You delete a line with ^B, erase to end line with ^Y, clear screen with ^L, cursor to end-line with ^]... Well, of course, these keys perform their ASCII functions (^L is ASCII 12), but the keys bear no mnemonic relation to what they do; you must learn them by rote. Shades of WordStar, which we gave away five years ago for that reason among others, when we went to a Commodore machine to get a sensible editing capability! Amiga's commands makes editing easy for developers who are accustomed to the ^key approach, but old Commodore hands will stand aghast.

We recognized that some of our dislike arises from familiarity with BEDIT, so we did not write this until we'd had a month of practice on Amiga. After the month, we conclude that nobody at Commodore-Amiga knows how to edit the easy way. And, obviously, nobody knocked programmer's heads together so that all editors use the same commands or a subset of them.

The old Waterloo microEDITOR is far superior to anything in Amiga. Sigh. We're glad Joe Bostic is hard at work on a new editor.

In Summary We like Amiga, despite its flaws. It has the potential of becoming a superb machine. The good far outweighs the bad. We've seen nothing major wrong with the hardware. Because all else is on disk, because all else is software, nothing is set in concrete. We expect to see substantial improvements in every feature we find wanting. It would have been easy to write (as does Amiga World, the new magazine devoted to Amiga) a fawning and uncritical review, praising the Amiga to the skies. Well, nothing is improved if it is deemed perfect. We may upset some folks at Commodore-Amiga by this criticism, but our readers expect the truth as we see it. We fervently hope that the things which are wrong are fixed.

BENCHMARKS ON AMIGA We won't repeat the benchmark listings, which have been published previously in the Gazette. You'll find references to the issue and page of the listings in the material following. Recognize that the results of benchmark tests depend more on the language and on the type of benchmark than on the machine. Whereas ABasiC on Amiga is relatively slow, C is speedy indeed on all things but floating point.

The results below are in single precision floating point, except for Amiga and HALGOL (we are not sure about the precision in HALGOL). The single-precision results are compared on the first line; ABasiC's double-precision result is on the second. We use BYTE results for Turbo Pascal and True Basic.

BYTE Calculation Benchmark (Vol. II, pp. 157, 159)

	ABasiC	HALGOL	6809 SPET	Turbo Pascal(PC)	True Basic (PC)
Time (sec)	22	2.46	150	82.6	19.7
Error	-5.9605..E-8	0	0	-1.3384..E-8	-4.583..E-13
Error	0	(time: 30 seconds in double precision)			

TRANSCENDENTALS

Dr. Dobbs Benchmark (As Modified by Terry Peterson. Vol II, p.4, p.157)

In either single or double precision, ABasiC does not compare well with the others in accuracy. SPET does far better. Nor is the speed impressive. The lack of accuracy in the double-precision run astonishes us.

	ABasiC (Single)	ABasiC (Double)	HALGOL	6809 SPET	Mac Basic
Time (sec)	44	48.5	16.8	780	586
Error	1.63425E-5	1.63426E-5	4.1006E-12	6.1153E-7	Not Known.

(Errors rounded to save space)

BYTE Sieve of Eratosthenes (Set to 7000, not 8190; one iteration)

Well, let's be simple-minded, and say SPET runs at 1 MHz, and Amiga at 7. Gee, ABasiC should run the benchmark in one-seventh the time of SPET, right? Well, uh, er...wrong. ABasiC runs it 3.17 times as fast. Note what a 4 MHz IBM PC does in Better Basic or Turbo. HALGOL is 29.6 times as fast; same chip, same program.

				All on an IBM PC		
	ABasiC	HALGOL	mBasic:SPET	Better Basic	Turbo Pascal	PC Basic
Time (sec) [integers]	53	1.79	168			
Time (sec) [reals]	64		243	31.4	15.4	190.7

BENCHMARKS IN LATTICE C

BYTE Sieve of Eratosthenes (set to 8190, 10 iterations)

There is no doubt that C, hands down, beats all languages except assembler in running the Sieve. Compare the times below, for 10 iterations, with the best of the Basics on one iteration (remember that C handled an array of 8190, not one of 7000 used for the Basics).

Data taken from our own tests, from BYTE, November, 1985, and from Dr. Dobbs, August, 1985 (for the Leading Edge PC, a close IBM compatible, running at 7.16 MHz). The Aztec C compiler was the fastest on Mac, so we show Aztec results. Programs below (except where noted) used 'int' variables, which are 32 bits in Amiga. In all cases, we averaged four good runs, discarding any runs which were way off (we timed by a stopwatch; sometimes reflexes aren't too good).

All the times shown below (except for compile and link on floppy) were taken with only Amiga DOS loaded. Concurrent programs slow down execution. The time of 3:18 to compile and link from floppy is a minimum, no other processes running.

Runs Without RAM Disk, No Register Variables.

	Amiga (on floppies)	Macintosh (on floppies)	Leading Edge PC (on hard disk)
Compile and Link	3:51 to 3:18*	4:07 (Aztec)	Not known
Execute	5.9 seconds	6:20 (Aztec)	99 seconds (???) (Doubt this...)
Code	19,040 bytes	16,897 bytes	Not known

Runs With Register Variables (32-bit Integers)

Execute	3.9 seconds	3.88 (Aztec)	58.0 (???)
Code	19,016	Not known	Not known

Runs with SHORT Integers, Amiga (16 bits)

Execute	5.01 seconds	19,048 bytes	No register variables.
	3.9 seconds	19,028	With register variables.

Compilation/Linking on RAM Disk

When the Sieve was compiled and linked on RAM disk, time averaged 2:46. Of this, the compiler used 40 to 45 seconds; the linker (sloooow) took the rest. The time includes backing up the object file from RAM disk to floppy (about 5 secs.) as a normal precaution. All files were compiled and linked by a batch file to reduce timing errors.

The basic overhead in compiling and linking can be seen from the time taken to process the simplest file: [printf("Hello, world\n");]. This requires 2:20 on RAM disk. The Sieve, considerably longer, needs only 2:46. It may be that we do

not yet fully appreciate how to get the most from the RAM disk. Certainly a better linker will reduce these times drastically.

* The compile/link time varies in Amiga because different processes may be running in the background. Some programs were compiled/linked on floppies with at least one background process running. Others were run "bare". Note the difference in compile/link times. On RAM disk compilation/linking, only Amiga DOS was loaded. This insures a fair comparison with other machines.

FROM TEN TO FIFTY DIGITS OF FLOATING POINT ACCURACY

Dr. John Cordes of the Department of Physics, Dalhousie University, Halifax, Nova Scotia, Canada B3H 3J5, for his work in physics needed floating point routines of great accuracy. None being available for SuperPET, he began to write them last year, but encountered difficulty in passing the numeric parameters back and forth between his machine-language routines and the driving program in microBASIC, which acts as a user interface, pre-processes the data, and prints the results received from the ML routines.

Luckily, Associate Editor Loch Rose had written a routine called FIND_VAR, which will locate any variable in microBASIC and return its address or its value. We passed it along to John Cordes, who plugged it into his routines and solved the parm-passing problem. Since that time, Dr. Cordes has generated a number of programs in ML and in microBASIC. They enable you to enter from the keyboard the numeric parms to be handled, pass them to the ML routines, and return the answer or answers to the microBASIC program as language variables, where you may manipulate or display them at will.

Dr. Cordes has developed four sets of routines in ML, as shown at the left. You use the package which will suit your purpose. The lower the accuracy, of course, the faster a given calculation proceeds. He has also written several microBasic drivers, each for a different purpose. You load the ".mod" program you want at main menu, then load microBASIC and the driving mBasic program. There, you enter the numbers to be manipulated and the programs take care of the rest, returning the results to mBasic as named language variables.

10 significant digits
18 significant digits
40 significant digits
50 significant digits

We feel that these routines may be of great value to others who must manipulate numbers with great precision, and have prepared an ISPUG disk which holds the mBasic drivers, the working ML routines, and all source code. If you are not an assembly-language programmer, fear not. You may use the programs as-is. You need to know only how to load them and how to press the right keys.

We are fascinated by one program on disk, named "calc:bu", which is an on-screen calculator. In it, you may perform any arithmetic operation on any two numbers, from one iteration to as many as you wish. If you use the 50-digit package, the results show on screen to 50 significant digits of accuracy. Considering the accuracy, it is surprisingly fast. And it is certainly simple to use.

The "Cordes FP disk" is available from ISPUG, PO Box 411, Hatteras N.C. 27943, for $10 in either 4040 or 8050 format. Please state that format! Dr. Cordes continues his work, and we expect that he will have additional routines to handle the transcendentals (sine, cosine, etc.) available later this year.

T H E A P L E X P R E S S by REG BECK
Box 16, Glen Drive, Fox Mountain, RR#2, Williams Lake, B.C., Canada V2G 2P2

There is some confusion between the terms "function" and "operator" as found in APL. The following definitions may clarify them:

> A function is that which takes in one or more data objects (or "arguments") and returns new data (results).
>
> An operator is that which takes in one or more data objects or functions and returns a new function.

The role of functions is to manipulate data; that of operators is to manipulate functions. In Waterloo microAPL V1.1, we have 5 operators. These are reduction, scan, axis, inner product, and outer product. These operators produce whole families of new functions. In more powerful APLs, new operators have been defined. I.P. Sharp APL has three composition operators; IBM's APL2 permits users to define operators. Anyone interested in APL theory (is anyone out there interested?) may wish to obtain a copy of a paper by Jon McGrew of IBM, Kingston New York, "Using Defined Operators." It is available in Proceedings, Volume I (Applications Session), 1982 APL Users Meeting, from APL Press. "APL as a Functional Programming Language," by Eugene McDonnell, in the same source, is useful also.

Functional Programming is a term which describes a style of programming in which the application of functions to arguments is the only permitted form. With certain restrictions, APL can be made into a purely functional language. These restrictions are:

* don't use assignment
* use the direct definition form of function definition
* don't use global variables.

Assignment and variables are excluded because when these are present the program becomes time-dependent. That is, at any particular time parts of the program may have different values than at other times. Such programs do not obey many simple mathematical laws and therefore cannot be verified mathematically; i.e., it cannot be proven that they work. Mathematically proven programs have no bugs.

J. Backus (the father of FORTRAN) wrote a key paper on functional programming: "Can Programming be Liberated from the Von Neumann Style? A Functional Style and its Algebra of Programs.", Communications of the ACM 21, 8 Aug 1978, pp. 613-641. The conventional linear programming languages are thought to have basic defects in that they process data one element at a time and use assignment as a fundamental control form. Backus felt that APL was exempted from the first defect but was still imperfect because of its use of assignment. He also felt that APL didn't have enough operators. Modern, powerful APLs--such as APL2--overcome this problem. An article in BYTE, August 1985, "Functional Programming Using FP," by Harrison and Khoshnevisan, shows the current interest in these ideas. Some of the terminology and constructs in FP were borrowed from APL.

We can experiment with these ideas in APL, keeping the previously mentioned restrictions in mind. Only operators, functions and their arguments are used. No assignment literally means no variables are used at all. Arguments consist of numbers, characters, and/or constructs using idioms and other functions. This

style makes much use of recursive functions. Arrays can be entered using functions similar to MAT and FMAT (see II, #3, p.83). The example developed in the McDonnell reference is the generation of a table which shows all possible ways of choosing M objects from N different objects. The following two functions are McDonnell's solution:

```
J:(1,(⍺-1) J ⍵-1),[1]0,⍺ J ⍵-1:⍺∈0,⍵:(1,⍵)⍴⍺=⍵
COMB:((⌊⍺!⍵),⍺)⍴(,⍺ J ⍵)/(⍵×⌊⍺!⍵)⍴⍳⍵
      3 COMB 5
1 2 3          COMB HAS BEEN MODIFIED SLIGHTLY FROM THE ORIGINAL BY
1 2 4          THE ADDITION OF ⌊. THIS WAS NECESSARY TO ENSURE THAT
1 2 5          ⍺!⍵ ALWAYS EVALUATES TO AN INTEGER. DUE TO ROUND OFF
1 3 4          ERROR ⍺!⍵ SOMETIMES HAS .00000001 TACKED ONTO IT. AN
1 3 5          EXAMPLE OF THIS IS 5!6 WHICH EVALUATES TO 6.00000001.
1 4 5          THERE IS ANOTHER BUG OF SOME KIND FOR WHICH I HAVE
2 3 4          NO EXPLANATION. AN EXAMPLE OF THIS IS 5!9 WHICH
2 3 5          EVALUATES TO AN INTEGER, 126, YET RESULTS IN A DOMAIN
2 4 5          ERROR AS IF IT HAD EVALUATED TO 126.00000001. THIS
3 4 5          IS ALSO CURED BY USING ⌊. TRY (⌊5!9)⍴⍳126 WITH AND
               WITHOUT THE ⌊.
```

A non-functional solution to this problem would involve assignment within the defined function. McDonnell breaks the problem down into smaller parts, some using assignment, then eliminates the assignment and builds the smaller parts up to the final result. It is a very nice development. J is an example of a doubly-recursive function. Though slow in microAPL, it is faster than we would expect. The appeal of this style of programming is clear. The workspace is very clean, consisting only of functions. Data is passed directly to functions from keyboard or other source, processed and the results displayed or otherwise used. Nothing is stored in main memory, but simply entered when needed, used, then discarded. SuperPET, as a teaching tool, is ideal for introducing ideas such as these to students who will go on in computing and will meet them again.

IBM's APL Programming Guide Vector Operations," by Timothy P. Holls, (IBM G320-6103-0), is another source of APL idioms. This slim manual is an inexpensive and worthwhile addition to an APL reference library. It is devoted to expressions for string manipulation. Holls uses the term "kernal" to refer to certain brief APL operations which occur repeatedly in the techniques he develops for handling vectors. The following are examples of kernals from this source.

```
I←⍳⍴V                    ⍝GENERATING A VECTOR OF INDICES.
P←(K×W)⍴W↑1              ⍝GENERATING A LOGICAL (BOOLEAN) VECTOR DEFINING
      K←5                ⍝THE FIELD PARTITION P OF V, FOR K FIELDS OF
      W←3                ⍝SAME WIDTH W. WHEN THERE ARE NO FIELD DELIMITERS.
      P                  ⍝A 1 INDICATES THE FIRST ELEMENT OF A FIELD.
1 0 0 1 0 0 1 0 0 1 0 0 1 0 0
V←(,W∘.>⍵(⍳L)-⎕IO)\V     ⍝EXPANDING FIELDS WITH DIFFERENT WIDTHS W TO
      V←'AABCDABABC'     ⍝FIELDS OF THE SAME WIDTH L. L MUST BE ≥ ⌈/W.
      W←1 4 2 3
      L←4
      V                  ⍝EXAMPLE FOR RIGHT-JUSTIFIED FIELDS.
   AABCD  AB ABC
↑   ↑   ↑   ↑            ⍝ARROWS POINT TO FIRST POSITION IN EACH FIELD.
V←(((V≠Q)⍳1)-⎕IO)↓V      ⍝DELETING ALL LEADING OCCURANCES OF ELEMENT Q.
```

```
      V←0 0 0 22 3 0 14 0 0 57 0 0
      Q←0                    ⍝((V≠Q)⍳1)-⎕IO IS A "KERNAL" APL OPERATION.
      V                      ⍝IF Q IS A VECTOR OF DIFFERENT ELEMENTS REPLACE
22 3 0 14 0 0 57 0 0         ⍝V≠Q WITH ~V∈Q.
```

A field delimiter is a character, not normally used in a record, which marks the beginning or end of a field. Readers not up on file terminology should consult the definitions given in the last column. Delimiters are used in variable field records if additional information about the locations and lengths of fields is not kept, otherwise there would be no way to determine the beginning and end of each field in each record. Holls' manual is almost mandatory for anyone doing file work in APL.

Earlier, I named the five operators available in microAPL. The detailed definitions in the Waterloo APL manual are a bit obscure. Operators are used to create additional functions. The reduction operator (symbol /) takes any scalar dyadic primitive function, f, (Table A.2 in the Waterloo microAPL manual) and creates a mixed monadic function, f/. A scalar function is one which has scalar arguments, returns a scalar result and is extended to array arguments element by element. A mixed function takes a whole array as an argument and acts on it as a whole, not element by element. The following examples show how reduction works.

```
V←1 5 7 ¯8  2 5          ⍝DEFINES A VECTOR, V.
      +/V                ⍝PLUS REDUCTION SIMPLY ADDS THE ELEMENTS OF V.
12
      -/V                ⍝EQUIVALENT TO 1-(5-(7-(¯8-(2-5))))
8
      ⌈/V                ⍝FINDS THE LARGEST ELEMENT OF THE VECTOR.
7
B←4 8⍴⍳32                ⍝DEFINES THE ARRAY, B.
      B
 1  2  3  4  5  6  7  8
 9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32
      +/B                ⍝ADDS UP THE ROWS OF B.
36 100 164 228
      +⌿B                ⍝ADDS UP THE COLUMNS OF B. NOTE USE OF ⌿ HERE.
52 56 60 64 68 72 76 80
```

Scan (symbol \) takes the same primitive functions as reduction and also returns mixed monadic functions. Examples follow:

```
V←1 2 3 4 5 6 7 8        ⍝DEFINES A VECTOR, V. (COULD HAVE USED V←⍳8)
      +\V                ⍝PLUS SCAN OF V DOES A RUNNING SUM ON THE
1 3 6 10 15 21 28 36     ⍝ELEMENTS OF V.
      -\V                ⍝YOU FIGURE THIS ONE OUT!
1 ¯1 2 ¯2 3 ¯3 4 ¯4
B←4 8⍴⍳32
      B                     ⍝+\B DOES A PLUS SCAN ALONG THE ROWS.
 1  2  3  4  5  6  7  8
 9 10 11 12 13 14 15 16     ⍝+⍀B DOES A PLUS SCAN ALONG THE COLUMNS.
17 18 19 20 21 22 23 24     ⍝AGAIN NOTE THE ⍀ TO INDICATE A CHANGE OF
25 26 27 28 29 30 31 32     ⍝AXIS. SEE THE AXIS OPERATOR EXAMPLES.
```

```
      +\B                                    +⍀B
  1   3   6  10  15  21  28  36      1  2  3  4  5  6  7  8
  9  19  30  42  55  69  84 100     10 12 14 16 18 20 22 24
 17  35  54  74  95 117 140 164     27 30 33 36 39 42 45 48
 25  51  78 106 135 165 196 228     52 56 60 64 68 72 76 80
```

The axis operator (symbol [V]) indicates which axis (vertical or horizontal for two dimensional arrays) a function is to operate along. Examples follow:

```
B←4 8⍴⍳32                 ⍝SAME ARRAY AS BEFORE.
      +/[1]B              ⍝INDICATES WHICH AXIS TO SUM ALONG.
52 56 60 64 68 72 76 80   ⍝SAME AS +⌿B
      +/[2]B              ⍝SAME AS +/B. THE AXIS OPERATOR IS ORIGIN
36 100 164 228            ⍝DEPENDENT. HERE ⎕IO←1.
      +\[2]B              ⍝ALSO CAN BE USED WITH SCAN.
  1   3   6  10  15  21  28  36
  9  19  30  42  55  69  84 100
 17  35  54  74  95 117 140 164
 25  51  78 106 135 165 196 228
⍝THE AXIS OPERATOR IS ALSO USED WITH CATENATE AND LAMINATE TO INDICATE
⍝ALONG WHICH AXIS THE OPERATION IS TO TAKE PLACE.
A←2 3⍴1
C←2 3⍴⍳6
      A,[0]C          A,[1]C
1 1 1 1 2 3         1 1 1
1 1 1 4 5 6         1 1 1
                    1 2 3
                    4 5 6
```

We leave the last two operators, inner and outer product. for another time.

The problem of obtaining the best fit on a curve for data is easily solved in APL (see Frank Herr's article elsewhere in this issue). The following idioms can be used depending on the nature of the data.

```
*A+.×(⍟Y)⌹A←X∘.*0 1      ⍝PREDICTED VALUES OF EXPONENTIAL FIT
A+.×Y⌹A←X∘.*0 1          ⍝PREDICTED VALUES OF BEST LINEAR FIT
                          ⍝(LEAST SQUARES)
Y⌹X∘.*0,⍳G                ⍝G-DEGREE POLYNOMIAL FIT OF POINTS (X,Y)
A Δ A[1]←*A[1] Δ A←(⍟Y)⌹X∘.*0 1   ⍝COEFFICIENTS OF  EXPONENTIAL
                          ⍝FIT OF POINTS (X,Y)
Y⌹X∘.*0 1                 ⍝BEST LINEAR FIT OF POINTS (X,Y)
                          ⍝(LEAST SQUARES)
```

Many natural relationships are of the power-law type. Coulomb's law for charges states that the force which charges exert on one another decreases as the square of the distance between them. This rule also holds for the intensity of light from a point source. The intensity decreases as the square of the distance away from the source. Areas are related to squares of linear measurements and volumes are related to cubes. When experimental data is obtained in the measurement of one or another of these relationships, the most important quantity to determine is the power. Since experimental data contains uncertainties and errors the powers obtained are seldom exact integers. Graphing techniques are often used.

One such technique is to graph the logarithm of the Y data vs. the logarithm of the X data. If the relationship is a power-law, the slope of the resulting linear graph will be the power in the relationship. The form of a power-law relationship is shown at left, where n is the power and k is the constant of proportionality. If we take the log of each side (to any base) of this equation, we obtain: log Y = log K + n(log X). Comparing this equation with the equation of a straight line in slope-intercept (Y = b + mX), we see that the slope of the log-log graph is indeed n, the power, and that the intercept is the log of the constant of proportionality. The slope will be quite accurate; it will give us an accurate value for the power. The constant of proportionality is subject to much greater innaccuracy because the intercept is the log of it and a small variation in the logarithm of a number can mean a large variation in the number itself.

$$Y = kX^n$$

Doing this with APL is simple. Use Frank Herr's program with a degree of fit of 1 and the logarithms of the X and Y data. The first coefficient is the logarithm of the constant of proportionality and the second is the power. An example follows:

```
DATA:
      Y←3.0 6.4 11.7 16.5 22.4
      X←1.0 1.3 1.6  1.8  2.0

      ∇FIT[□]∇                    ⍝FRANK HERR'S POLYNOMIAL FIT PROGRAM
[  0]    Y FIT X
[  1]   'ENTER DEGREE OF FIT'
[  2]   COEF←Y⌹X∘.*0,⍳DEG←□

      (⊛Y) FIT ⊛X                 ⍝LOGARITHM OF X AND Y DATA
ENTER DEGREE OF FIT
□:
      1                           ⍝LINEAR (LEAST SQUARES) FIT OF LOG DATA
      COEF
1.09709848 2.90141709

      THE POWER IS 2.9
```

I recently wrote IBM's publication office and requested a listing of their APL publications. They overwhelmed me with eight pages containing 469 titles! Some of the titles are for internal use and many are specific to IBM implementations. It will take a while to sort through these and select some of general interest. Unfortunately they never sent the prices. I hope to report on some of these publications in future columns.

NEW GRAPICS PROGRAM for THINKJET
and
Bar-Graph Application Programs for the 8023 and ThinkJet

In Volume II, No. 3 (p. 74), we announced the issue of SPGP (SuperPET Graphics Program) for the 8023 printer. SPGP allows you to create (in 6502 mode) graphics designs which you may output to printer to the limit of printer resolution. Delton B. Richardson, the author, has laid hands on a Hewlett-Packard ThinkJet printer, and has issued a similar disk for that printer. Though most of the code is in 6502 machine language, you needn't know anything about ML to use the programs.

The program runs in 6502 mode, with a 4K program of machine language in user memory, plus a BASIC 4.0 driver. The 64K of bank-switched memory is used to store the images generated, which can consist of 512 x 768 pixels (393,216 of them) for an 8.5 x 11 inch page. The BASIC driver is itself menu-driven; you merely select your options. Graphics are printed on the ThinkJet at 96 dots per inch resolution vertically and horizontally.

You may design your own images and character fonts or develop your own applications programs. Graphics and text may be combined; images may be saved both to disk and to printer. The disk holds all source code as well; you're free to revise it. Assembler files may be read in the microEditor in 6809.

On disk are both instructions (for reference) and a tutorial on how to use the programs. The material almost fills an 8050 disk (35 Blocks free); we can't issue it in 4040 format. Instructions, tutorial, and menu-driven program walk you through the program step-by-step and are of a quality well above the usual public domain disk because Delton has issued the disk as FREEWARE.

ISPUG distributes it under the name of SPGP Master (ThinkJet) for $10; write ISPUG at PO Box 411, Hatteras, N.C. 27943. If you like what Delton has done, and feel the author should be rewarded, Delton asks a $20 contribution. You are not obligated to send anything. There's a similar ISPUG disk for the 8023 printer, SPGP Master (8023). We emphasize that these Master disks are for those who want to design their own graphics or to develop their own applications, either for the ThinkJet or for the 8023.

An application, on the other hand, is a program which uses the tools provided in one of the Master disks to generate a specific program tailored to one job. One such specific application is now available from Delton: SPGP (Bar-Graph : ThinkJet & 8023). It will print bar-graphs of almost any size you can put on printer paper, and do this automatically; you merely specify the bar sizes, numbers, and titles. The disk holds separate programs which drive either a ThinkJet or an 8023. The graphs are printed "sideways", so that you may have up to 40 bars on a single graph without running off the paper. As you get more bars, the titles you can give to each grow shorter (the bars get narrower, and space for long bar titles simply isn't there).

Shown on the last page of this issue is such a graph, which took all of ten minutes to make. We loaded the program, titled the graph, entered the bar data, and said "print it."

The applicaction disk, SPGP (Bar-Graph : ThinkJet & 8023), is separately available from ISPUG for $10. You do not need either of the SPGP Master disks to use the Bar-Graph disk (unless you plan to modify the graphs or programs). The Bar-Graph disk is complete with its own instructions, ML program, and BASIC driver--which operates from menu. Load it, tell it what bar graph to draw--and it draws it. You may store and retrieve images on disk. This disk is available in 8050 and 4040 formats (masters come only in 8050). If you order, PLEASE state format.

CURVE FITTING DATA WITH APL ROUTINES

by Frank Herr, 20821 Germain St.
Chatsworth, CA 91311

APL offers the capability to perform curve fitting with a least squares fit. The form of the equation fit is found at left, below. APL lets us simplify curve fits by making the fit and evaluating the accuracy of the results relative

to the original data. If the fit does not meet our desired accuracy, we simply re-run the program for a higher-order equation (i.e., enter a higher degree of fit). The higher the order, however, the more data you may require. A curve fit program in APL follows:

```
Y = A + BX + CX^2 + DX^3 + ...NX^(N-1)
```

```
[ 0]   Y FIT X
[ 1]   'ENTER DEGREE OF FIT'
[ 2]   COEF←Y⌹X∘.*0,⍳DEG←⎕
```

You select the order of fit after the program begins. You must enter data for the program when you call it. Y values you list directly (or specify as a variable) preceding the program name; X values you enter after the program name. A typical call looks like this:

```
2 3 4  FIT  1 2 3   OR  (Y←2 3 4) FIT X←1 2 3
```

You obtain best results by using variable values, because the data remains in memory for instant recall for another fit, if required. Once a command such as that above is entered, the computer asks for the degree of fit and then defines fit coefficients. How closely does the resulting equation fit the input data? Fit evaluation is only a short program away:

```
        ∇FITEVAL[⎕]∇
[ 0]    FITEVAL X
[ 1]    YY←(X∘.*0,(⍳(⍴COEF)-1))+.×COEF
[ 2]    ERR←YY-Y
[ 3]    ⍉(5,(⍴X))⍴X,Y,YY,(ERR÷Y),ERR
```

In the program above, all lines can be combined into one of the famous APL one-line routines, but they are clearer as stated. To use the routine above, you must have entered data using the second method (data entered as variables). In this case, simply call: FITEVAL X . The result is a printout of the original X and Y values, the new Y values (YY), the fractional error relative to the Y input data, and the increment error as the computed Y value minus input Y data.

The power of APL in formatting output is illustrated on line 3 of the last example, where the transpose function, ⍉, is used to convert the catenated string of data (X, Y, YY, (ERR/Y), and ERR) into five columns of data which have the above titles.

The routines above do not require you to specify the dimensions of any inputs or the number of coefficients to be generated. APL counts the number of data items input as the program executes. There are no loop statements to handle consecutive data elements; APL uses inputs as arrays in single computations.

The two routines above allow fast curve fits; they quickly evaluate the accuracy of the fit and select the lowest order equation that meets required accuracy.

REMEMBER THE LAST MOVE AND TRY AGAIN
an Exercise in Recursion and Local Variables

We're deeply indebted to Frank Brewster for the program which follows, and for his patience in helping us to fully understand how it employs local variables and recursion to accomplish a most complex job in very little code. It is based

on a program in Pascal from Nicholas Wirth's book, "Algorithms + Data Structures = Programs." Frank translated it from Pascal into structured microBasic. It addresses this question: Can a chess knight, starting from a given position, and moving only in the ways permitted to a knight, reach every position on a chess board once and only once?

The answer is far from simple to find. A knight can move in eight different ways from any position (variations on + or - 2 columns or rows, + or - 1 column or row). We show the eight possible moves from the central square on a 5 x 5 board at the left. B is the beginning position; the eight X's show all possible moves. Because each move to a new position generates eight possible new moves, which in turn generate eight more possible moves from each of eight positions, the number of different combinations of moves which can cover even a 4 x 4 board is huge. (Hmmm. If we exclude the starting square, we must explore 8 possible moves on each of 15 remaining squares except the last, but some of the moves will take us off the board, and some squares we may not reach. Care to calculate the possible sets of moves? The number is somewhere below 8^14. If anyone knows the right number, we're curious enough to want it.)

		Columns				
		1	2	3	4	5
R	1	0	X	0	X	0
o	2	X	0	0	0	X
w	3	0	0	B	0	0
s	4	X	0	0	0	X
	5	0	X	0	X	0

Assume for a moment we attempt some moves by hand on a 4 x 4 board; that we are able to reach 15 of the 16 squares and then cannot make a legal move. We must then backtrack to our previous position and try a different move. If that fails, we must try another and different move. If, after eight different moves, we're still trapped, we must back off two moves and try the seven remaining moves at that position--and, if that doesn't work, we must go back again and try at a position one more level on our backward track...

Would you care to write a program which will remember all its previous moves, which of the legal moves it has made at any position, which moves remain open--and which will backtrack, whenever it is trapped, to the proper previous move and try again--until all possible moves on the board have been tried or until it succeeds in filling the board?

The short program below does this, and does it simply and well using recursion and local variables. It makes a series of moves by recursion; if that series fails to fill the board, the recursive calls "uncoil" (as in our previous example) and return us to our next and untried legal move on the last move which was successful. The "remembering" of moves which were tried is handled by the varying values of local variables. Backtracking is handled by recursion itself in combination with these local variables.

Be warned: you will not understand what happens or how until you type in the program and run it. It outputs the board to the screen by pass and move number; you can see recursion occur, see it uncoil, see the program return to positions and try again. The program also saves the best board (the maximum number of moves it has made to date). It makes 87 passes to tell you that there is no solution to the problem on a 3 x 3 board. We ran it to some 11,000 passes on a 4 x 4 board (for which there is no solution either), and were far from done. You can pause the program at any time with STOP to examine variables, and then resume with "cont <RETURN>". The variable "size%" can be changed for any size of board; it is set for a 3 x 3 board below.

```
 10 ! analysis3.tour. A knight's tour of a 3 x 3 board with screen printout.
100  size% = 3 : sizesq% = size% * size% : CS$=chr$(12)
110  option base 1: dim a%(8),b%(8),board%(size%,size%),best%(size%,size%)
140  a%(1)= 2: b%(1)= 1
150  a%(2)= 1: b%(2)= 2       ! The eight possible moves from any
160  a%(3)=-1: b%(3)= 2       ! position, set down in relative terms
170  a%(4)=-2: b%(4)= 1       ! from the current row and column position.
180  a%(5)=-2: b%(5)=-1       ! Initially, the board matrix contains zeros.
190  a%(6)=-1: b%(6)=-2       ! As successful moves are made, the move number
200  a%(7)= 1: b%(7)=-2       ! is recorded at that position on the board.
210  a%(8)= 2: b%(8)=-1
310  startx%=1: starty%=1                        ! We start on row 1, col. 1.
320  board%(startx%,starty%)=1 : print CS$;
330  call try(2,startx%,starty%,trynbr%,0,0)  ! The last three parms at call
350  if ok%                                   ! are dummies, with values of 0.
380    mat print board%
390  else                          ! In proc try, "x" designates the row and "y"
400    print "No solution..."      ! the column at which we start for any move.
410  endif                         ! "movenbr" records the number of successful
420  stop                          ! legal moves made to date; it never exceeds
430                                ! the number of squares on the board. "trynbr"
440                                ! records the number of tries since the last
450                                ! successful move; it cannot exceed 8.
460
470 proc try(movenbr%,x%,y%,trynbr%,nextx%,nexty%)
480  trynbr% = 0
490  loop
500    trynbr% = trynbr%+1 : ok%=0 : pass% = pass% + 1
510    nextx% = x% + a%(trynbr%): nexty% = y% + b%(trynbr%)
520    if nextx% >= 1 and nextx% <= size% and nexty% >= 1 and nexty% <= size%
530      if board%(nextx%,nexty%) = 0
540        board%(nextx%,nexty%) = movenbr% : ok% = 1
550        print "Pass:"; pass%; "Move #:"; movenbr%; mat board%;
560        if movenbr% > bestboard%
570          bestboard% = movenbr% : print "Best="; bestboard%
580          mat best% = board%                    ! SAVE BEST BOARD TO DATE.
590        endif
600        if movenbr% < sizesq%                   ! RECURSIVE CALL.
610          call try(movenbr%+1,nextx%,nexty%,trynbr%,nextx%,nexty%)
620          if ok% = 0 then board%(nextx%,nexty%) = 0
630        else                                    ! Note that the board is reset
640          ok% = 1                               ! to zero for ALL moves which
650        endif                                   ! led to a trap as recursion
660      endif                                     ! uncoils, at line 620.
670    endif
680  until ok% or trynbr% = 8
690 endproc
```

The program above is a classic example of the power of recursion and local variables. We again urge you to enter and run it to fully appreciate what happens.

The original program, in Pascal, as published by Wirth, runs to 47 lines of code (without comments or printout of each move). The microBasic program above occu-

pies 37 lines (excluding comments and printout). Brevity is elegance?

PRINTER BUFFERS BEWARE!

A friend of ours bought this great new printer and equipped it with a 64K printer buffer. He thought he could dump any document he generated to the buffer and let the printer chew away at it whilst he worked on the computer. As with all simple solutions tions to complex problems, this one fails if anything at all goes wrong at the printer. Our friend was printing mailing labels, and one peeled off and stuck in the platen... Later, he ran out of paper in the middle of a run; after that, his sheet feeder malfunctioned. Did the printer have a button to reset to the start of the butched-up page? Sorry, Charley. Stop what you are doing; get back the file; find the bad place, retransmit the file starting at the top of the butched page. Then get back the file you were working on...

A lot of new printers have big buffers and offer optional bigger ones; think about it before you jump for a large buffer. If you do get a big buffer, be sure that you have the right controls. Many separate printer buffers give you the option to reprint, starting at the top of any page--but what do you do if your pages aren't numbered (as with mailing labels)? It's best to think about such problems before you sink a lot of cash into a big buffer of any kind. And, while we speak of printer buffers:

We had a few calls for an old echo print routine we used a long time back when we hadn't given up filling out forms on a computer printer. Whoever designed the printers we use never heard of forms; the printers boast no line alignment guides; the printhead covers up whatever box you attempt to print in. You are therefore required to own a typewriter as well as a computer printer (when will they marry the two in one machine which doesn't poke along at 10 or 15 cps?). Forms, as with death and taxes, are a certainty. If you're lucky enough to have a printer which will handle forms, you need an echo print routine to send each key, as touched, directly to printer and screen. Gee, it's easy to write one (at left, below):

```
loop
  loop
    get char
  until char >0
  if char=255 then quit ! OFF key
  print #printer, char;
endloop
```

It works fine on our DIABLO, but fails miserably on our ThinkJet--nothing ever seems to print. Turns out the TJ has a big buffer, and never prints a single line until either 1) the file is closed or 2) the buffer is full. We mentioned the problem to guru Terry P., who said he would expect that same problem on any EPSON printer, and probably on many others. The answer, of course, is to print the character, a space, and a backspace, all without a carriage return--to which most printers will respond by immediately printing a character and backspacing to the proper position for the next. But--what do you do with printers which cannot backspace? Don't buy one! Printer buffers, in sum, can pose some real problems and are not the unqualified answer to a maiden's prayer.

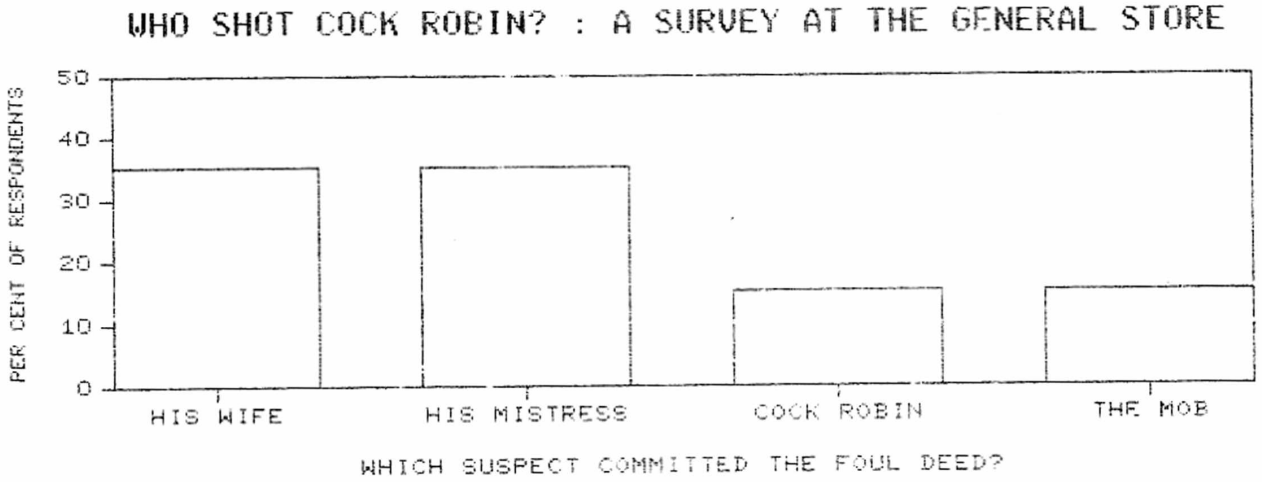
```
print #4, char; " ";chr$(4);
```

Why not print without a buffer? If you watch a good, logic-seeking printer at work, you'll note it prints bi-directionally, skips large groups of spaces instead of printing a space at a time, always CR's at the end of a short line, starts a line holding a centered caption at the first character (not at the left or right margin), etc. Such logical shortcuts speed up printing and demand

a buffer of some sort so the printer has sufficient data to analyze before it prints. 512-byte buffers are plenty big enough for printers which handle 80-character lines. Anything more may be a blessing or a pain in the behind.

NEW BEDIT OFFERED NEXT ISSUE! If you are thinking about ordering ISPUG Utility Disk II and BEDIT, please hold off until next issue. Joe Bostic has just about finished a new version. Yeah, you can set margins, FORmat text to those margins, split and join lines, cursor to the end of a word, sentence, or paragraph directly, and even output paged text to your printer with page numbers and with headers and footers if you want them...

Bar-Graph Sample: The graph below was created on our ThinkJet printer, using Delton B. Richardson's bar-graph program for that printer, which is offered on disk elsewhere this issue.



Prices, back copies, Vol. I (Postpaid), $ U.S. : Vol. I, No. 1 **not** available.

No. 2: $1.25	No. 5: $1.25	No. 8: $2.50	No. 11: $3.50	No. 14: $3.75
No. 3: $1.25	No. 6: $3.75	No. 9: $2.75	No. 12: $3.50	No. 15: $3.75
No. 4: $1.25	No. 7: $2.50	No. 10:$2.50	No. 13: $3.75	Set: $36.00

----------Volume II----------

Numbers 1 thru 8: $3.75 each.

Send check to the Editor, PO Box 411, Hatteras, N.C. 27943. Add 30% to prices above for additional postage if outside North America. Make checks to ISPUG.

==========

DUES IN U.S. $$ DOLLARS U.S. $$ U.S. $$ DOLLARS U.S. $$ U.S. DOLLARS $$

APPLICATION FOR MEMBERSHIP, INTERNATIONAL SUPERPET USERS' GROUP

(A non-profit organization of SuperPET Users)

[] Check if you're renewing; clip and mail this form with address label on the reverse side. If you send the label, don't fill in the form below.

Name:_____ Disk Drive: _____ Printer:_____

Address:_____

Street, PO Box City or Town State/Province/Country Postal ID#

For **Canada** and the U.S.: Enclose Annual Dues of $15:00 (U.S.) by check payable to ISPUG in U.S. Dollars. DUES ELSEWHERE: $25 U.S. Mail to: ISPUG, PO Box 411, Hatteras, N.C. 27943, USA. **SCHOOLS!: send check with Purchase Order.** We do not voucher or send bills.

This journal is published by the **International SuperPET Users Group** (ISPUG), a non-profit association; purpose, interchange of useful data. Offices at PO Box 411, Hatteras, N.C. 27943. Please mail all inquiries, manuscripts, and applications for membership to Dick Barnes, Editor, PO Box 411, Hatteras, N.C. 27943.

Table of Contents, Issue 8, Volume II

SuperPET Gazette
PO Box 411
Hatteras, N.C. 27943
U.S.A.

First Class Mail
in U.S. and Canada.
Air Mail Overseas.