Open-Apple™

December 1987
Vol. 3, No. 11

ISSN 0885-4017
newstand price: $2.00
photocopy charge per page: $0.15

Releasing the power to everyone.

Living Languages

PROMAL Reviewed

by Dennis Doms

If there is one factor affecting the development of software for the Apple II, it's the search for a true high-level (that is, higher than assembly language) development language for the computer. Applesoft Basic, the high-level language supplied in ROM with the Apple, lacks certain features that make writing programs of any appreciable size easy.

Programmers complain about the "sloth" of Applesoft and its lack of "programming structures," such as IF/THEN/ELSE and DO/WHILE. Most programming structures can be simulated by creative programming, though, and speed is often not as much an issue as being able to write a readable, bug-free program while minimizing high blood pressure.

To my mind, one of the most serious failings of Applesoft is that it relies on indexing its program logic by line numbers to transfer control. Therefore, you have to remember what a subroutine at line 1000 does while typing in "GOSUB 1000" to execute the code, lest you invoke a set of lines defining a printout routine when you meant to open a disk file. There's nothing mnemonic about a line number. In small programs, this is an irritation; in large programs with 20-30 or more subroutines, you have to use your memory as an extension to Basic to remember where you put things in the program.

The use of *line* numbers for indexing also makes SAVEing a general purpose subroutine so that it can be used in other programs an exercise in patience, requiring a "hold and merge" utility that lets you "hide" the main program in memory, load the subroutine into memory, renumber the subroutine so that its line numbers don't clash with those in the main program, then "unhide" the main program and merge the renumbered subroutine into it. If you did happen to know what the line number of the original subroutine was, it was almost certainly changed during this process.

Also, unless you have total recall, there's a possibility that some variables in the original (main) program may clash with the newly added subroutine. Which means you not only have to remember what all the (changing) line numbers stand for, you also have to remember which variable names are in use and (when changed) what they stand for. The larger the program, the more of a problem this becomes.

One solution is to allow the naming of a subroutine so that it can be accessed within the program by stating its name. Therefore, "GOTO 1000" might become "PRINT.STRING". Second, we'd like to be able to name variables used in the subroutine separately from the main program, so that a variable "ASTRING" used in the subroutine would not change the value of a variable of the same name in the main program, unless we wanted it to.

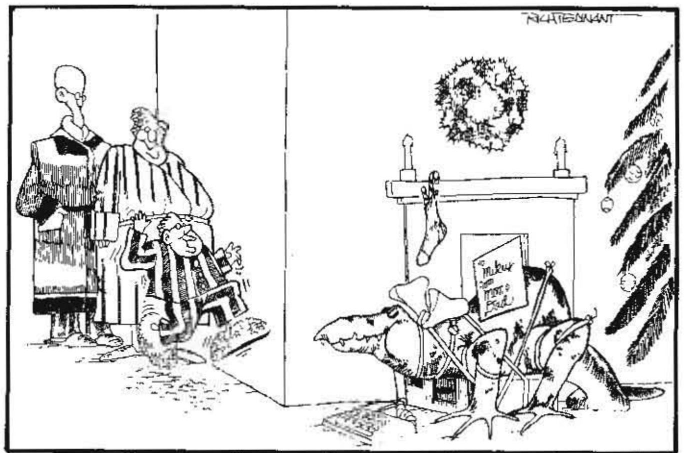
There are a few programming languages on the Apple that allow this, but they may use different methods to achieve the effect.

PROMAL is such a language. Maybe the easiest way to look at the philosophy of the language is to analyze a small program, where it's easy to see we're not in Basic anymore (the text following a ";" on a program line is a comment, similar to a REM in Applesoft):

```
PROGRAM CALC
; Floating point calculation benchmark for PROMAL
; (based on June 1984 Byte, p. 336)

INCLUDE LIBRARY                     ; for some extra PROMAL functions
INCLUDE DJD/PRTIME                  ; for "print current time" routine

DATA REAL A = 2.71828               ; define a few variables
DATA REAL B = 3.14159
REAL C
WORD COUNT                          ; including our loop counter
DATA WORD NREPS = 5000              ; and number of repetitions

BEGIN                               ; CALC
OUTPUT "Starting calculations at "  ; print starting time
PRTIME
PUT CR                              ; follow with carriage return
C = 1.0                             ; start with C = 1.0
FOR COUNT = 1 TO NREPS              ; then manipulate it a while
  C = C * A
  C = C * B
  C = C / A
  C = C / B
OUTPUT "Calculations done at "      ; print completion time
PRTIME
PUT CR
OUTPUT "Error = #E",C-1.0,CR        ; and amount of error created
END                                 ; CALC
```

First, unlike Basic, we have to describe ("DIMension," in Applesoft terminology) all variables before they are used. We can define variables as being REAL (floating-point), WORD (a two-byte positive integer from 0 to 65535), INT (a two-byte integer from -32768 to 32767), BYTE (a one-byte value from 0 to 255) or BOOLEAN, (a two-state value such as TRUE and FALSE).

Second, and very easy to spot, we have ~~no line~~ numbers. The flow of control in *PROMAL* is determined by certain keywords (some disarmingly familiar) in conjunction with the *indentation* of the lines. For example, there are four



"I JUST HOPE THIS WILL PUT TO AN END ALL THE MOPING AND WHINING ABOUT NEEDING A MONITOR FOR HIS COMPUTER."

indented lines containing simple mathematical expressions following the line "FOR COUNT = 1 to NREPS". The FOR loop in this case executes in a manner similar to a loop in Basic, but *PROMAL* determines which statements are inside the loop by their indentation. The four indented expressions are executed over and over while COUNT increments (by 1) from 1 to NREPS, then the FOR "falls through" to the following OUTPUT statement. Notice there's nothing in this program similar to Applesoft's NEXT statement—indentation takes its place.

Using indentation as part of the control structure takes some adjustment if you're a die-hard Basic programmer, but it has a good side affect: it strongly encourages you to write structured code that you (or another programmer) can read later. In this matter, *PROMAL* is more adamant about structure than some other languages; as an example, indentation is normally *used* in writing Pascal or C programs but is not forcibly *required* as in *PROMAL*. Even other languages I have used on the Apple that are stringent about structure, such as Forth and LISP, do not do as much to force you to write readable code.

The third difference from Basic is that the command words are different; BEGIN, OUTPUT, and PUT are not part of the standard set of Applesoft statements. BEGIN proclaims the beginning of a grouped set of instructions that can define a PROGRAM, PROC (procedure; a subroutine that does not return a value), or FUNC (function; a subroutine that returns a value or values). END in *PROMAL* is (unlike END in Applesoft) required to define the end of a PROGRAM or segment (PROC or FUNC).

Pairing the use of BEGIN and END to define program segments allows us a luxury in *PROMAL* that is not allowed in Basic; the *easy* definition of a series of general purpose routines for inclusion in other programs. What makes *PROMAL* better than Basic in this regard is that the routines are accessed painlessly by name rather than called by a line number. And another problem solved: unless you explicitly make them otherwise, variables declared and used within in PROCedures or FUNCtions are manipulated separately from those in the main program or other procedures, even if the variables have the same name. That is, we can have a variable named COUNT within a PROCedure and do all sorts of things to it without affecting the contents of another variable named COUNT in a separate PROCedure, FUNCtion, or our main PROGRAM. Incidentally, variable names can be up to 31 characters long with all characters significant (not just the first two as with Applesoft).

This modularity feature is a reason for going through the pain of learning a new programming language; you can speed up Applesoft, and you can simulate DO/WHILE and other structured loops, but you can't get rid of the line numbers. Only by going to a new version of Basic or a whole new language can you escape the dreaded "hold/merge" required to make Applesoft somewhat modular.

An example of the power of modularity: while the FOR loop and BEGIN/END are part of *PROMAL*, some commands such as OUTPUT and PUT are not part of the basic core of the language. OUTPUT and PUT are, in fact, subroutines defined in the *PROMAL* source file "LIBRARY," which is added to the source file for the program CALC by the program line "INCLUDE LIBRARY." This line causes the *PROMAL* compiler to effectively insert the text of the source file "LIBRARY" and compile it as if the text were included in the file for PROGRAM CALC at that point. LIBRARY includes many routines needed for normal programming tasks and is provided as part of the language package.

The PRTIME statement refers to a routine I wrote to print the time at the beginning and end of the benchmark code:

```
PROC PRTIME
: prints time from ProDOS-compatible clock to nearest second.
:            * expects Thunderclock-compatible format! *

EXT BYTE TARRAY[] AT $209          ; start of time string for HD or TC
EXT ASM PROC JSR AT $0FB4          ; PROMAL M/L interface

OWN WORD TPOINTER                  ; used for indexing
DATA BYTE DOGETTIME[] =            ; M/L for get_time call
  $20,$00,$BF,                     ; JSR ML1
  $82,                             ; DFB GET_TIME
  $00,00,                          ; DW  PARMLIST
  $60                              ; RTS

BEGIN                              : PRTIME
JSR DOGETTIME                      ; get time string into buffer
FOR TPOINTER = $00 TO $0B          ; clear high bits
  TARRAY[TPOINTER] = TARRAY[TPOINTER] AND $7F
TARRAY[2] = ':'                    ; change commas to colons
TARRAY[5] = ':'
TARRAY[8] = $00                    ; terminate string with $00
PUT TARRAY                         ; ...and print it
END
```

This code is more obscure than the CALC program, but it shows an important feature of *PROMAL*—the ability to access the Apple through definition of PROCedures incorporating machine language routines. The byte array DOGETTIME is defined to contain a series of hexadecimal values (yes, you can enter hex numbers into *PROMAL*) that provide the machine code to execute a ProDOS GET_TIME call. We also declare an EXT ASM PROC (external assembly procedure) named JSR at memory address location $0FB4; this routine is actually part of the *PROMAL* runtime support that allows us to pass control to an assembly language subroutine. We use the procedure JSR to call our GET_TIME code, then dig the (Thunderclock format) time string data out of the Apple's input buffer. This allows us to print the time to the current second instead of limiting it to minutes as "peeking" at the ProDOS TIME and DATE registers would.

This specific PROC also shows a limitation of *PROMAL*—it isn't set up to deal with strings as easily as Applesoft. Since there is no "string" data type, we access a string as a set of contiguous bytes, ending with a terminating value of 0. This limits us to using defined string fields of known maximum size; writing programs that would use dynamic string-handling in *PROMAL* would require defining our own data structures and garbage collection routines for a "string area" of memory. Still, this is not an unheard-of limitation; I've had to do this type of programming in FORTRAN on minicomputers, and the current version of *Kyan Pascal* also does not have a string data type.

PROMAL, though C-like in most of its features, also lacks (as far as I can determine) a data structure to incorporate several variables into a "record" similar to C's STRUCT, though programmers can write routines to build their own associated data structures. *PROMAL*'s library does include several functions that allow common manipulations of string data (concatenation, substring search) so that you don't have to start off writing a string support package.

Basic real-number calculations (add, subtract, divide, multiply) are directly supported in *PROMAL*, but get more difficult with trigonometric and transcendental functions such as LOG and SIN. These functions are provided by a set of real number library routines that are slower than Applesoft's binary floating point routines, but much faster than *Kyan Pascal*'s BCD (binary coded decimal) routines. I came up with the following results on a few simple benchmarks from *Byte* (see the December 1986 issue of **Open-Apple** page 2.88, for information on sources for the benchmarks). All times are in seconds:

	calc	sieve	write	read
PROMAL 2.1	142	131	139	64
Applesoft	97	245	37	36

Benchmarks are only one factor among many in evaluating a language, so take the following comments in that light.

PROMAL's calculations were to 11 digits of precision; error for the iterative calculations was about 0.0000000036 percent. This was for simple "four function" math (addition/subtraction/multiplication/division). For the more rigorous Savage benchmark, which uses trigonometric and exponential functions, *PROMAL* took 706 seconds and returned a relatively high error of about 0.002 percent over 10,000 iterations (versus 472 seconds with an error of 0.0000050 percent for Applesoft). Bruce Carbrey at Systems Management Associates, developers of PROMAL, says he is still looking into improvements for the math library (they are looking into support using the SANE math package provided in the IIgs).

PROMAL's function library also uses iterative methods that can overflow the stack on heavy calculations; I had to split the test formula used in the Savage benchmark into two expressions to avoid a stack problem. *PROMAL*'s manual warned about this possibility.

In simple looping operations (a good portion of any program), *PROMAL* blows Applesoft away; Applesoft takes 143 seconds to execute 100,000 iterations of an empty FOR/NEXT loop; *PROMAL* takes 9 seconds for the same operation.

We can also go to the Gilbreath Sieve of Eratosthenes benchmark reported in *Byte* ("Eratosthenes Revisited", January 1983) to see some differences—Applesoft takes 3,764 seconds versus 163 seconds for *Kyan Pascal* and 154 seconds for *PROMAL*.

What the benchmarks don't show is the flexibility of *PROMAL* in data manipulation. In addition to the math operations and some of the string support we've come to expect from Applesoft, *PROMAL* can do bitwise operations on data. Also, any routine or function we can write can be added to any program with a simple INCLUDE statement; all we have to do is save the debugged subroutine in a text file for use the next time. The manual describes a large number of such routines, which are included with the *PROMAL* package.

The substantial *PROMAL* manual was a bit hard to wade through at first; partly due to my lack of familiarity with the language, and partly due to the fact that the manual interleaves descriptions of both the Apple and Commodore 64 versions of the *PROMAL* system, which occasionally breaks the flow of reading about certain features. The manual contains a tutorial but the reference sections are definitely intended to be read by someone with programming experience. Still, there is an index and I found all the information to be there for solving some complicated problems in porting the *Byte* benchmarks to *PROMAL*. The reference sections and appendices of the manual are superb in the fact that they give concise, technically accurate descriptions of the *PROMAL* language and do not assume the reader is simple-minded. Some of the information is hard to read the first time, but it is the completeness of the explanations of some features of the language (such as program chaining and overlays) that makes it so. If you decide to learn about such advanced features, all the information is there waiting for you rather than simple words pointing you to another manual.

If the manual is serviceable, the amount of source code that comes with the *PROMAL* system is amazing. Included is source code for a simple terminal driver (for the Apple Super Serial card and compatibles), the floating point routines, access to the ProDOS MLI, and a lot of samples. The $100 developer's system allows you to distribute compiled programs written in *PROMAL*, including the use of the source code routines and a provided stand-alone runtime package for self-starting disks. A $50 end-user system is also available. Both *PROMAL* packages include a very usable full-screen editor, a linker for combining compiled routines, a command-line executive with "memory management" and I/O redirection, and an extensive library of additional functions. A high-res graphics support package is available at extra cost.

Phone support during several conversations with SMA was excellent. In addition, there is a disk library of *PROMAL* support routines and programs available to *PROMAL* users.

For languages other than Basic, it looks to me like today's front runners under ProDOS 8 are *Kyan Pascal* and *PROMAL*. Both have speed advantages and disadvantages compared to Applesoft that depend on the operations being performed. Pascal has the advantage of wider portability; *PROMAL* versions exist for the Apple IIe and IIc (IIgs also, with exception of the terminal driver routines, which need to be updated for the IIgs ports), Commodore 64, and MS-DOS 100 per cent IBM-compatibles. That covers a large portion of the micro market, but Pascal also runs on most computer systems with some minor changes. The Apple version of *PROMAL* also expects a IIe or IIc (*Kyan Pascal* supports any 64K Apple II). The primary limitation for both languages is that neither has support of strings that approaches the simplicity of Applesoft string variables.

PROMAL 2.1, by System Management Associates (3325 Executive Drive, Raleigh, NC 27609; 919-878-3600) development system $99.95; end-user system $49.95; graphics toolbox $29.95.

Ask (or tell) Uncle DOS

On the other hand

Hey! Every time I write to **Open-Apple** with a suggestion, the next issue always contains a letter from someone saying, "you can't do that." Come on guys, if Steve Wozniak had that attitude, you'd all be writing Cobol programs for IBM.

I'm particularly disappointed that Peter Baum called my "frigid" reboot idea "extremely dangerous" ("Frigid flaw," page 3.80). Baum's objection, like earlier objections, is that it might not work with future hardware revisions. He's concerned that the power-up byte location may change.

First of all, Apple's documentation states that this byte is reserved, and won't change. Of course, that doesn't mean much. If it does change, so what? We'll just change a byte in the frigid reboot routine. Finally, the frigid reboot routine is designed to keep you from having to turn off your power switch. God forbid this routine might crash! You might have to turn off the power switch.

Unfortunately, the "you can't do that" attitude seems to be taking over at Apple. The most exciting hardware product at AppleFest was the Zip Chip, a plug-in replacement for the 65C02 that triples the CPU speed of the Apple IIc, IIe, and II-Plus. The company says they'll have a IIgs version out next year to replace the 65816, but two of the top Apple IIgs engineers I talked to said, "you can't do that." My money is on the Zipchippers.

I talked to a number of ProDOS 16 developers at AppleFest, who complained of performance problems caused by strictly following Apple's tool calling protocol, particularly for keyboard input and screen output. I say that if you sacrifice current program performance for compatibility with future hardware, you'll be out of business before that future hardware appears.

For the people who disagree with me and want to write a rebuttal letter to **Open-Apple**, you can't do that!

Bill Basham
Diversified Software Research
Farmington, Mich.

Ah, come on Bill, sure they can. One of the best parts of AppleFest for some of us was eavesdropping on the parleys you were having with people who designed various parts of the IIgs.

*Compatibility is very important to the long-term mental health of the Apple II user community. But you and I, as authors of **Diversi-DOS** and **ProntoDOS**, know quite well that **users will pay for speed.** We got our start making Apples faster. It can be done. If Apple had done it our way to begin with, where would we be now?*

P.S. I was impressed by the Zip Chip too, but we've decided to reserve judgment until we actually get our hands on one. They're still not shipping.

Some feedback on reality

While I am an avid reader of **Open-Apple**, I am seldom as galvanized by an article as I was by your lead piece for the November 1987 issue, "Reality and Apple's Vision." You have articulated and made explicit in this essay the uneasy feeling that has been slowly and quietly creeping up on many of us in the Apple II world. Thank you for sounding the alarm.

As one who bought a IIgs early on, I am most distressed by the absence of any indication that a 16-bit AppleWorks will emerge from Apple or from anyone else. I'm *not* talking about a ported-over version of *Microsoft Works*. I'm talking about an *extension* of the current AppleWorks, filecard interface and all, to the 16-bit IIgs. The potential for building on this solid foundation of success is enormous.

If Apple absolutely, positively must have a Mac-like appearance then let's use the mouse with MouseText *in conjunction with cursor control* as many of Pinpoint Publishing's products do. We'll see what gets the best reviews. The use of MouseText is infinitely faster than the Mac-clone stuff I've seen so far.

When it takes an eternity for a 16-bit program launcher to load and position itself for duty in comparison to 8-bit desktop managers like Pinpoint's *Run-Run* or Glen Bredon's *ProSEL*, then someone has really goofed. Consumers are not that stupid. If you don't believe that, just try selling full-size cars with lawn mower engines in them.

I have a IIgs. I have several of the latest and most popular 16-bit software packages. When I want to get something done, I use my 8-bit software, most notably AppleWorks. Why? Because I don't want to wait longer to use a computer than it takes to assemble pencil and paper. Because I don't like to be patronized by a weak and slow version of the Macintosh user interface. If I had wanted or needed a Mac, I'd have bought a Mac.

You suggest that Apple execs are pursuing a rational, though misguided, course of action in order to mollify disaffected third-party software developers. Specifically, you suggest that Apple is trying not to compete with software developers.

If that's their aim then how in the world did *HyperCard* get out? *HyperCard* on the Mac will, I predict, devastate the Mac software industry, especially the big-boys. With *HyperCard*, anyone can be a sophisticated software developer. My belief is that they don't know what they're doing.

We probably can't cure Apple of its death-wish, that's too deeply ingrained. What we can do is make the choice so plain that even a demented fool would recognize the hand that feeds him; so plain that basic survival instincts override the afore-mentioned death-wish.

Here's what I propose. We begin a realistic public discussion of what a 16-bit AppleWorks could and should do. This will accomplish at least two things. First, it will provide Apple with a free market-analysis, one they wouldn't commission on their own because of the fear that they might find out that their "Apple II as mini-Mac" theory is bankrupt. Second, it will place the initiative for what such a software item might be in the hands of those who will ultimately use it. Power to the people.

Let's hear what the Apple II folks out there in keyboard-land really think.

Frank Lowney
Milledgeville, Ga.

Your editorial "Reality and Apple's Vision" was *right on the money*. I believe your statement that we are "on the edge of a disaster" was not hyperbole. It is an accurate statement when applied to the Apple II's true situation, and Apple should notice.

It probably won't notice, however. Why worry, when IIs are still selling strong? Why find-tune marketing and software evangelism, when you've made a massive investment in pigeonholing the Apple II as an "entertainment and education" computer rather than recognizing how IIs are actually used by many people? The roaring success of AppleWorks provides excellent feedback on how IIs are applied by adults in the real world, but as you correctly noted, Apple itself has almost gone out of its way to ignore this feedback, other than to deposit the proceeds.

I am 31 years old, an account executive with an advanced degree. I do not use a Macintosh; I use an Apple IIc because I like it and the publications and the people. I am frustrated, however, by what I perceive to be an undersupply of good productivity software for my computer. The advent of the IIgs had me excited until I read from Jean-Louis Gassee that "in no way, shape or form will we be in the office with this computer." (*Editor's note: I am not familiar with this quote, and it doesn't ring true with what I* ***have*** *heard Gassee say. I will attempt to find out more for a future issue—sorry for the interruption.*)

What kind of message does that send developers? John Sculley was on TV last night, singing the praises of the "Macintosh in the office, the Apple II in education." What about the Apple II in the office, Mr. Sculley, or in the home office?

I use *Apple Writer, Point-to-Point*, and software from Glen Bredon and Bill Basham. These are better than almost every MS-DOS application I have used, and I have used a lot. Problem is, software for adults as useful as these programs is rare. Apple perpetuates this condition through its "positioning," which discounts the possibility of non-school markets for the II. And through its training of computer store salesmen, who can't help you if it doesn't go on or in a Macintosh. (Hint: Bypass the suits and talk to the technicians.)

At the same time, Apple will not develop any more good, crash-proof applications for the II because it is afraid of criticism from developers. I really do feel that developers have had their chance. Most of them are too busy with MS-DOS to create new II stuff that doesn't draw a greeting card or animate a bear. Nobody criticized GRiD when it came out with integrated packages for its computers. Borland and Ashton-Tate don't bellyache when Atari releases software, or rake IBM over the coals for manufacturing software. Why should Apple be so concerned about static from outside developers when those developers have had years to make more productivity software for the II, but didn't do much, except for AppleWorks add-ons and programs in specific vertical markets. When, with a good product, it's easier to be a big fish in the II pool than in the MS-DOS ocean?

Clone prices are, of course, plummeting and MS-DOS software is getting better and easier to use—witness *Microsoft Works* for the PC. I'm thrilled that Apple is using supercomputers and geniuses to develop better system software. But what difference does it make when they promote the II essentially as a gadget to display flash cards? Through too-narrowly-focused advertising, ambivalent developer programs, and a lack of desire to support the wider world of Woz' invention, Apple will widen the gap between what people need the computer to do and what Apple wants them to do with the computer. That puts Apple IIs in closets. It also puts clone makers, who haven't a fraction of the genius and resources Apple has at its disposal, in the winners circle.

Benn Kobb
Washington, D.C.

I like the concept of AppleWorks as an operating system. Although I'm a true Macintosh zealot today, my roots are in the Apple II. Whenever I use an Apple II today, my work generally involves AppleWorks' power and flexibility. In fact, we are about to release an interface to AppleWorks for our library circulation system, *Circulation Plus*. It will provide a path for our users to import and export data in both directions, giving them the power of AppleWorks to massage their data.

Don Rose
Follett Software Company
Crystal Lake, Ill.

I just finished reading your opening article in the November issue and feel that it requires a response. If not for myself then in defense of Apple. I have been using Apple computers since early 1982 and consider myself an intermediate to expert user. I do not own, nor have I used AppleWorks (except once when I got so confused that I gave up).

The program uses commands that, though many people are familiar with them and add-on programs use them, are completely different from any other program I know of. Also the program (and it is a program and not an operating system) uses a file structure that is different from all other programs. I know you can use text and DIF files but you yourself have said that it's a pain. No DOS 3.3 program and very few ProDOS (note: these are operating systems) programs use the same file structure. All BBSs and information services use standard text files, as do many good ProDOS programs.

Before you think that I'm completely against AppleWorks let me say that the program is of good quality and very easy to learn if you don't have to unlearn another program in the mean time. It's proven itself in sales and testing. That's an excellent track record for such a complex program.

As for Apple not having ads for AppleWorks, I don't think they need them. Think about it. If you had a product that sold as well as AppleWorks and also had some other products that weren't selling, which ones would you use your advertising budget for?

Martin Wallgren
Prophetstown, Ill.

I would like to disagree with you on a couple of points regarding "Reality and Apple's Vision." Your suggestion of acknowledging Robert Lissner's contribution to the Apple II is a good one. However, including AppleWorks as system software has some serious flaws.

First some background. I was an avid Apple II user for several years until just this last summer when I acquired a Mac and sold my Apple II. Until about 18 months ago, Apple bundled *MacWrite* and *MacPaint* with every new Mac sold. For the first three years of the Mac's life, there was only one word processor the Mac, *MacWrite*. Because everyone received *MacWrite* free, there was little room in the market for other word processors. The market stagnated for three long years in word processors and paint programs. Apple finally decided to unbundle *MacWrite* and *MacPaint* for the Mac, and the market is now booming with word processor and paint programs. And during the three years that *MacWrite* and *MacPaint* came in the box with the computer, did Apple grow and improve the programs? No! They merely updated each program enough to keep it up-to-date with the evolving Macintosh Operating System.

So you see, Apple really is a computer hardware company. Not only do they ignore *AppleWorks*, the program that has sold so many Apple IIs, but also *MacWrite* and *MacPaint*. Including AppleWorks as system software for the Apple II will not solve the stagnation of AppleWorks, it will only stagnate the competition also.

I also have problems with AppleWorks being the standard platform for ProDOS. Before I continue, I must admit I like the idea of an improved AppleWorks that has many hooks developers could use to attach their software. This would alleviate the problem of one add-on patch to AppleWorks not getting along with other add-ons. But ProDOS should be kept separate from AppleWorks. One shouldn't have to run AppleWorks just to delete or copy a file. The beauty of ProDOS is that it is *just* an operating system. The SYSTEM file defines the actual user interface. This is one of the most common misunderstandings of ProDOS. I have seen many articles where a command like "BSAVE LOADER.SYSTEM, TSYS, L16284, A8192" is attributed to ProDOS. This is absolutely incorrect! That is a Basic.system command. ProDOS only understands Machine Language Interface calls. The beauty of ProDOS is that it leaves the user interface to the SYSTEM file, and SYSTEM files can be developed to make ProDOS look like any operating system (Kyan Pascal's KIX.SYSTEM, for example, has a Unix look).

So for some constructive ideas, I agree that Apple has to create a vision with the Apple II as they have with the Macintosh. This vision might settle on a standard interface for the Apple II under ProDOS; I believe that Basic.system is the de facto standard right now. Apple should decide on supporting the Apple II and stop worrying that it will undercut their Mac sales. Whether Apple sells an Apple II or a Mac, they still have a customer.

Craig Miller
Honolulu, Hawaii

I completely agree with your point that ProDOS and AppleWorks should be separate. As you point out, significantly different user interfaces can be embedded in SYSTEM programs; that is one of the beauties of the Apple II. The point at which we disagree is that you think Basic.system is still the de facto standard interface on the Apple II and I think Aplworks.system became the de facto standard months ago.

As the de facto standard, I see no reason a 16-bit version of AppleWorks needs to change the user interface at all. For once why couldn't we have a 16-bit version of a program that actually recalculates spreadsheets and sorts databases ***faster*** *than the 8-bit version, instead of wasting all the extra power on bells and whistles. Why not a 16-bit version that uses today's larger memories for data space rather than for program space? The major enhancements I look for in a 16-bit AppleWorks are speed, larger data files, system hooks, developer's documentation, and memory management that would allow third-party developers to do the real enhancing.*

The ***Hypercard*** *precedent gives Apple the liberty to begin putting AppleWorks in the box with all Apple IIs. This can hardly stifle a software market that has already been choked to death by an AppleWorks that has a suggested retail price of $250. It would, in fact, give software developers a new base (in Apple's words, a "platform") to develop from.*

If we can't convince Apple to put AppleWorks in the box with the Apple II within the next couple of months, the program that defines our machines will fall into the hands of Claris. Perhaps that's no worse than having it at Apple, but still, the idea makes me sweat.

All ProDOS Sider update

Please warn your readers that the popular all-ProDOS ROM developed by Steve Park, which has been used by many people on all models of the 10 meg Sider and the 20 meg Sider II, is not compatible with our new 20 meg model, called the Sider D2. We are using a different drive and controller in the 20 now and we are hearing that not only does Park's ROM not work, but it will screw up the format on the drive to the point that our own software and ROM will no longer be able to install the drive.

We have been shipping the D2 for about 6 weeks. You can easily identify it because it has a black label on the front that simply says "The Sider." The old 10s and 20s had blue labels that said "The Sider" and "The Sider II" respectively. The 10 meg siders are no longer being built. The suggested retail price of the Sider D2 was just lowered to $595. Despite the use of a third-party product with the Sider, we have not voided anyone's warranty and have replaced all D2's that have run into this problem. I am told that a factory format can get the drive back up and running fine.

Lance Jacobs, Technical Support
First Class Peripherals

Those of you with older 10 and 20 meg Siders whose ears perked up at the words "all-ProDOS ROM" can contact Steve Park at Advanced Tech Services, PO Box 920413, Norcross, GA 30092 404-441-3322. The ROM sells for $49.95.

Zero-page: No Vacancy

I have seen lists of zero-page locations compiled by Beagle Bros and others, but none of them list *all* the locations. What about the rest? Are they simply not used by Applesoft or DOS?

Nick Doulas
Chicago, Ill.

On page 142 of the *ProDOS Technical Reference Manual* is a list of zero-page locations, who uses them, and which are free. This map lists location $D6 as unused. In writing an ampersand routine for use by an Applesoft program, I discovered, after much frustration over an intermittent and elusive bug, that $D6 is used. Just a caveat for your readers who may be using the ProDOS manual as an exclusive reference.

Chuck Bilow
Oregon, Wisc.

In general, assembly language programmers should assume that ***every*** *byte on zero-page is used. In fact, Applesoft, DOS, and the Monitor do use almost every last byte; assume that other ampersand routines use the few bytes that are left. If you need some zero-page bytes for indirect addressing, save what's already in a couple of bytes, use them, then replace the previous contents. Don't attempt to use zero-page as a storage area—build your own data storage area inside your program.*

The book ***What's Where in the Apple****, by William Luebbert, is probably the best single source on zero-page usage. Two of the major companies that publish Apple II assemblers, Roger Wagner Software and S-C Software, have programs available that disassemble Applesoft and give you complete details on its zero-page usage.*

AppleWorks on the II-Plus

My soccer club already had a II-Plus but wanted to use AppleWorks. After a bit of checking around to compare prices and the market for II-Pluses without drives or monitor, just the box, we decided to upgrade the II-Plus instead of buying a IIe. Not necessarily the best way, but the cheapest.

Before starting, we determined that the "minimum" requirement was a 64K II-Plus with the famous "one-wire" shift key modification. We had both, so we were on our way.

We had to have an 80-column card to run AppleWorks. I found a "clone" card for a very reasonable price from WG Technologies. Look in the back pages of *Computer Shopper* magazine for their address and a current price quote.

It is possible to use an Apple-style memory card for desktop expansion, *if* you have AppleWorks 1.3 (we did). We bought Applied Engineering's RamFactor. The RamFactor comes with AE's patch software that makes AppleWorks run on the II-Plus. The version of the patch we received was 1.2.1—I know from another project that this is the same software that AE sells with their ViewMaster 80-column card.

After installing the 80-column card and the RamFactor the rest was easy. (You may need to adjust your monitor after installing the 80-column card.) Make a copy of AppleWorks. Again, it must be version 1.2 or 1.3, not 2.0. The AE program is menu driven and I am not going to describe how to use it. Just follow the on-screen instructions.

If you have AppleWorks 1.2, you will not get much. Even after patching, it will only use 64K for programs and data. This means you have just a 10K desktop and the program must go back to the disk to "overlay" program segments every time you switch operations. Get 1.3 if you can.

With AppleWorks 1.3 and the RamFactor, data space is limited only by the amount of memory installed on the Ramfactor. The *real* plus, however, is that 1.3 will load itself into the memory card for *fast* operation. The only program segment that must be loaded from the disk is the one for printing. A small price to pay for the speed of using the memory card for all the rest of the AppleWorks' program functions.

Since the II-Plus does not have an open-apple key, the patched program has you press the ESC key once, and then the desired key. Such as ESC-P to print or ESC-S to save. You press ESC ESC to get the actual ESCape key.

Due to the lack of a solid-apple key, and the MMU chip of the IIe, add-on programs such as AppleWorks or PinPoint will not work on a II-Plus system.

Tom Smith
Fort Vancouver, WA

I was unable to find an ad from WG Technologies in the latest issue of ***Computer Shopper.*** *However, we recently received a flyer from one of our subscribers who works for a company called Nexo Distribution (914 E 8th St, #109, National City, CA 92050 619-474-3328). They sell all the pieces needed to get AppleWorks to run on a 64K Apple II or II-Plus for under $100:*

```
PlusWorks II software, Norwich Data   $39.00
Videx-compatible 80-column board       49.00
Shift-Key Modification kit              6.95
```

PlusWorks has several advantages over Applied Engineering's software—it's compatible with all Appleworks versions from 1.1 through 2.0 and it's compatible with all memory boards—even older 128K Saturn- and Legend-type boards.

If you are starting with a 48K Apple II, Nexo has a 16K card for $35, but the money might be better spent on an accelerator card for slot 0—these all add the missing 16K as well as speed things up (see "RAM found in accelerators," in our December 1986 issue, page 2.88).

Multiplan update

In "Spreadsheet $tring variables," a reader asks for a spreadsheet that lets you use the IF statement with text. He says "neither AppleWorks nor *Multiplan* allows text as a clause or argument in formulas." AppleWorks, true, but my version 1.07 of *Multiplan* allows more than adequate string manipulation. For example, IF(R1C1>R1C2, "MORE THAN", "LESS THAN") yields the text "MORE THAN" if R1C1 is greater than R1C2 and the text "LESS THAN" in all other cases.

Multiplan 1.07 also allows the concatenation of strings using an ampersand ("Open-" & "Apple" yields "Open-Apple"); the separation of strings using the MID("TEXT",start,num) formula; the translation of strings to values using the VALUE("text") formula; and the translation of values to strings using the FIXED-(value,num) formula. In short, there isn't much which *Multiplan* can't do with strings (or any type of data).

Interestingly enough, though, I'm in the process of changing all of my *Multiplan* spreadsheets to Appleworks for two reasons: 1. the ease with which I can move from application to application and 2. the speed with which AppleWorks calculates. True, *Multiplan* overpowers AppleWorks in some areas, but since AppleWorks is probably doing less (or are there other reasons for *Multiplan*'s sluggishness?), I've found speed improvements of up to 40 seconds on similar 12 column by 60 row spreadsheets.

On another spreadsheet topic: on page 3.39 of your June 1987 issue, you stated: "A simple way to get blanks instead of zeroes in calculated cells is to set up an @IF statement that displays 'NA' if a cell's value is zero. Then start up your favorite disk zap program, find the NA...and change it to blanks." Thank you, it works...but not enough.

If the calculated "double-blank" cell is the last calculation in the chain, fine. But, for example, if you add a column of numbers that includes even one double-blank cell, the answer will also be a double-blank. The whole idea was to clean up the looks of a spreadsheet with lots of zero cells, but your solution cleans up all the result cells, too. So, does anyone out there have a patch to allow something like @IF(A1<>0, A1*B1, " ")? *Multiplan* allows this and it's one of the features I miss with AppleWorks.

Robert Cerchio
Carbondale, Ill.

We thought Multiplan for the Apple II was dead, but a call to MicroSoft's sales department at 800-426-9400 (206-882-8088) revealed they still sell it. Version 1.07 is the latest. Updates from older versions *cost $25. Suggested retail on a new copy is $95. It's still "Apple" DOS 3.3 based and still copy-protected. Microsoft hasn't tested it on the IIgs (and doesn't intend to), but says it's compatible with all earlier Apples.*

You've identified a significant limitation to the NA-method for displaying cells with zeros as blanks. Does anyone have a better idea for this one?

AppleWorks IIgs defeater

Maybe I am too old to switch (70 years) but I still want to use AppleWorks version 1.3 on my IIgs. But even though I have a RamFactor card in slot 7 and a 1.5 meg GSRAM card, I can only get a 55K desktop. Surely, someone can come up with a patch to make AppleWorks recognize the RamFactor.

Elmer Meissner
Bedford, Ind.

Incredible as it may seem, AppleWorks 1.3, which was released months before the IIgs, includes a test to see if it is running on a IIgs. If it is, it refuses to expand into an Apple-standard memory card, such as RamFactor. AppleWorks 2.0 also refuses to use this type of memory card when run on a IIgs.

Dennis contacted our AppleWorks guru Alan Bird, who had a patch handy. He came up with it while debugging the Time Out manager so he could fool AppleWorks 2.0 into thinking it was running on a IIe when it was actually running on his IIgs.

Alan's one-byte patch simply changes a JSR $FE1F instruction (20 1F FE) to BIT $FE1F (2C 1F FE). As a JSR, this instruction jumps to the IIgs identification routine. As BIT, it makes the IIgs appear to be a IIe. To make the patch, use a disk zap utility to search through APLWORKS.SYSTEM for the byte to change, or get into Applesoft, insert an unmodified **copy** *of your AppleWorks disk in the drive, and enter:*

```
POKE 768,44
BSAVE APLWORKS.SYSTEM,TSYS,A768,L1,B13223   (V 1.3)
BSAVE APLWORKS.SYSTEM,TSYS,A768,L1,B13605   (V 2.0)
```

Because of the patch technique being used, there is no need to BLOAD the file first (see "Patch instructions patchy," page 3.79, for more).

AppleWorks 2.0gs bugs

After running AppleWorks 2.0 on your IIgs, try running a program that uses the high-resolution graphics screen. You'll just see garbage. AppleWorks 2.0 turns off IIgs "shadowing" for everything but the text screen and neglects to turn it back on. The following short program will fix this:

```
10 REM fix IIgs shadow register
20 FOR I=0 to 10 : READ X : POKE 768+I, X : NEXT
30 DATA 173, 53, 192, 41, 160, 9, 8, 141, 53, 192, 96
40 CALL 768
```

Jim Luther
Kansas City, Mo.

For the full scoop on what "shadowing" is, see "IIgs alternate display mode" in our December 1986 issue, page 2.86.

A similar bug in AppleWorks 2.0 is that it gets an ID from the IIgs ID manager, but doesn't delete it before quitting. If you run AppleWorks 2.0 on a IIgs 256 times without turning the machine off (not impossible if you leave your machine on 24 hours a day), the ID manager will run out of IDs. We haven't had time to actually test to see what kind of disaster happens then, however.

Another fix for both of these problems is to simply make AppleWorks think it's running on a IIe, as demonstrated in the previous letter.

IIe aux-slot advantages

Potential users of the *Beagle Compiler* should note that compiled programs may use either aux-slot or standard-slot RAM cards to hold variables. However, it is only the aux-slot-type card that can be partitioned to hold both variables and a RAMdisk. I have been able to have both AppleWorks and my own compiled programs in a RAMdisk while using the rest of the card for variable storage (my programs) or desktop (AppleWorks).

Anyone developing or using programs that read AppleWorks files should find this type of arrangement very convenient. Using *ProSel* it is possible to jump from AppleWorks to your program and back in less than 10 seconds. This feature, in my opinion, tips the balance in favor of auxiliary cards for the time being.

Paul McMullin
Campinas, S.P. BRAZIL

Gutenberg lives

After reading September's "Reviewer's Corner" (pages 3.59-3.61), one might think that *Printrix* is the first text processor to deliver the type of performance you describe. (I know you didn't say that.) However, the majority of the features you describe have been available to 48K Apple owners since 1981, when the first version of *Gutenberg* was introduced.

The current ProDOS-based version 3.0 allows four fonts to be in memory at once and allows still more fonts to be used on a given job. These fonts may be downloaded fonts or special graphics fonts that allow the use of huge characters. They may be stored on a RAMdisk for quick access. Vertical spacing, letter-spacing, and minimum and maximum word-spacing can be changed in tiny increments, as with *Printrix*, at any place in the text.

Special formats, such as fractions, definite integrals, etc., can be automated so that inputting such things is a simple matter. The program supports multi-column setups, automatic formatting of simple and complex ruled and unruled tables, customizing of automatically printed headers and footers (changeable at any point in the text), footnotes and endnotes, mail merge, several types of automatic numbering for enumerated paragraphs, imbedding of graphics material, and much more.

Its appeal to foreign language specialists is obvious, since it comes ready to print in Greek, Hebrew (yes, left to right), Arabic, Russian, Ukranian, Syriac, and Cree (an Indian tribe of Western Canada). Just in case you want to change your dip switch settings to use one of the European fonts supplied with the ImageWriter, Gutenberg supplies you with the matching screen fonts.

It isn't perfect though. Not nearly enough fonts are supplied to print a newsletter such as yours, although there is a hint in the manual that more are on the way. It is not cheap, but technical support is fast and excellent. It supports only the Apple DMP, ImageWriter, and ImageWriter II. It formats at the printer, not on the screen, but on the other hand offers unlimited formatting capabilities and superb precision.

For any Apple owner with a special job (eg., linguist, scientist, mathematician) it may be the answer to a prayer. A few months ago a certain magazine carried a rather thick center-fold ad for a new IBM-based technical word processor from a very well-known company. It asked the question, "Was your word processor designed to do this?" As a *Gutenberg* user, I could answer "yes" and be about 95 per cent truthful.

Olof Monson
Kingston, Ontario

***Gutenberg Sr. 3.0** requires 128K and an 800K disk device. It sells for US$360 (C$460) from Gutenberg Software, 47 Lewiston Road, Scarborough, Ontario, CANADA M1P 1X8, 416-757-3320. The company also offers a US$91 (C$118) version called **Gutenberg Jr.** It comes with a manual on disk and specific defined printing formats. The junior version is printer-specific, however, it is available for a wider selection of printers than the senior version. For more information, write or call Gutenberg Software and ask for a copy of their 16-page sample brochure (printed with **Gutenberg**, of course).*

Help wanted

Is there anyone you can pay to create an AppleWorks custom printer definition for a non-Apple printer?

Todd Shelton
Chico, Calif.

Or how about a disk full of SEG.PR files for non-Apple printers? We don't know of any, yet.

Multiple custom printers

In its April 1987 issue, the National AppleWorks Users Group's *AppleWorks Forum* published the addresses of the pre-defined printer codes held in SEG.PR. The article is by Garth Shultz, who attributes the information to an article by David Walker that can be found in the DL-4 library on CompuServe's MAUG.

With this information it is a relatively simple task to replace a set pre-defined codes (for example the Imagewriter codes) with a set of custom codes you've entered using the AppleWorks Add a Printer menu. You can also change the printer names that appear in the in the Add a Printer menu by editing SEG.M1 (at $0E9 of block $0117 on the AppleWorks program disk).

I've created a single SEG.PR file that includes an Okidata driver with draft mode codes, an Okidata driver with near letter quality codes, and an Okidata driver with boldface begin defined as ESCAPE. Works fine.

Bruce Ristow
Rochester, N.Y.

Shultz's idea is to define a custom printer from within AppleWorks, BSAVE that section of the SEG.PR file, then overlay one of the pre-defined printer's codes with it. Then you go back to AppleWorks and add that pre-defined printer. Now, when you use that printer, you'll get your custom printer's codes. This allows you to put three custom printers in SEG.PR rather than just one.

The only difficulty with this technique is that the length of a set of SEG.PR printer codes varies. The length depends on exactly what codes a printer uses. For example, a printer that uses a code such as "ESCAPE R ESCAPE 44 LF" for super-script begin needs more space than a printer that uses a code such as "ESCAPE S" for this.

SEG.PR has a 500-byte space for the custom printer. If you BLOAD SEG.PR, A$2000, this space is at $2B42. You can overlay a complete 500-byte section of codes into SEG.PR at the start of the Scribe section ($24BA) and at the start of the Epson FX section ($2845) without overwriting either the Imagew-

*riter/Apple Dot Matrix printer codes or the custom printer codes (you **will** overwrite most if not all of the other pre-defined printer codes however, so don't attempt to use them). Proceed like this:*

```
enter AppleWorks, define custom printer #1
exit AppleWorks, enter Applesoft

BLOAD SEG.PR, TSYS, A$2000
BSAVE CUSTOM.1, A11074, L500

enter AppleWorks, define custom printer #2
exit AppleWorks, enter Applesoft

BLOAD SEG.PR, TSYS, A$2000
BSAVE CUSTOM.2, A11074, L500
BLOAD CUSTOM.2, A10309
BLOAD CUSTOM.1, A9402
BSAVE SEG.PR, TSYS, A$2000

enter AppleWorks, remove all printers
add a Scribe; this is custom.1
add an Epson FX; this is custom.2
add a custom printer; this would be custom.3
```

The proportional problem

Why isn't it possible to set up a custom printer for AppleWorks and the ImageWriter II that will print proportionally?

Michael Leddy
Charleston, Ill.

Proportional fonts are hard on word processors. The concept of "characters-per-inch" simply doesn't apply to proportional fonts—the characters, by definition, are all different widths. Consequently, to get printed lines to be approximately the same width, the word processor has to know the exact width of each character and do line calculations based on those widths. Many more "i"s fit in a line than "M"s. To do full justification, which most people with access to a proportional font immediately want to do, the word processor also needs to calculate how much space should go between each word. Then it needs to know the commands for printing blank spaces of various widths.

AppleWorks' author correctly, in my estimation, judged that entering all this information into custom printer definitions would be beyond the ability of most users, and so he didn't include proportional capability for custom printer setups. He did, however, go to the trouble to support proportional fonts on the Apple DMP/ImageWriter, the Apple Daisy Wheel, and the Epson FX.

*It sounds like what you want to do is add some custom features to the Imagewriter without losing those proportional fonts. This can be done by manually editing the Imagewriter definition in the SEG.PR file. See "Zapping Imagewriter codes" in our February 1987 issue, page 3.4. The AppleWorks BBS (see "AppleWorks + **Multiscribe**" in August 1987, page 3.54-55) includes an assembly source file for SEG.PR that you can follow like a roadmap. Using this information, it's possible for a knowledgeable individual with lots of spare time to create a "custom" driver for any printer. But be aware that the number of capabilities defined in SEG.PR is limited; to add a feature will mean losing existing capabilities.*

*If this sounds too difficult, **Open-Apple** subscriber Eugene Whitehouse has developed a program called **ExtraWorks Printer Utility** that rewrites AppleWorks' Imagewriter or Epson printer setups so that you can get things like color printing, mousetext, half-height sub- and super-scripts, and slashed zeros without giving up proportional type ($20, 25 Kensington Ave., #503, Jersey City, NJ 07304).*

Database crash lead

I'm pretty sure I've found a weird problem with AppleWorks that occurs to people who upgrade from version 1.3 to 2.0. I've seen it about a dozen times now, and have examples. All of a sudden, AppleWorks 2.0 will not load old database files created with 1.3 anymore. In every case, the byte in the header that shows the number of report formats defined was incorrect. I haven't found the code that causes this yet, and it obviously doesn't happen a lot, or we'd have heard of it already, but I've seen it enough to be convinced that somewhere there is a problem. If you hear from anyone who has this problem, have them send me a copy of the disk and I'll at least try to fix it; it may help me find the cause.

Eugene Whitehouse
25 Kensington Avenue, #503
Jersey City, NJ 07304

AppleWorks page nos. (cont)

I use the AppleWorks word processor to create long documents, some over 300-400 pages. According to the manual, I should be able to get up to 511 consecutively numbered pages. But with both AppleWorks 1.3 and 2.0 I get interrupted pagination following page 256. Making corrections on the misnumbered pages is a chore, so if there's any way of getting AppleWorks to deliver its 511 consecutive numbers, I'd be delighted to know it.

David Alman
Highland Park, N.J.

We've been following this bug since April 1986 (page 2.23) and once had a report the bug would be fixed in AppleWorks 2.0 (page 2.31). Apparently it is more complicated than at first thought—tests I've run using a page length of half an inch, a header with the page number, and one line of text, print page numbers correctly well into the 300s. (This kind of testing takes a long time and uses lots of paper—printing to a formatted text file (August 1985, page 2.60) helps some.)

*At any rate, you might want to look at the program **WriteWorks** from W*A*R (Working Apples Relentlessly) Software ($29.95 plus $2 shipping; 4974 N Fresno St, #282, Fresno, CA 93726). **WriteWorks** is a "post-processor" for AppleWorks word processor files. To use it you leave AppleWorks, run **WriteWorks**, and tell it which file to print. It picks up internal AppleWorks formatting codes, reacts to additional "double-dot" commands (lines that begin with two periods) that you embed in your file to send special codes to your printer, automatically formats footnotes, and correctly prints page numbers.*

CHAIN bugs and overlays

I am developing a ProDOS-based Applesoft program that would benefit greatly from using binary program overlays as described in your April 1986 issue, page 2.20 and 2.21. I want to use three program segments that share lengthy menu and character input subroutines.

So, following your directions, I slavishly prepared a file containing those shared subroutines and named it "OVERLAY." After many unsuccessful attempts at doing a run and much tinkering and the fourth re-read of the article I came to the conclusion that there was a problem with LOMEM. And, lo and behold, on page 2.20 I find "Neither the DOS 3.3 nor BASIC.SYSTEM versions of CHAIN pay attention to whether you have reset LOMEM". And this is a fact.

But the instructions you give for a STARTUP program at the bottom of page 2.21 say to move LOMEM beyond the end of your longest overlay and then CHAIN to your main program. Have I missed something?

Theodore F. Smolen
Danvers, Mass.

What we're dealing with here is your bugs, my bugs, and Apple's bugs. Your bug is the simplest to explain, so let's start with it.

Your shared menu and character-input routines should be in your main program and your three program segments in three separate overlays. Instead, you have designed your program with a single, permanent "overlay" and three "programs" that you want to CHAIN in under the overlay. This won't work. The overlay technique is intended to replace CHAIN. It's based on the idea of a main program holding all shared routines and of overlays holding independent program segments. You have to think of CHAIN and overlays as two different techniques. The difference between them is that CHAIN changes the whole program in memory, while the overlay technique changes only part of the program in memory.

*My bugs come in two parts. You haven't missed anything, my comments at the bottom of page 2.21 are just stupid. Take a pencil and change the next to last paragraph so that it says, "If you attempt these tricks, your main program should begin with the following steps." Then cross out the last two steps, which relate to the CHAIN command. We **could** have a separate STARTUP program, like the one described, that simply copies all program segments from floppy disk to RAMdisk. But it should end by RUNning the main program. And the main program itself, not STARTUP, should change LOMEM and should RESTORE its own variables.*

The second bug I made was higher on page 2.21, where I showed how to jump from the main program to the overlay. The way I did it works only when the line numbers used are "right." For a complete description of this bug and how to get the overlay technique to work with any line numbers, see the letter "Problems with splits," from Paul Nix, in our September 1986 issue, page 2.64.

Apple's bugs have to do with Basic.system's CHAIN, STORE, and FRE commands. These three commands share a set of routines that move Applesoft variables around in memory. Unfortunately, the routines need to work slightly differently for CHAIN and STORE than for FRE and for automatic garbage collection, but they don't. As installed at the factory, Basic.system works fine for FRE and garbage collection, but messes up CHAIN and STORE when the Applesoft variable tables happen to be an exact multiple of 256 bytes long.

*In the same issue of **Open-Apple**, on page 2.20, I mentioned a couple of articles in **Call-A.P.P.L.E.** that first demonstrated, then fixed, the CHAIN/STORE bug. What I didn't know at the time was that the fix has the side effect of messing up FRE and garbage collection. Complete details are available in the July 1987 issue of **Byte's** "Best of BIX: Apple" column, on pages 305-310. Assuming I now understand the whole problem, which is debatable, the complete fix is:*

```
Immediately before CHAIN or STORE:
IF PEEK(49149)=1 THEN POKE 41859,3

Immediately after CHAIN or STORE:
IF PEEK(49149)=1 THEN POKE 41859,7
```

Basic.system 1.1 is unusually bug free, so I'd hate to see it modified much, but I do think Apple should give us a version 1.2 to fix this particular bug.

Basic choices

Is there a different Basic for the Apple IIe?

Daniel Mason
San Jose, Calif.

Quickly skipping by all the CP/M Basics available to you if you have a CP/M card, and all the MS-DOS Basics available if you have a PC Transporter, and all the upcoming Basics that are IIgs-specific, the answer is yes.

But first sit back and decide what exactly it is you don't like about Applesoft. Is it lack of speed? Is it limited memory space for variables? Is it lack of structured loop commands? Is it a flaccid command set? Is it difficulty with adding machine language modules? Is it line numbers? Is it two-character variable names? Is it lack of portability to other computers? Go make a list of what's important to you and meet me back here in five minutes.

Now, if lack of speed and memory space are your only major complaints, what you need is the **Beagle Compiler** (current version is 2.5). We don't want to be redundant, so see our February and March 1987 issues, pages 3.1-3.2 and 3.9, and "Expanding Applesoft" in last month's issue, page 3.80, for more information. The **Beagle Compiler** will run on any 64K or larger Apple II, supports aux-mem and Apple-standard memory cards, is ProDOS-based, and is unprotected. Distribution of self-running disks requires a royalty payment of $50 a year ($74.95, Beagle Bros, 3990 Old Town Ave, #102C, San Diego, CA 92110 619-296-6400).

If lack of structure and a weak command set are your major problems, take a look at **Blankenship Basic** (current version is 2.7). Like the **Beagle Compiler**, it is highly compatible with Applesoft itself—the only Applesoft command **Blankenship Basic** doesn't support is HGR2. On the other hand, it adds a number of new commands that allow structure, such as REPEAT/UNTIL, WHILE/ENDWHILE, WHEN/ELSE/ENDWHEN, and LOOP/EXITWHEN/ENDLOOP. It adds commands that allow routines to be called by name rather than by line number. It can print text and it can draw and fill boxes on the graphics screen; it has SORT, SEARCH, and PRINT.USING commands; and it has facilities that allow you to save often-used subroutines in individual files and easily merge them into programs you are writing. It includes its own line editor. For more, see our answer to "Toward a perfect Basic" in March 1986, page 2.14. **Blankenship Basic** comes in both DOS 3.3 (any 48K Apple II) and ProDOS (any 64K Apple II) versions, is unprotected, and can be distributed on self-running disks as shareware with no royalty payment. ($25 DOS 3.3 or ProDOS, $39.95 for both, from Blankenship and Associates, PO Box 47934, Atlanta, GA 30362 404-491-3151).

In between the **Beagle Compiler** and **Blankenship Basic** is an ever-growing collection of ampersand packages that add various abilities to Applesoft. This collection is much too large to tick off here; if we tried we'd be sure to unintentionally insult at least one of the dozen or more subscribers who has sent us an ampersand package to review, to say nothing of overlooking stuff we haven't seen. So how about this—we'll throw out all the ampersand packages we've received and start over with two new rules. To get a plug in **Open-Apple**, an ampersand package must 1) be compatible with Roger Wagner Software's **Toolbox Series** and 2.) be compatible with the **Beagle Compiler**. These rules mean that all ampersand packages we talk about from here on will be state-of-the-art and will be compatible with each other. Those of you who aren't familiar with Roger Wagner's **Toolbox Series** should look at "The ampersand solution" in our September 1986 issue, page 2.63.

Pushing just past Applesoft and ampersands, we next come to **ProBasic** (current version is 1.0), developed by the same Alan Bird who wrote the **Beagle Compiler**. **ProBasic** is an Applesoft-based language you can extend by writing "modules," using either assembly language or **ProBasic** itself, that become **ProBasic** commands. The modules support local variables (ie, using FOR I=1 to 10 in a module will not change the value of I in the main program), parameter passing (ie, in POKE 768,0, "768" and "0" are "parameters" that are passed to the command POKE), and recursion (a module can call itself). **ProBasic** includes a VIRTUAL module that allows you to put arrays on any ProDOS-compatible disk device, including RAMdisks (array size is limited only by the size of the device), and modules can be easily switched in and out of memory. **ProBasic** also comes with a full-screen editor (well, ... actually, ... you buy the full-screen editor, **Program Writer**, and you get **ProBasic** free on the back of the disk). **ProBasic** is ProDOS-based, works on any 64K or larger Apple II, supports any memory card that can be configured as a RAMdisk, is unprotected, and can be distributed on self-running disks by leaving in the copyright notice. For additional information see "**ProBasic, Logo**" in our September 1986 issue, page 2.62. ($49.95 from Beagle Bros, address above.)

While **Blankenship Basic** and **ProBasic** solve many of the problems people have with Applesoft, neither is compatible with the **Beagle Compiler**. To get speed and structure without throwing everything you know about Applesoft away, the best compromise is **Micol Basic** (current version is 2.1). It's fast like the **Beagle Compiler**, it provides structure like **Blankenship Basic**, it allows multi-character variable names (the first package mentioned so far that does), has local and global variables, named procedures with parameter passing, and includes an editor. It is not totally Applesoft compatible, but it is close. We carried a rather extensive review in September 1986, page 2.61. **Micol Basic** is ProDOS-based, will work on any 64K Apple II, is unprotected, and can be distributed on self-running disks by leaving in the copyright notice. A 90 to 95 per cent source-code compatible IIgs version of **Micol Basic** is under development. (US$69.95, Micol Systems, 9 Lynch Rd, Toronto, ONT, Canada M2J 2V6 416-495-6864).

A step farther away from Applesoft is **ZBasic**, which is available for DOS 3.3 (current version 3.20), ProDOS (current version 4.00), CP/M, the Macintosh, and MS-DOS. The big strength of ZBasic is the portability of its programs. In theory, you write just one program, then, using different compilers, you can make the program run on any of the supported computers. The advantage of being able to move a program from one computer to another so easily is obvious. Even graphics transfer—Dennis devised a program that rotates a pyramid on the screen; when run under CP/M, which doesn't support graphics, the pyramid's lines are drawn on the text screen with text characters. But the disadvantage of this universality is that you have to move away from familiar Applesoft commands into a kind of Basic "Esperanto" that has a closer resemblance to the CP/M or MS-DOS versions of **MicroSoft Basic** than to Applesoft. **ZBasic** includes a full screen editor; is the only Basic mentioned here that allows you to completely do away with line numbers; provides for long, case-sensitive variables; and includes a feature called "long functions" you could use to emulate the modularity of **Blankenship Basic**, **ProBasic**, or **Micol Basic**. A IIgs-specific version is reportedly under development. Although capable of 2- to 54-digit precision, **ZBasic's** major weakness is the speed of floating-point calculations, which are noticeably slower than standard Applesoft at similar precision. The DOS 3.3 version of **ZBasic** runs on any 64K Apple, the ProDOS version requires 128K and won't work on a II-Plus. **ZBasic** is not copy-protected and can be distributed on self-running disks by leaving in the copyright notice. ($49.95 for first package, $39.95 for others—all use the same manual—Zedcor, 4500 E Speedway, #22, Tucson, AZ 85712 602-881-8101). An Applesoft to ZBasic source-code translator program is available for $29.95 from Bringardner Data Products, 1736 E North Broadway, Columbus, OH 43224.

Finally, out on the edge of the Basic universe is **Promal** (see this month's front page), which is more Basic-like than most other computer languages. Beyond the edge of the Basic universe, don't forget that Apple II versions of **Pascal**, **C**, **Forth**, and **Logo** are also easily available.

Open-Apple

is written, edited, published, and

© Copyright 1987 by
Tom Weishaar

Business Consultant	Richard Barger
Technical Consultant	Dennis Doms
Circulation Manager	Sally Tally
Business Manager	Sally Dwyer

Most rights reserved. All programs published in ***Open-Apple*** are public domain and may be copied and distributed without charge. Apple user groups and significant others may reprint articles from time to time by specific written request. Requests and other editorial material, including letters to Uncle DOS, should be sent to:

Open-Apple
P.O. Box 7651
Overland Park, Kansas 66207 U.S.A.

Published monthly since January 1985. World-wide prices (in U.S. dollars, airmail delivery included at no additional charge): $24 for 1 year; $44 for 2 years; $60 for 3 years. All single back issues are currently available for $2 each; bound, indexed editions of Volume 1 and Volume 2 are $14.95 each. Volumes end with the January issue; an index for the prior volume is included with the February issue. Please send all subscription-related correspondence to:

Open-Apple
P.O. Box 6331
Syracuse, N.Y. 13217 U.S.A.

Open-Apple is available on disk from Speech Enterprises, P.O. Box 7986, Houston, Texas 77270 (713-461-1666).

Unlike most commercial software, ***Open-Apple*** is sold in an unprotected format for your convenience. You are encouraged to make back-up archival copies or easy-to-read enlarged copies for your own use without charge. You may also copy ***Open-Apple*** for distribution to others. The distribution fee is 15 cents per page per copy distributed.

WARRANTY AND LIMITATION OF LIABILITY. I warrant that most of the information in ***Open-Apple*** is useful and correct, although drivel and mistakes are included from time to time, usually unintentionally. Unsatisfied subscribers may return issues within 180 days of delivery for a full refund. Please include a note from your parents or children confirming that all archival copies have been destroyed. The unfullfilled portion of any paid subscription will be refunded on request. MY LIABILITY FOR ERRORS AND OMISSIONS IS LIMITED TO THIS PUBLICATION'S PURCHASE PRICE. In no case shall I or my contributors be liable for any incidental or consequential damages, nor for any damages in excess of the fees paid by a subscriber.

ISSN 0885-4017 **Source Mail: TCF238**
Printed in the U.S.A. **CompuServe: 70120,202**