A2-Central™

formerly
Open-Apple

March 1991
Vol.7, No.2

ISSN 0885-4017
newstand price: $2.50
photocopy charge per page: $0.15

A journal and exchange of Apple II discoveries

An Applesoft for the 1990s

From the amount of mail we get regarding "fixes" to programs, it seems the majority of users have two desires for their computers: they want the computer to work the way they want it to and (in an increasing number of cases) they want this ability without having to learn technical details or a new language, specifically a programming language like Pascal or even BASIC.

Obviously, the only way both of these objectives can occur (until mind-reading, self-programming computers arrive) is if you hire someone to write custom programs to do exactly what you want; that will be $100+ per hour, please. Given that an individual program can take weeks, months or years to write, paying a programmer isn't a viable option for most people. Even if you have unlimited funds, communicating what you want in such a way that another person can interpret and implement it isn't easy to do.

For 8-bit Apple II systems, many users would bite the bullet and learn Applesoft to do the programming that BASIC allowed. It wasn't unusual to see small specialized databases, or loan amortization programs, or other applications concerned with manipulating small to moderate amounts of data written in Applesoft.

But as a computer matures, users expect it to be able to do more, and do it faster and better. For most users, homebrewed BASIC programs eventually give way to commercial word processor, database, spreadsheet, and graphing packages that are written in assembly language to be fast and powerful. BASIC is still useful to do your custom programming in, but it has become more of a personal pastime. Look at the top selling programs for the Apple II and it's unlikely that any are written in Applesoft. The ante has gone up.

The popularity of graphics interfaces resembling those on the Mac and IIgs only widened the gap between the recreational and the professional programmer. Whether you are writing programs for the Mac, IIgs, or MS-DOS *Windows* environments there is a lot of ground to cover beyond learning a programming language itself; several *thousand* pages of information regarding user interface design and the tools provided in each of the environments to support it.

The first wildly popular end-user programming language for the Macintosh was not BASIC; BASIC versions for the Mac pretty much fizzled as they were introduced. In 1986, Apple started bundling a programming envionment by Mac guru Bill Atkinson with new Mac systems and that environment has spawned (by Apple's account) over 200,000 applications since. Atkinson's brainchild was HyperCard, and, along with desktop publishing, it made the Mac viable. HyperCard sold Macintosh computers; people wanted the programming environment and the Mac was the only computer that had it.

But no more. Apple took HyperCard 1.2.5, added color and other significant features, and made it run on the Apple IIgs. The program is available to dealers now. It's called *HyperCard IIgs* and it promises to change the Apple IIgs computing experience.

The HyperCard IIgs package consists of six 3.5 disks and three manuals. The package retails for $99 and includes all of the information you'll need to start creating your own simple stacks. Since it is a development environment, if an Apple dealer in your area isn't carrying it, you can order it direct through Apple's developer tools source, APDA. (Or you can order it from us or other mail order outfits who are buying in bulk from authorized Apple dealers.)

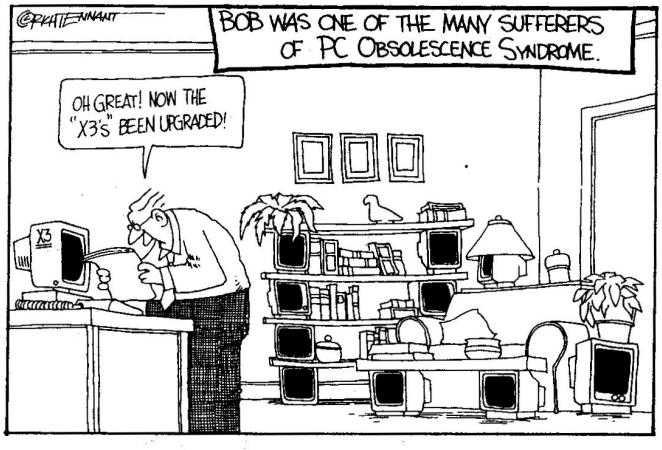
The three manuals include a tutorial, a reference, and an introduction to programming HyperCard IIgs. The disks include the HyperCard program, installation program, "tour" stacks to demonstrate features and capabilities, comprehensive help, and additional tools.

HyperCard IIgs requires a minimum of 1.5 megabytes of RAM memory (two megabytes is recommended) and a hard disk. Once installed, the program, including help files and tools, occupies close to four megabytes of disk space, not counting the space used by GS/OS.

HyperCard IIgs joins two other IIgs products that embody the concept of *hypermedia*, *HyperStudio* and *Nexus*. *Multimedia* is the integration of various audiovisual means of communication; *hypermedia* is multimedia plus user interaction (interactive multimedia is probably a more accessible term, but it takes longer to say and type). Hypermedia usually combines elements of text, graphics, sound, and other methods of communication under the control of a user communicating with the various elements through a computer.

Hypermedia grew out of a concept espoused by Ted Nelson (author of *Computer Lib* and *Literary Machines*) called *hypertext*. Unlike "normal" text forms such as books, which are primarily accessed linearly, hypertext allows the user to jump around among concepts in a nonlinear fashion by defining linkages between the concepts. The IIgs program *Nexus* (by DataSmith) is the IIgs software product closest to true hypertext; although *Nexus* also deals with graphics and sounds, its ability to link a specific word within an editable text field to another object (and not have that link corrupted when the text is altered) is unique. *Nexus* "stacks" are distinctly file oriented, not following the card-based model of most hypermedia programs; links are established with a simple "point and click" mechanism. If you wanted to publish a hypertext-based manual where clicking on an unfamiliar word would take you to a glossary entry or (if you prefer more depth) an entire new discourse on the meaning of the word, *Nexus* is the tool you seek.

The other prominent program is, of course, Roger Wagner Publishing's *HyperStudio*, which is the consummate hyper*media* program. *HyperStudio* uses the basic card-button-field model associated with most hypermedia programs; it's unique strength versus the other IIgs hypermedia programs is the depth and breadth of the actions that can be associated with its buttons without requiring the user to learn



either scripting or programming. (*HyperStudio* does have a scripting capability, but it is used primarily for controlling external command enhancements rather than the *HyperStudio* program itself.)

With these programs already available, *HyperCard IIgs* itself has been categorized by some as "wasted effort" by Apple. Given the amount of effort Apple had to commit to its development, one has to believe there is some aspect of *HyperCard IIgs* that Apple felt made it "special". HyperCard's special talent isn't as easily discerned as that of *Nexus* or *HyperStudio*, the strength of either of these programs can be demonstrated in a few minutes by sitting down and creating a few stacks. In conversations, Apple's engineers have been very complimentary of the other products, and have also been very careful to distinguish the purpose of *HyperCard IIgs* from these products.

The importance of *HyperCard IIgs* stems from its design as a programming environment. HyperCard's structure starts with an individual screen of information referred to as a *card*; a collection of these cards can be linked into an organized collection called a *stack*.

A card also has a *background*, which can contain the same elements as the card itself; these elements can be graphics, buttons, or (text) fields.

All of these items are combined in a layered hierarchy in the order of card buttons and fields, background buttons and fields, the card(picture), the background (picture), the current stack, the "Home" stack (a special stack used as "home base" by HyperCard), and the HyperCard program itself. Figure 1 shows a sample card with some of these items; you can see how card items obscure the background items where they overlap. The two "windoids" (detached windows) at the lower right are not card items but actually tool and paint pattern menus that can be moved around the screen as you work. The Foreground button at the lower center of the screen laps off the card picture and extends over the background field and obscures part of it, just as the card picture blocks our vision of the lower right corner of the background field. But since all buttons and fields take precedence over card and background pictures, a background button can be activated even if hidden by a card picture. For example, in our example,moving the HyperCard Browse cursor (the "hand" beneath the Foreground button) up to the tip of the arrow above the button and clicking will cause the background field to activate even though we're clicking on top of the card picture.

This structure is one important concept of HyperCard IIgs; but the most important feature is the way the integration of the items into the structure is accomplished. The component items of *HyperCard IIgs* are referred to as *objects* in the sense of a programming model known as *object oriented programming*, or (abbreviated) *OOP*. OOP refers to the way in which you approach designing programs rather than to a specific programming language you might use. Some languages are associated more with object oriented programming, just as some are associated more with other distinct types of programming styles (Pascal and C versus BASIC or assembly language for structured programming, for example).

Object oriented programming attempts to construct a program from smaller modules referred to as *objects*. Each object contains the information it needs to perform it's designed task including internal data the object may need and any procedures that the object can perform. This set of data and procedures is "private" for each object.

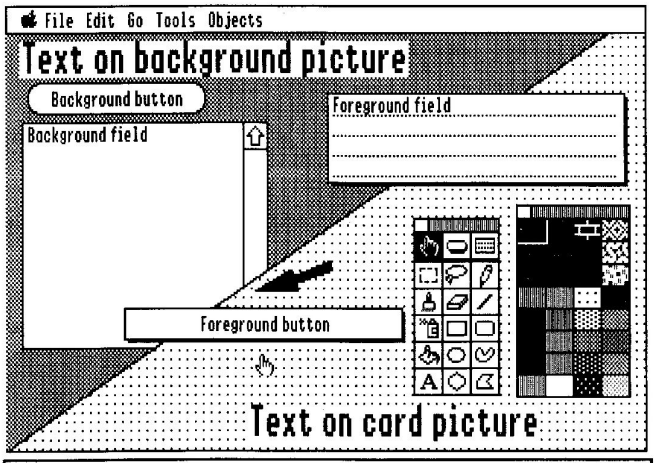


Figure 1: A layered HyperCard screen

Anyone who has written a program of more than a few lines is familiar with the concept of modularizing their code; in BASIC modules are usually implemented as subroutines. Obviously, a non-modularized form of a program to print the squares of ten numbers:

```
10 PRINT 1^2
20 PRINT 2^2
30 PRINT 3^2
40 PRINT 5^2
50 PRINT 7^2
60 PRINT 11^2
70 PRINT 13^2
80 PRINT 17^2
90 PRINT 19^2
100 PRINT 23^2
110 END
```

is more difficult to revise than a version where the calculation is placed within a subroutine module:

```
10 FOR I = 1 TO 10
20 READ X
30 GOSUB 50
40 NEXT
50 END

60 PRINT X^2
70 RETURN
80 DATA 1,2,3,5,7,11,13,17,19,23
```

But when we modularize the BASIC program, we have to use a variable so that we can pass different values to a subroutine written to perform an operation on a "generic" variable ("X" in our example). Each time we invoke the subroutine we do so with X set to a different value.

Applesoft doesn't distinguish between variables used in the main program and those used in subroutines. Some languages do; for example, in Pascal our subroutine and main program might look like this:

```
program doSquares(Output);

var
    X:      Real;
    i:      Integer;
    Data:   array[1..10] of Real;

procedure printSquare(ourX: Real);
begin
    writeln(Output,sqr(ourX));
end;

begin
    Data[0] := 1;   Data[1] := 2;   Data[2] := 3;
    Data[3] := 5;   Data[4] := 7;   Data[5] := 11;
    Data[6] := 13;  Data[7] := 17;  Data[8] := 19;
    Data[9] := 23;

    for i := 1 to 10 do begin
        X := Data[i-1];
        printSquare(X);
    end;

end.
```

In this example, X is passed from the main program to the subroutine printSquare which uses the local variable ourX as the argument. Since the value passed to printSquare by the main program is a real (floating point) value and printSquare expects a real value, printSquare happily copies the value in X into its local variable ourX for its internal use. This gives our main program an additional stage of isolation from having to know how printSquare operates; we don't have to know the name of the internal variable printSquare uses, we

only have to pass the routine the proper number and type of arguments.

With such non-OOP design, usually a subroutine is limited to handling a specific type of data (integer or real or text values). You define a common structure for variables in the main routine and subroutine for any data to be passed, then call the specific subroutine to be performed.

Object oriented programming wants to take this isolation of specifics within the modules much further. OOP concepts include *encapsulation* of data and procedures within a module so that the module becomes a type of software "black box" that performs an operation at the behest of the programmer without requiring that the programmer know detailed specifics about the construction of the module. Using the module involves a process more like communication than the "set up data and invoke subroutine" process used in "classic" programming. This communication is performed by sending a *message* to the object, which then interprets and acts upon the message (even relaying it's own messages if necessary) according to its design.

HyperCard elements such as buttons, fields, cards, stacks, and the HyperCard program itself are all *objects* (in the OOP sense) that accept and relay messages according to their heirarchy. This makes the mechanism of implementing a HyperCard stack substantially different than that for *HyperStudio*.

To allow communication among its objects, HyperCard IIgs includes an OOP language: a dialect Apple calls HyperTalk. If you've been intimidated by programming languages you've seen in the past, you may find HyperTalk to be much more "natural" in its form. You can probably guess that

```
on mouseUp
    visual effect scroll left
    go to next card
end mouseUp
```

waits for a mouseUp event (waits for the release of a depressed mouse button), defines a visual effect for implementing a leftward scroll, then moves to the next card in the stack using the specified effect.

This *script* obviously can't operate in a vacuum; the mouseUp event message has to come from somewhere else. In HyperCard the script would be associated with a specific object as a *handler* for a mouseUp message. Every HyperCard object has a script, though the script can be empty.

HyperCard's main program tracks the occurrence of events such as mouseUp and reports them as messages to affected objects, following the hierarchy. If the above script were assigned to a button on the current card and you clicked within the button, the button would receive a mouseUp message that would trigger the program and move you to the next card. If the button had no such script, the mouseUp message would be passed along to the next object in the hierarchy until it encountered a handler; ultimately the message would be returned to HyperCard itself if no other object contained a handler.

HyperCard implements its stacks through this combination of objects and associated scripts. This makes it very distinct from *HyperStudio*, which is much more user-friendly in the way it allows you to add actions to its elements.

To define a *HyperStudio* button that plays a sound, you ask *HyperStudio* to create a new button on your current card. After you position and size the button, *HyperStudio* presents a dialog box with many options that can be defined for the button. One of the options is "Play a sound"; you select this option and then decide if you want to add an existing sound from a file on disk or even record the sound at that time using *HyperStudio*'s provided digitizer and microphone. *HyperStudio* gives you the option to add many such elements just as easily: sounds, animation, laser disc control (if you have the proper disc player connected), and so on.

HyperCard IIgs requires you to incorporate scripts to do anything other than simple card movements. You can add such a function by cutting and pasting an existing button that performs the desired action from another card; Apple has broken the "chicken and egg" cycle by supplying a selection of pre-made buttons as part of a Tools stack in the *HyperCard IIgs* package. But, from a user perspective, the incorporation of new elements into a *HyperCard IIgs* stack is less intuitive than it is for *HyperStudio* or *Nexus*, which primarily use mouse "point and click" methods to select actions.

Why did Apple design *HyperCard IIgs* to work the way it does? The overriding consideration in the design of *HyperCard IIgs* was compatibility with the Mac version of HyperCard; *HyperCard IIgs* is essentially a IIgs version of HyperCard version 1.2.5 for the Macintosh, with some enhancements. Although some features had to be altered slightly for the IIgs environment where it differs from that of the Macintosh, in most cases those alterations are in favor of the IIgs.

Version 1.2.5 (rather than the new 2.0) for the Mac was chosen as the model for *HyperCard IIgs* since it was the most advanced existing standard at the time *HyperCard IIgs* was being developed and almost all stacks currently available operate under that version. The current Macintosh HyperCard v2.0 added integrated sound capability for the new Mac LC and IIsi (which include sound input as standard); although *HyperStudio's* digitizer demonstrates the IIgs can easily and inexpensively support a similar capability, Apple hasn't made it standard yet, however, and *HyperCard IIgs* doesn't currently support digitization. Multiple windowing capability for all Mac systems was also added to v2.0; the IIgs is limited to the single card per window design of previous Mac HyperCard versions. But the IIgs version adds better printing support, also a concern of v2.0 for the Mac, and the IIgs version supports color, which *not even v2.0 for the Mac has.*

By far the most important feature of *HyperCard IIgs* is it's programmability through the HyperTalk language and the compatibility of that language with the Mac version of HyperCard. HyperCard 1.2.5 and *HyperCard IIgs* therefore become parallel development environments for the two Apple product lines accessible to the end user. Unlike Apple's "parallel" development environment for programmers (MPW and MPW IIgs) where all development is performed on the Mac side, there is a near-parity of HyperCard development techniques whether you use a Mac or a IIgs to develop your stacks.

HyperTalk integrates with the HyperCard environment by defining the terms and syntax that allow a script to react to and modify the HyperCard environment. Our early example showed both sides; our handler reacts to a mouseUp message by defining the type of transition to use and then commanding HyperCard to move to the next card. But scripts in *HyperCard IIgs* can do much more.

We can issue messages that will be passed to other objects. Any message not intercepted by another handler in the hierarchy will eventually find its way to the supervising HyperCard program; if the message corresponds to a valid HyperCard command it will be executed. For example

```
doMenu "Next"
```

will cause HyperCard to examine its menus for an item name "Next" and execute that function (move to next card in the stack).

If you want to re-define this function, you can do it by interposing an object with its own handler for the doMenu message. For example, we could add the following script to our card background:

```
on doMenu menuItemReq
  if menuItemReq is "Next" then answer "I'm sorry, Dave, but" && ¬
  menuItemReq && "is not available"
  else pass doMenu
end doMenu
```

and if you select "Next" HyperCard will throw up a dialog indicating that the menu item isn't available. (And you'd better remember to include the "pass doMenu" to pass *other* menu items or your menus will be disabled!)

You can also see from this example that HyperTalk does have similarities to other programming languages. There are control structures; the above handler contains an "if-then-else" construct. There are keywords and symbols: "answer" displays a text string in a dialog box, the "&&" characters are used to combine portions of the text (adding a separating space) to create the string, and the "¬" character is used to indicate a "soft return" (an end of line that doesn't signal the end of our "if" statement).

There are also portions of HyperTalk that are more like a "natural" language, such as allowing the use of the word "is" to indicate "="

(though we could have used "=" instead). We can't get into all the features of HyperTalk here, but for the hardcore Apple has completed a HyperCard IIgs *Script Language Guide* which is already in publication by Addison-Wesley.

HyperTalk supports *functions* that return values. Just as Applesoft has a SQR(X) function that returns the square root of the value of X, HyperTalk has a sqrt(X) function. HyperTalk's functions are rich; in addition to including most basic math functions, financial functions (such as annuity() to calculate the return on an investment), functions to return the status of the environment (such as tool() to return the number of the currently chosen HyperCard tool), and others are included.

Both commands and functions can be defined externally by writing a program module to interface to *HyperCard IIgs* and enhance it; Apple supplies some of these "XCMDs" and "XFCNs" as part of the "Tools" stack. The *Language Guide* includes a list of "callbacks" (entry points to *HyperCard IIgs* routines) that can be used in writing these modules.

HyperCard objects have *properties* that can be read and altered via HyperTalk. For example, the location of the bottom right corner of an object is contained in its "bottomRight" property, which can be read or reset by a script. Each object has its distinct set of properties tracked by HyperCard; you can read or change them if you like, but most of the time you can ignore them and let HyperCard manage them. One of the properties of a HyperCard object is its associated script.

It's confusing to talk about HyperTalk "programs" since HyperTalk scripts are not operable independent of a programming environment. There is no such thing as a "HyperCard runtime package"; in order to run a HyperTalk script, which can access every aspect of HyperCard, the HyperCard program must be used along with it. Therefore, in order to distribute stacks, you must assume your potential audience already has *HyperCard IIgs*, or you must license the *HyperCard IIgs* program (sans the support stacks and manuals supplied in the user package)to distribute along with your stack.

We expect *HyperCard IIgs* and HyperTalk to be the Applesoft-like environment IIgs owners have been looking for. The big complaint about the IIgs is that most of its feature enhancements over older Apple II systems haven't been accessible to those wanting to design their own programs. The learning hurdle for IIgs ToolBox programming is high compared to being able to use Applesoft on an Apple IIe, for example. *HyperCard IIgs*, like its Mac brethren, gives you a playground to access the graphics environment. Also like the Mac version, this doesn't give you the ability to write any program you want, but some classes of programs will be possible. We do hope to add HyperTalk to discussions of programming as the "everyperson's" language of the IIgs, as Applesoft is for the 8-bit Apple II's.

One class HyperTalk makes accessible is data file programming. Part of HyperCard's command set allows for reading and writing files, and it's relatively easy to write a script that allows importing a tab-delimited stack into a set of cards:

```
on mouseUp
  ask "File to import?" with "A2.Sources"
  if it is empty then exit mouseUp
  put it into fileName
  open file fileName
  repeat forever
    read from file fileName until return
    if it is empty then
      go to first card
      close file fileName
      exit mouseUp
    else doMenu "New Card"
    put tab into last char of it
    repeat with x=1 to the number of fields
      put char 1 to offset (tab,it) of it into field x
      delete last char of field x
      delete char 1 to offset (tab,it) of it
    end repeat
  end repeat
end mouseUp
```

This program looks somewhat more "English-like" than BASIC or Pascal. Even a neophyte may be able to follow it.

When a mouseUp message is sent to the object that is assigned this script, the handler executes it (we assigned it to a button on a card that had been designed with the proper number of background fields). "Ask" puts up a dialog that allows you to input a line of text with the pathname of the text file you wish to import. In this case, a default pathname of "A2.Sources" is used, which you can accept (by pressing "Return", or using the mouse to select the "OK" button ask will provide in the dialog). The response is put into the variable "fileName" and *HyperCard IIgs* attempts to open the file.

Assuming all goes well (the file is there and can be opened), we enter a repeat loop that will be executed until "forever"; that is, until some condition occurs that causes the loop to abort.

Each record is read by taking input from the file until we reach a "return" character. HyperTalk recognizes some common data

---

Ask (or tell) Uncle DOS

Roger Wagner called last month with a comment regarding the failure of Apple monitors (see "Fading Image," February 1991, page 7.6). Roger says he has two of the monitors that do the "dimming" trick and has discovered a magic procedure that "corrects" it for about 6 months.

*Neither Roger nor I are very anxious to expound on his exact technique to others since an overexhuberance in application of the technique may result in bodily damage to the monitor or yourself. But his treatment implies that the problem may be physical rather than electronic; one postulate is that the "yoke" controlling picture clarity may shift as the monitor ages (Roger didn't know for sure, having not opened the monitor, and we stay strictly away from sources of lethal voltage ourselves). Should that be the case, armed with a description of the symptoms a knowledgeable television technician **may** be able to adjust the monitor. Given the cost of a new monitor, it may be a recourse worth investigating.*

*I also need to issue myself a slap in the head. Last month in "Expectations versus reality' (pp. 7.7-8) I had two separate suggestions; one for increasing the word processor limit of AppleWorks v3.0 with Applied Engineering's expander software, another for adding **TimeOut Paint** to add paint capability. Mark Munz from Beagle Bros wrote to remind me that Beagle does not not recommend combining those two enhacements due to some compatibility issues. AppleWorks v3.0's word processor does provide 16,250 lines of capacity (assuming you have enough expansion memory to hold a document of that size), so the built-in expansion may be sufficient.*

*Last month's article on "The need for speed" drew a few responses, including some ROM 01 IIgs owners who have had problems using the **ZipGSX** with ProDOS 8 programs. Some problems have been traced to a grounding problem that can be corrected; if you're having these symptoms, contact Zip for details.*

There were also a few callers puzzled by the benchmark results, so let's open that can of worms...—DJD

Apples and oranges

A past issue of *Nibble* magazine had a question from a reader wondering if anyone had ever done a speed comparison between Apple and IBM computers. I could not remember ever seeing such a comparison, so I decided to do some testing myself. I do not have the "scientific" benchmark programs that are often quoted in advertising (by vendors wanting you to buy

constants, including the return character in this case, by name. HyperTalk also recognizes the special local variable "it" as referring to the object we are currently dealing with; in this case, a variable container which holds the text curently being read.

If the record we read is empty (likely to happen at the end of the file), we jump back to the first card and exit our mouseUp handler.

Otherwise, we create a new card to hold the record. "DoMenu" executes the named *HyperCard IIgs* menu item that creates a new card using the current background, including any associated background buttons and fields.

Next we change the last character of the record we read to a tab, from a return, in preparation for scanning the string.

Another "repeat" loop scans the string; the "number of fields" property of the current card is used to identify how many times the loop must be repeated. For each field, we read the characters from the start of the record up to and including the first tab character encountered, copy that segment of the record into the corresponding field, delete the last character of the new field (the tab character at the end of the field), and then delete the information we just copied from the start of the record.

All that's left is to continue copying records until reading a record into "it" results in an empty record (nothing was there to read), at which point we end our handler for mouseUp, and we have a stack of cards.

You'll notice the indentation of the script illustrates the structure; HyperCard's editor enforces the formatting for you. Any lines that don't indent properly serve as an indication of a structural problem in your program (logical errors you still have to catch yourself, as we continually discover the hard way).

HyperCard can be used for hypermedia, but it actually has some prominent weaknesses versus *HyperStudio* in this area, notably in ease of creating stacks consisting primarily of multimedia elements. HyperCard's scripting ability may help in some circumstances, but *HyperStudio* has also been adding this ability with more of an eye toward its hypermedia design.

But as an "Applesoft of the '90's", HyperCard may find it's niche. At last, writing simple programs is easy again.–DJD

Miscellanea

***InWords* is here!** Combined with a Vitesse *Quickie* scanner, *InWords* gives you the ability to scan a page of printed text and process the scanned document into text that can be loaded into a word processor.

InWords can be "trained" to recognize various font shapes. During a training session, a printed sample of the font is scanned and as *InWords* processes the scan it displays character patterns it can't identify and allows you to enter the corresponding sequence on the keyboard or (if the character seems ill-formed) skip to the next pattern. Once a font is trained, the font recognition table can be saved for future use with the same font. *InWords* includes a few tables with the program. Multiple fonts can occupy a single table; such an option is handy for text such as ***A2-Central***, where we use at least three fonts (Helvetica, Benguiat, and Courier) on a regular basis.

Once you have the font table (either one supplied, or one you've trained), you can start scanning documents. *InWords* allows you to scan text in narrow columns (like this newsletter) and ignore the text to either side of a column. Or you can scan a text column wider than the scanner head in two passes (one for the left side and one for the right) and *InWords* will splice the two halves together into a document for processing. Once the scan is converted, the resulting text can be edited using *InWords*' own editor, or saved to a file on disk in several formats for use with other programs.

You can't expect perfection from optical character recognition software; occasionally "O" ("oh") will be confused with "0" (zero) and "l" (lowercase "L") with "1" (one). But you don't spend much time waiting for *InWords* to complete the conversion, and you can edit within *InWords* if you want to correct the text while the freshly scanned original is at hand.

The good news for Apple IIe owners is that *InWords* will work on an enhanced IIe with at least 512K of expansion memory (an accelerator is recommended); *InWords* is ProDOS-8 based. It also works on a IIgs with at least 512K of expansion memory.

It's tax time. We're not planning on reviewing tax programs, but users have recommended the old standbys: *1040Works* ($29.95 plus $3 shipping and handling, now distributed by NAUG, Box 87453, Canton, Mich. 48187, 313-454-1115), *SwiftTax* (TimeWorks, Inc., 444 Lake Cook Road, Deerfield, Ill. 60015, 800-535-9497), and HowardSoft's comprehensive*Tax Preparer* (HowardSoft, 1224 Prospect Street, Suite 150, La Jolla, Calif. 92037, 619-454-0121).

Weve also received (as far as we can recall) our first Canadian tax form templates (for AppleWorks's spreadsheet) from Granite Software, Box 105, Postal Station Toronto, Ontario, M6B 3Z9. The core template consists of a questionnaire, T1 General Tax Form, the tax schedules, and calculation tables for items such as the Federal Political Contribution Tax Credit, RRSP, Unemployment Insurance Benefit Payment, and so on. If you have *TimeOut SpreadTools*, independent templates can be linked to the main template with CelLink

The current template is $34.99 plus & 7% GST (and 8% provincial sales tax for Ontario residents).—DJD

---

their brand of computer) but then I also do not know anyone who does the type of computing that these tests are written to emulate. So, since what I do have is access to several of the models of each type of computer, I wrote a simple program myself.

Most, if not all, of the tests that are run by the technical review publications are variations on mathematical calculations to fully exercise the processing features of the particular computer microprocessor. That is a wonderful way to test—if you have such a program available. The average person, on the other hand, is using a computer to do word processing, data management, and to build a spreadsheet. All of that requires a computer that is reasonably fast both in its internal operations **and** in interactions with the user by means of the display monitor.

I want to make a point of that distinction because it is possible to go out and spend a small fortune on the fastest computer you can buy and then be disappointed by the "apparent" speed of the computer when you really use it with your favorite program. Now, if you will remember that idea for a few moments, on to a description of the test I wrote and the computers that I used for testing.

The test I used was actually very simple. The only common element that I had to make the test as near the same for each brand as possible was the interpreted BASIC that comes with each of the appropriate operating systems. On an Apple II, that is currently the BASIC.System that comes with ProDOS. On an IBM or clone, that is the GWBASIC.EXE that comes with MS-DOS version 3.21 that I have. Since each BASIC was written by Microsoft, I called them equal. I wrote a small program and timed how long it took to run on each computer. For the IBM and clones I also ran the *Norton Utilities's* SI program. Since I do not have the equivalent of that measure for the Apple II, I will list the Apple clock speed in the results table. The program follows:

```
1 FOR I = 1 TO 1000
2 T = I
3 T = (T*I)/(T+I)
4 PRINT T
5 NEXT I
6 PRINT "Done"
```

This is not a very complicated program, but it tests both the speed of the computer **and** how fast the results are displayed.

Computer One is the Apple IIe that I used for several years and that is still good. I tested this computer two ways. The first was with the "stock" Apple 1 MHz clock.

Computer Two is the same computer but with a 3.6 MHz *TransWarp* accelerator card installed.

Computer Three is a true-blue IBM XT computer where I work. This computer runs at the 4.77 MHz clock speed that is the original standard of the IBM computer. One warning about clock speeds before you get to the test results table. The IBM uses the Intel 8088 processor and the Apple uses a 6502 processor designed by Western Design Center. Since the processors and the software are so completely different, you should be careful and not jump to the conclusion that the 4.77 MHz IBM will automatically be faster than the 3.6 MHz (or 1 MHz) Apple computer. An old IBM PC/XT, since it is the "standard" against which all MS-DOS computers are measured, had a 1.0 SI (system index)

rating when I ran the Norton program.

Computer Four is an AST Premium 286, also where I work. I do not know the clock speed of this computer, since our central staff who handles hardware very thoughfully keeps all hardware manuals. (I think it is 10 MHz.)

Computer Five is my own ALR 386/2 computer. It has a 16 MHz clock and is based on the "AT-standard" data bus, so it is not as fast as some of the newer 80386 computers that feature advanced bus design. It also does not cost as much as these more advanced computers, which was a very large consideration when I was shopping.

Computer Six is a recent model Compaq Portable III, with a 20 MHz clock and the 80386 microprocessor. This computer has one of the new motherboard data bus designs and is, in some operations, much faster than my ALR computer. It is also about twice as expensive as the computer I have.

Here is the test results table; all times are in seconds:

	1	2	3	4	5	6
Time	38	20	67	25	14	14
Index	1.0 M	3.6 M	1.0 SI	11 SI	13 SI	22 SI

The stock Apple IIe, with its lowly 1 MHz clock, was **not** the slowest in my time trials. That honor was reserved for the IBM computer. It was not until the AST clone with the 80286 processor was tested that an MS-DOS speed came in to beat the 1 MHz Apple IIe computer. It also took me a little while to figure out why the 3.6 MHz Apple was not 3.6 times as fast as the 1 MHz Apple. I finally decided that the simple test I had designed was not only testing how fast the computer would run **internally**, but also how fast the results could be **displayed** on screen. (Remember my note about "apparent" speed, back a few paragraphs?)

Another example of this difference between expected and actual speed, completely within the IBM-style computing world, is shown by the test results of my ALR and the Compaq. Even though the SI for the Compaq is nearly 70% higher than my ALR, their program test results were **exactly** the same.

This means that a display intensive program is likely to be very little (if any) faster when run on the Compaq than when run on my slower, cheaper ALR computer. Of course, calculation intensive programs will run faster on the Compaq. My new ALR "feels" faster than my accelerated Apple IIe when I run several equivalent programs, as verified by this test and the ALR's 14 seconds versus 20 seconds for the fast IIe.

While this is not a scientific test of every possible feature of each type and brand of computer, I do not think it shows that the standard Apple computer does not compare all that badly to the standard IBM computer. Of course, I have completely switched over to using an IBM clone based on availability of software I need. There are just as many (or more) user programs available for Apple as there are for IBM, but the high-level language compilers that I use (*Clipper*, COBOL, etc.) are just not there for the Apple II family. My wife, on the other hand, would not trade her AppleWorks for anything. So, before you brag too much about your IBM or clone, remember that clock speed is not everything...the program you use is the main reason to buy one type or brand of computer over another.

I have no intention of ever going back to using my old Apple IIe, but that is not due to my IBM clone being some "super" microcomputer. My decision also has nothing to do with the IBM computing standard being in some basic way "better" than the Apple II standard. Just as a Ford is not inherently better than a Chevy, an IBM or a clone is not automatically better than an Apple (or any other brand). When you are shopping for a computer the **programs** you want to use will determine the hardware platform that you buy. If you want to use programs that are only available for MS-DOS (like me), or if you must buy an MS-DOS computer that is compatible with someone else's computer, **then** you should buy an IBM or clone. Otherwise, you may be better off buying an Apple II or an Apple Macintosh.

Tom Smith
Portland, Ore.

Benchmark testing will often reach up and bite the person doing the testing unless the exact same code is being run on the exact same machine. But obviously, in most cases the interest that drives such testing is not an interest in comparing the speed of identical machines.

You've definitely pointed out some of the flaws of judging system speed by MHz, but most benchmarks are also limited in what they reveal about the real speed of the system by their very nature. There are too many variables that can affect the scope of what a benchmark tests, and even how well it tests within that scope.

Your simple benchmark may not be much worse than some others at judging the speed of a system; indeed, taking the screen speed into consideration is certainly more realistic than ignoring it (how useful is a computer without input and output?). However, let me poke a few criticisms at it:

Are the versions of BASIC really equivalent? Since your variables (I and T) will default to floating point values, is the floating point precision the same for all of the versions of BASIC you tried? (I'm willing to bet there is a difference between AppleSoft and the versions of MS-DOS BASIC you used).

Assuming there are differences in the precision of the BASIC versions in performing the calculations, this may also affect the accuracy of your video speed test (via PRINT). The more precise BASIC may print floating point values for T with more digits, which will change the throughput the video display is asked to handle.

Finally, is BASIC really a good test for the types of programs most people use? Most computers are used to run pre-programmed applications these days, and very few of these are written in interpreted BASIC.

The problem with benchmarks is that most of them spit out a number that may naively be interpreted as the relative speed of thatmachine. Under closer scrutiny, the number may be affected by system differences small enough to make application of the numbers to an individual computer (configured as the owner has chosen) less accurate.

Even seemingly "foolproof" tests can get tricky; some compilers "optimize" code they generate in ways that can trap an unwary benchmark author. For example, one might suspect that a simple test of raw speed for a system would be an empty loop:

```
FOR I = 1 TO 10000:NEXT I
```

But some compilers are trained to look for such an "empty loop" structure and skip past it, recognizing that it does no useful work. In such a case, varying the size of the upper limit of I won't affect the execution time of the loop; you can try using this to detect such optimization.

As another example, Gilbreath's use of the Sieve of Eratosthenes in a series of ***Byte*** *magazine benchmarks reportedly led some compiler manufacturers to use the same test themselves. Obviously, knowing the speed of your compiler is good, but judging it primarily by the speed it performs a specific task is not good. Especially if the desire to have the fastest-rated compiler causes optimization for the specific benchmark at the expense of better overall speed.*

The closest one can come to getting a good sense of a system's performance is to design a benchmark that uses the same tasks the system will normally be used to perform, or to run a large variety of different tests in order to determine where the "average" performance of the system may lie. Actually, a wide range of tests may be more useful in pinpointing weaknesses; if the figures obtained by testing floating point math, integer math, moving blocks of memory, disk access, and so on are comparable between two systems, but the video display of one system is obviously slower, then it becomes obvious that it's not necessary to invest in a "faster" system if the video display of the slower system can be upgraded. Such upgrades are easier on MS-DOS machines, where there is a large selection of video display cards. (However, the use of a peripheral display card may have some speed sacrifices over the use of a memory-mapped display like that of the Apple II.)

Thus I was wary of the implications last month when I listed the dhrystone results for the ***ZipGSX*** *versus the* ***TransWarp GS****; I was quite serious about the fact that the results should* ***not*** *be considered a significant difference unless all you ever plan to run on your computer is a dhrystone test (not a terribly useful application for the computer). Microcomputer pioneer Adam Osborne published assembly language benchmark programs for most microprocessors, but cautioned that a difference less than a factor of* ***ten*** *might well be inconsequential in many applications.*

Other considerations that outweigh the "judgment" of benchmarks are system features, reliability, and your own need for speed. It doesn't matter how fast your system runs if it doesn't do what you want it to, or if it is only working 10% of the time, or if it is waiting on you (rather than vice versa).

So what do benchmarks tell us? Benchmarks run on substantially similar hardware do give a rough idea of the relative speed of a system. A few percent difference may not matter, but an order of magnitude definitely does ***unless*** *the extra speed costs more and is unneeded in your primary application (buying a 33 MHz 80486 system to run a simple word processor is not economically wise).*

Or, as in your case, if you know that you are primarily working in a specific software environment such as interpreted BASIC, comparing that configuration on various systems is useful. From your timings, it doesn't look like buying an 80386 to run interpreted BASIC is that

much of a necessity unless other features of the environment (more BASIC commands, as an example) are desirable. If you run spreadsheet software 90% of the time you use your system, then timing a "representative" data set on the prominent spreadsheets on several different platforms is a valid way of comparing them. In this case, you're actually testing the combination of hardware and software to see what set synergistically works the best.

Obviously, a part of the environment that the individual user can't easily control is the operating system software. If you open a window under the Finder with IIgs System Software 4.0 and compare the speed to the same action under System 5.0, it's obvious that the system performance was substantially increased solely by finding and improving slow "subsystems" in the software.

It would be very handy to have "standard" benchmarks for the Apple II family so that system enhancements could be compared, such as the ***Norton SI*** *and* ***Landmark*** *indices for MS-DOS systems are used to compare different configurations. Such tests should be conducted as a "suite"; each member test focusing on a specific subsystem as much as possible, thus providing for an overall speed (for all tests) along with a "profiled" result for individual tasks. That would provide more diagnostic information about whether a speed limitation is the fault of the overall hardware or primarily located in a specific portion.—DJD*

New improved(?) drivers

I have recently upgraded my IIgs System Software to 5.0.3. You mentioned in the December 1990 issue that the 5.0.3 included a "much faster ImageWriter driver".

So far, I have discovered that my ImageWriter II now prints graphics much slower than before. (I am using Activision's *Draw Plus* on a *Sider* hard drive.) What is going on??

Mike Barr
Nacogdoches, Texas

...and so we launch directly into more testing, alas.

I had noticed that the text printing is much faster; so I picked a text document (about a third of a page of 10-point Courier type, laid out in condensed mode) and a graphics document (a single SHR screen, laid out in normal mode) and printed a sample of each using ***AppleWorks GS*** *under System 5.0.2 and 5.0.4. Rather than making this an 8-hour project, I elected to stick to testing the "Better text" mode of the 5.0.2 ImageWriter driver versus the Fast, Standard, and Best modes of the 5.0.4 drivers (5.0.3 timings were similar to those for 5.0.4). All times are in seconds:*

System/mode	Text	Graphics
5.0.2		
	Better text	111111
5.0.4		
	Fast	15/1536/43
	Standard	28/36151/172
	Best	50/50139/160

The great improvement in speed does appear to be for text printing. ***This is one series of tests with one application; assimilate with caution.***

The numbers don't tell the whole story; the quality of graphics printing with the 5.0.4 "Fast" mode is not as good as the "Better Text" mode of 5.0.2, but both the 5.0.4 "Standard" and "Best" mode seem to print better (darker, and with more resolution) which may offset the consideration that those modes print slightly slower. The "Fast" mode is ***much*** *faster; fast enough to actually encourage printing test copies before jumping to a better quality mode for the final printout.*

In some cases with 5.0.4, the IIgs finished printing and returned to use well before the printer was finished printing to paper; that's why there are two figures for 5.0.4 in the table. The first figure is the time for the IIgs, the second is for the printer.

5.0.3's drivers timed in about the same as those for 5.0.4, but there was reportedly a problem that could crop up in low-memory situations (not a problem for our tests; we used a 4 megabyte system). Apple didn't even get around to mailing 5.0.3 to its dealers before 5.0.4 was coming down the pipe; 5.0.4 should be at the dealers at this time.

If your program is printing slower than you think it should with the new system software, check to see if it uses the newer drivers (the new drivers display options using pop-up menus in the "Print..." dialog). Also, the drivers work faster if they have some excess memory available; if you are using a program that occupies almost all the memory on your IIgs, try adding memory or reducing the size of memory in use (by disabling any "extra" resident programs such as non-essential desk accessories)—DJD

Music and the IIgs

I have an Apple IIgs, Yamaha PSR-48 keyboard, hard disk, 2 megabytes of memory and Apple MIDI interface. I need tips on good programs for arranging four-voice (barbershop-style) music with lyrics. I have *MusicStudio v2.0* (that I can't get to work) and *Music Construction Set* (simple version).

Joe Lazar
Hopewell, N.J.

I've discovered two little tricks to using ***Music Studio v2.0****: you need to have its "APPLEMIDI" driver installed along with Apple's "APPLE.MIDI" driver (in your System/Drivers directory), and the WAVES subdirectory must be in the root directory of your hard disk. I consider both of these "features" to be poor planning, but sometimes you make concessions to get things to work.*

As far as doing composition and transcription above the hobbiest stage, the best alternatiuve I have seen is ***MusicWriter*** *(Pygraphics, P.O. Box 639, Grapevine, Texas, 76051, 800-222-7536), which is available in versions for the older Apple II systems and for the IIgs. There are three levels to the* ***MusicWriter*** *programs, which differ primarily in the number of parts (staves) that they support. All support MIDI input and output. The IIgs demo version we've played with (I emphasize that we haven't spent endless hours with it) did a very credible job of transcribing from a MIDI keyboard, keeping up with moderately fast and complicated sequences.—DJD*

GS/OS error codes

How about a listing of **all** the GS/OS error codes and what they mean? I know there are CDA/NDA's to give these errors but I try to keep my desk accessories to a minimum to avoid crashes. An AppleWorks data base file of these error messages would be a boon to all IIgs owners. Surely you must have accumulated some background on these cryptic messages.

Brian Sculley
Kitchener, Ont.

We keep our record of error codes scattered in the several pages of listings in the Apple IIgs references. It might be possible to come up with an abbreviated listing if we knew what the most "popular" errors were.

If you're looking for the codes in a text file, the ***APW Tools and Interfaces*** *from APDA includes a SYSERRS (Rez source) file that lists IIgs error codes. You could adapt this file for your own use.—DJD*

Cool it

I have a IIgs loaded with cards and a 20 megabyte Vulcan hard drive. It needed cooling, but the Applied Engineering fan isn't available here and in any case it's the wrong voltage. The Apple fan blocks two slots and there is no room for it.

So I built a wooden box that sits between the CPU and the monitor. It has no bottom, and I cut a round hole in the top that accomodates a $19 240-volt fan. I glued foam strips to the bottom to make a seal with the CPU and the fan draws air through the computer from the back and underneath, and exhausts under the monitor, which also helps to keep the monitor cool.

I can't find in any of the manuals just how much air is sufficient or if it's unwise to have too much air. The present fan draws 30 cubic feet a minute. I also wonder if this extraction method, if it's excessive, will cause interference with the Vulcan fan.

Testing the system with a probe, the case appears to hover around room temperature. The whole thing cost less than $25, but then it surely doesn't look like it came from Cupertino.

Richard Stephens
Toowoomba, Queensland

To our knowledge, internal fans don't match up well physically with the internal hard disks that replace the Apple IIe or IIgs power supply.

If you've got the internal temperature hovering around room temperature, that's a good sign unless you are drawing (and depositing) a lot of airborn dust into the computer and monitor, or unless the fan itself (or the air it is pushing) is vibrating internal components of the computer enough to damage them. We aren't engineers, so we aren't clear on how to gauge acceptable vibration levels; we work on the assumption that you shouldn't be able to feel any vibration from the fan being transferred to the computer.

Assuming the air is not rocking the internal parts, the other risk is that the volume of air passing through the system may draw more dust through the system; dust likes to attach itself to electronically charged surfaces and your computer is very attractive in this regard. Especially considering that you apparently have air flow to spare, it would probably be wise to add some type of a filtering system to the side of the fan used for air intake and plan on cleaning the filter occasionally. This assumes that you are drawing air through the fan and blowing it into the computer. Of course, if the fan is between the monitor and computer, one of those items is going to get the dust drawn through it; given a choice, we'd pick the

computer since the exposed voltages (to attract the dust) are lower and it is easier to open and clean the interior of the IIgs (color monitors and televisions can retain some nasty voltages internally, and it's not a good idea to poke about inside of them unless you know exactly what you are doing).—DJD

CD-ROM for Apple

I am interested in CD-ROMs for use with my Apple. Do you know of the availability of some? Most seem to require an IBM compatible system.

Robert H. Fetner
Lawrenceville, Ga.

We've examined a few MS-DOS CD-ROMs and many appear to be in the High Sierra format; GS/OS actually includes an FST (file system translator) for accessing High Sierra discs, and we've been able to look at some of the files using the IIgs with little or no problem.

In some cases, the data appears to be stored in proprietary but otherwise "decipherable" database formats. It's likely that HyperCard IIgs, with it's HyperTalk file access commands, could be used to develop application stacks to access the data. Though not as easy, given a High Sierra file reader for the IIe, it may be possible to also write applications to access the data from ProDOS 8. That's the good news.

The bad news is that some databases are apparently either condensed (compressed) or encrypted. This is reasonable given the intrinsic value of some data, but it does mean that development of an access program will hinge on the manufacturer's willingness to cooperate.

Our suggestion is that, if you see a CD-ROM you think is of interest, that you write the manufacturer explaining your interest and asking if it would be possible for them to support development of access software. If they have Mac software to access the data, it may be HyperCard based, which brings HyperCard IIgs to the fore. If they have only IBM software, then HyperCard IIgs may still be an option, or it may be necessary to try and "port" the MS-DOS access program to GS/OS.

The additional ability to provide ProDOS 8 support would be ideal, but this step may be made more difficult by the paucity of high-level languages (C or Pascal) for ProDOS 8.—DJD

A2-Central™

[redacted]

Publisher: **Tom Weishaar**

Editor: **Dennis Doms**

with help from:
Sally Dwyer **Dean Esmay** **Joyce Hammond**
Jay Jennings **Jeff Neuer** **Denise Shaffer**
Tom Vanderpool **Jean Weishaar**

A2-Central,—titled ***Open-Apple*** through January, 1989—has been published monthly since January 1985. World-wide prices (in U.S. dollars; airmail delivery included at no additional charge): $28 for 1 year; $54 for 2 years; $78 for 3 years. All back issues are currently available for $2 each; bound, indexed editions of our first four volumes are $14.95 each. Volumes end with the January issue; an index for the prior volume is included with the February issue.

The full text of each issue of ***A2-Central*** is available on 3.5 disks, along with a selection of the best new public domain and shareware files and programs, for $84 a year (newsletter and disk combined). Single disks are $10.

Please send all correspondence to:

A2-Central
P.O. Box 11250
Overland Park, Kansas 66207 U.S.A.

ISSN 0885-4017

Printed in the U.S.A.

GEnie mail: A2-CENTRAL
Voice: 913-469-6502
Fax: 913-469-6507

Roll out the Apple

I am president of an Apple user group in England; we call ourselves Midapple. Most of our member are Apple II users.

One of my members is keen to find anyone out there who may know anything about using an Apple IIe to produce—wait for it!—punched rolls for barrel organs. I believe that reproducing (player) pianos use a similar system. Perhaps one of your readers can help?

William Watson
Kingswinford, West Highlands
England

Odd characters

Regarding IIgs font control character printing as discussed in ***A2-Central*** in May 1990 (page 6.30, yeah...I'm slow):

To print the control characters $11, $12, $13, and $14 in the Chicago font, use *TimeOut SuperFont*'s non-existent "<X0>" (or "<X4>") command:

Chicago 12 Font:

<H1>	Q	R	S	T
<H0>	⌘	✓	◆	

Beverly Cadieux
Kingwood Micro Software
3103 Lake Stream Drive
Kingwood, Texas 77339

Monitoring problems

Do you have any recommendations as to a method of connecting the Apple IIc and IIe to an Apple IIgs RGB monitor?

It's an odd question, I know, but I find it hard to get answered. It's a query that comes up a lot in this area since the monitor here is very much cheaper than anywhere else, which is a first for the U.K. It's 220-240V only.

Huw Price
Bidmuthin Technologies, Ltd.

The only option we can think of for the IIe is a Video Overlay Card. In your case, you'd need the PAL version (AppleLink indicates it has been released through Apple Australia), but it would certainly not be an economical answer. Possibly a reader knows of alternatives.—DJD

Portable non-Apple

Tandy's WP-2 portable word processor is light (about one pound, inexpensive (about $300), and perfect for note-taking. It uses an 8-bit, Z80-type CPU, a 256K ROM to store built-in telecommunications and word processing programs (including spell-checker and thesaurus) and 32K RAM for files. It also includes parallel and serial ports.

Here's the problem: is there any way that can download an ASCII file from the WP-2 to an Apple IIc?

I've tried a direct cable connection and a null modem connection at many settings, yet nothing seems to work. As a stopgap, I use my Mac II at work to transfer files. There I have two phone lines and I can use the Tandy and modem to call the desktop Mac II and modem. Once the file is on the Mac, I can use the Apple File Exchange utility to make a file that's readible on my Apple IIc at home.

Isn't there a simpler way?

Bob Varetton
Bergenfield, N.J

Not having one of the WP-2's, we don't know the answer, but maybe somone else out there has gotten this combination to work.—DJD

Improved *AppleWriter*

I am editing long books. I currently have 16 books in progress, ranging in length from 25 to 200 pages. The editing is to condense them from outline form to paragraph form. Most of the editing consists of multiple simple "Find and Replace" command routines, to reduce the length to about half. The main routine I use for all editing contains 27 such "Find and Replace" commands.

I have AppleWorks and *UltraMacros*. They don't handle the job as easily, simply, and readily, nor are they as powerful a word processor as *AppleWriter* and WPL. For example *AppleWriter* allows storage of **multiple** command routines in a glossary that can be loaded into the word processor as needed. Also *AppleWriter* can use various wildcards and delimiters, etc.

I have an enhanced Apple IIe with a one meg *RamWorks III* card and will add more RAM as necessary. However, *AppleWriter* currently does not take advantage of RAM beyond 47K.

Has, is, or will anyone develop a patch for *AppleWriter* that will extend the desktop and take advantage of larger RAM cards? We desperately need such a patch.

John W. Pankey
St. Elmo, Ill

There is both hope and desperation here.

*As far as we know, there has been no contact with the author of **AppleWriter** (Paul Lutus) in the attempt to make any further improvements. Those we knew who were looking for him independently in the hope of "licensing" **AppleWriter** have failed to locate him. The program itself has been out of publication for many years, and those waiting for its re-emergence should realize that the chance of that is vanishingly small.*

*There are some **AppleWriter** hackers out there who continue to work with the program; every once in a while something surfaces. Recently, we discovered a shareware patch on GEnie that supposedly extends the memory reach of **AppleWriter**, but it does have what many AppleWriter users will consider a limitation: it only works on an Apple IIgs. If you are interested in finding out details about the patch, or in asking the author about the possibility of support for other types of extended RAM, try contacting Chester H. Page, 1707 Merrifields Drive, Silver Spring, MD 20906.—DJD*