$1.50

Washington Apple Pi

Volume 3 February 1981 Number 2

Highlights

THE SEARCH FOR THE ALMOST-PERFECT WORD PROCESSOR

MICRO COMPUTER AS AN INNOVATION: NOTES ON EDUCATION

BLAISE AWAY

In This Issue

OFFICERS & STAFF

President	-Bernard Urban	(301) 229-3458
Vice President	-Rich Wasserstrom	(703) 893-9147
Treasurer	-Bob Peck	(301) 468-2305
Secretary	-Dana Schwartz	(301) 621-2894
Members-at-Large	-Mark Crosby	(202) 488-1980
	-Sandy Greenfarb	(301) 674-5982
	-Hersch Pilloff	(301) 292-3100
Immed.Past Pres.	-John L. Moon	(301) 332-0945
Editor	-Bernard Urban	(above)
Associate Editor	-Rich Wasserstrom	(above)
	-Genevie Urban	(301) 229-3458
Librarians:		
Disk Mail Order	-David Morganstein	(301) 972-4263
Disk Pick-Up	-Bill Bowie	(301) 924-3455
Group Purchases	-Howard Lefkowitz	(301) 649-3373
SIG Chairmen:		
ASMSIG	-Jim Rose	(301) 730-2230
EDSIG	-Chuck Philipp	(301) 924-2354
NEWSIG	-Al Weiner	(301) 468-0663
PIG (Pascal)	-Tom Woteki	(202) 547-0984
SIGAMES	-Al Gass	(703) 371-3560

Washington Apple Pi
P. O. Box 34511
Washington, D.C. 20034
(301) 468-2305

ABBS (301) 983-9317

Membership dues for Washington Apple Pi are $18.00 per calendar year. If you would like to join, please call the club phone and leave your name and address, or write to the PO Box above. A membership application will be mailed to you.

EVENT QUEUE

Washington Apple Pi meets on the 4th Saturday of each month at 9:30 AM, at George Washington University, usually in Building C, on G Street at 23rd Street, NW. (To be sure of the exact location call the club phone or ABBS during the week of the meeting.) The February meeting will be on the 28th and the March meeting will also be on the 28th.

The Executive Board meets on the 2nd Wednesday evening of each month. All members are welcome to attend. Details will be on the club phone and ABBS, or call the President at 229-3458.

NOVAPPLE meets on the 2nd Saturday of the month at 1:00 PM at Kings Park Library on Burke Lake Road in Fairfax County; and on the 4th Thursday of the month at 7:30 PM at Computerland of Tysons Corner.

Mike Cornblith of Apple Computer Inc. will speak at the April 11 meeting of NOVAPPLE at 1:00 PM at the Burke Lake Library. He will answer questions from the floor on APPLE and its future. All members of NOVAPPLE and Washington Apple Pi are invited to attend this very special meeting.

There will be a special meeting of EDSIG on Saturday, March 14, at the University of Maryland. Details are given in the SIGNEWS column.

EDITORIAL

Some bad/good news. The bad, Steve Wozniak - the Woz - co-inventor of the APPLE, crashed his private plane about three weeks ago. Both he and his fiancee, Candy, suffered severe lacerations of the head. He also suffered a concussion. He had just been transferred to El Camino Hospital in Mountain View, California when I called him. He said that he had no recollection of the events of some two days prior to the crash, of flying or of the crash. He was quite concerned about Candy. He was overwhelmed by the concern and well wishes expressed by all the "APPLE people". He said that he plans to fly again. Good news, Candy has been released from the hospital. Don't know if she will need additional plastic surgery. He is still hospitalized for observation.

It's time to think about nominations for club officers. The slate should be ready by April and the actual election occurs in May, with the elected officers taking office in June. Washington Apple Pi continues to grow by leaps and bounds. It was only last month that we were wondering who would be WAP#500, and now we're already over 530. It becomes increasingly important for you to choose your officers carefully. It requires a major commitment on their part to keep the club viable and useful to you. Call John Moon with your suggestions, or leave a message for SYSOP on the ABBS.

We have a new column "Notices" which we started in the January issue. Various and sundry club news will be posted there.

CLASSIFIEDS

FOR SALE: LA-36 DecWriter printer/terminal. Includes accessory shelf, extra ribbons, paper cage, and a full set of service manuals. Price is $800 for the printer only, or $925 for the printer and the APPLE high-speed serial interface card. Also FOR SALE: Sanyo 9-inch monitor in original carton $140; APPLE disk controller card $40; Integer Basic ROM (without Programmer's Aid) card $75; Professional 19-inch black and white monitor, without cabinet $75. Please contact Steve Sondag, (703) 281-5392.

WANTED: Spare Pascal PROMs. Call Fred Schulz, (202) 223-1397

FOR SALE: APPLE Serial Interface Card with PROM for letter quality printer. David Moses, 270-1117.

FOR SALE: Mountain Hardware Supertalker 2, complete, $150. Call Bill Bowie, 924-3455, or WAP163 on the ABBS.

SIG-NEWS

SIGAMES is the special interest group of computer hobbyists interested in having fun with their APPLEs through some aspect of games. The main meeting of this group is held at a location announced at and following the Washington Apple Pi monthly meeting.

This month's meeting will continue construction of the joy sticks for the APPLE II's game I/O socket. Parts lists and sources of materials should be available this month. Group sessions will be held during the construction phase to insure that novices get the right things soldered together. Bill Bowie and Brian Dormer are co-chairing this project. James Hall will start the meeting off with a product review of REVERSAL by Hayden Book Company.

PIG, the Pascal Interest Group, is going strong! We meet the third Thursday of each month at 7:30PM at the Uniformed Services University of the Health Services, Bldg. A, Room A2054 (2nd floor), near the National Medical Center at 4301 Jones Bridge Road, Bethesda, MD.

Last month's meeting was led by Burt Chambers who presented an excellent discussion of library units - what they are, how to use them, etc. The month before, Bill Wurzel discussed his P-code disassembler. In February Paul Sand will lead the discussion.

In addition to meetings, we have produced almost two disks worth of Pascal programs, all source code. Included are the disassembler mentioned above and an earlier version of the unit presented in the newsletter. These disks will soon be available from the main club library.

EDSIG is having a special meeting on Saturday, March 14, 1981, at the University of Maryland.

Mary Jo Messenger, a high school math teacher, will demonstrate the various classroom uses of the following: a graphics tablet; a mark-sense card reader; and a Silentype printer. Mary Jo is now using APPLEs in her classroom and has had experience writing computer curriculum for secondary schools.

In addition, Dr. Henry Heikkinen will demonstrate the use of the APPLE as a remote terminal to access Micro-Net and electronic bulletin boards.

Nancy and Chuck Philipp will share, demonstrate, and describe a statistics program for analyzing multiple choice tests. The program is designed for an APPLE with a disk drive.

The meeting will be in the Eudcation Building, Room 0220 (in the basement) and will start at 9:30AM. Automobiles should be parked in Lot No. 1, located just west of the Education Building.

NEWSIG will meet just after the regular Washington Apple Pi meeting. Boris Levine will give a short talk on elementary programming in Basic on the APPLE. Later, questions regarding 13 vs. 16 sector disks, how to get the APPLE up and going, etc. will be answered.

The introduction to Washington Apple Pi will be held during the regular business meeting. This will be only for people who have never been to a WAP meeting before. Its purpose is to tell the new members about us, what we do, how to buy the library disks, what the SIGs are all about, etc.

Notices

THE CLUB PHONE IS MOVING

The club phone will be moving to the home of our Secretary, Dana Schwartz. Therefore, it will be out of service from February 28 to March 2. At this writing we do not have the new number, but it can be obtained by dialing the old number. Bob Peck has "manned" the club phone in the past, but now finds that he needs a break. Thanks to him for all his time spent on this service in the past.

UNIVERSITY OF MARYLAND PROGRAM LIBRARY

The Program Library of the University of Maryland Computer Science Center has started a reference collection of APPLE related documents and newsletters. The Library is located in Room 2337 and is open to the public during the following hours:

Monday - Thursday	9:00AM - 9:00PM
Friday	9:00AM - 6:30PM
Saturday	1:00PM - 5:00PM

There is a coin operated copy machine available to users.

WAP NUMBER 500!

Among the new members registering at last month's meeting was WAP#500. His name is David Moses, and as advertised in the January newsletter, we are awarding him ten free library disks. Congratulations, David!

MEMBERSHIP RENEWALS

If any of you have neglected to renew your dues for 1981, this is your last chance to do so without missing any copies of our newsletter. We will be purging our membership list, and this will be the last one you receive if you haven't renewed.

Questions, Questions ,Questions

by Mark L. Crosby

Q. I want to use the Applesoft CHR$ function to print in Integer BASIC. I have heard that this is possible but don't know how to do it. Can you help?

A. The following routine will give you a good start:

```
)LIST
  5 DIM A$(19)
 10 A$="WASHINGTON APPLE PI"
 20 POKE 1,165 :  POKE 2,0 :  POKE 3,76
    POKE 4,237 :  POKE 5,253
 30 CALL -936:  VTAB 10:  TAB 10
 40 POKE 50,63:  GOSUB 100
 50 VTAB 11:  TAB 10
 60 POKE 50,255:  GOSUB 100
 70 END
100 FOR I = 1 TO LEN(A$)
110 POKE 0, ASC(A$(I,I))
120 CALL 1
130 NEXT I
140 RETURN
```

The short machine language routine first loads the accumulator with the value in location zero (0) which is the ASCII value of the character to be printed. It then jumps to the Monitor Print routine at $FDED and returns to BASIC.

Q. I have an ANADEX DP-9501 printer which uses a 30 yard mobius cartridge ribbon. I notice that the nylon ribbon is extremely durable, and lasts much longer than the ink it contains. With the cartridge price on the order of $20 each, I am interested in re-inking the ribbon, rather than doing with a faint copy or expensive replacement. I have tried re-inking, using a Sanford roll-on ink, but the engineer from ANADEX has cautioned me that this may cause trouble with the print-head (the re-inked ribbon worked beautifully, incidentally). Can you tell me where I might find an appropriate dot-matrix substance (it's called a DYE, and contains a lubricant) for "re-inking"?

A. Manufacturers of printers generally have their ribbons inked by a professional office supply house that specializes in "matrix" inks. Because of volume, the supply houses generally produce the ribbons exclusively for OEM use and do not sell, for example, small bottles of re-inking ink. Some of them do sell their own brand of ribbons usually made to the same specifications as the OEMs but at reduced prices.

At the present time, I could not locate any supply houses that carried either the ANADEX or the IDS style of ribbons. I would recommend you "bug" some of them until they get the point - that there is a market for the ribbons. Now, about the Sanford roll-on - DON'T - YOU'LL BE SORRY and that's a fact. Matrix inks contain little or no solids (hence the term DYE) and usually have a lubricant in them. The matrix pins must be able to move freely in and out of their mounts. Any friction will slow them down and cause them to "freeze". When this happens the pins get caught in the nylon ribbon and bend. Both the lubricant and the lack of solids in commercial ribbons prevent this from happening. Your ribbon probably still has enough lubricant to keep it working for a short time but if you use it for any length of time you'll probably have problems.

Try contacting: A.M. Office Supplies, 7209 St. Clair Avenue, Cleveland, Ohio 44103 (216) 391-6300 or Uarco Incorporated, 121 North Ninth Street, DeKalb, Illinois 60115 (815) 756-8471.

Q. I'd like to find a nicely-readable treatment of 6502 machine-language, with plenty of specific examples. Can you make any recommendations?

A. Certainly:

PROGRAMMING & INTERFACING THE 6502 WITH EXPERIMENTS by Marvin L. DeJong. The Blackburg Continuing Education Series, Howard W. Sams & Co., Inc., 4300 West 62nd St., Indianapolis, Indiana 46168. 414 pages, paperbound. $13.95

This book is both for the novice and the knowledgeable 6502 user.

PROGRAMMING THE 6502 by Rodnay Zaks. Sybex, 2344 Sixth Street, Berkeley, California 94710, 1978 305 pages. $12.95. Also: 6502 APPLICATIONS BOOK and 6502 GAMES, each $12.95.

These three volumes are listed in order of complexity.

Also, I have found a series of articles called "Assembly Lines" by Roger Wagner to be extremely instructive. See Softalk starting with Volume 1 October 1980 through the February issue. His style is very readable with many examples.

Q. I would like to "partition" the Apple's screen to display menus, etc. in one location and take input while displaying it from another part of the screen. I need several on the screen at once. How can I do this?

A. Here is a program that will show you what can be done using the Text Window pointers. Briefly, there are 4 pointers that control the screen size or the "window" - Top, Bottom, Left, and Width (not "RIGHT" as you might expect). In the program, "R" represents width. When used with CALL-936 or HOME, only the window is cleared and the cursor HOMEd within that window regardless of where the cursor WAS. Try this:

```
]LIST
  5  TEXT :  HOME
 10  A$ = "WASHINGTON APPLE PI "
 20  T = INT ( RND (1) * 25)
 30  B = INT ( RND (1) * 25)
```

contd.

```
 40  L = INT ( RND (1) * 40)
 50  R = INT ( RND (1) * 25)
 60  IF L + R ) 39 OR L ) = R
     OR T ) = B THEN 20
 70  IF R - L ( 2 OR B - T ( 2 THEN 20
 80  POKE 34,T:  POKE 35,B:  POKE 33,R:
     POKE 32,L
 90  HOME
100  I = INT ( RND (1) * 2) + 1
110  ON I GOTO 120,130
120  INVERSE
130  FOR  I  =  1 TO 100:   PRINT  A$,:
     NEXT:  NORMAL
140  GOTO 20
```

If you adapt this technique in your own programs be careful to avoid exceeding a total L + R of 39. If you ignore this, you will lose both your program AND Applesoft. Also, the top of the screen cannot be set lower than the bottom without erratic results. VTAB is always absolute and is not "window-dependent". HTAB and TAB are relative to the left margin but often yield unreliable results if you exceed the width setting.

Q. Sometimes while sitting at my Apple II Plus, the screen goes crazy. Letters become numbers and they change positions. Some become inverse and others just disappear. Hitting the case with my fist (not too hard) sometimes cures this. Also it does not always boot when the power is turned on. Is there anything I can do short of taking it in for an overhaul?

A. This is most common during the Summer months when moisture in the air is highest. Corrosion is caused at the contacts of all the chips and each peripheral card. Without gold-plating each and every one of them, there is no permanent cure. A good one, and the proper one, is to remove power, remove each chip with a chip puller (buy one - it's cheap and avoids bending pins) and then replace each. Actually, you can just lift each one of them up a bit without actually removing them, then push them back in. Also remove each peripheral card and do the same to the chips. Take a "ruby" eraser and shine up each card's slot contacts too. Replace them all and you're done for at least 3-6 months in this locale.

Q. I am looking for a relatively cheap music board for the Apple II that has at least 4 voices and perhaps some envelope control. $200 is the tops I want to pay.

A. Try the APPLEDAC Hardware Music Board from Micro Technology Unlimited, 2806 Hillsborough Street, P.O. Box 12106, Raleigh, NC 27605 (919) 833-1458. Current price as of this writing is $89 for the board (K-1002-4), $49 for software to run it (K-1002-9D) and $20 for a disk song pack (K-1003-3D).

The board itself contains an 8 bit D/A converter, 6-pole low-pass filter and a half-watt audio amplifier with volume control. Cut-off is at 3.5kHz so don't expect super high frequencies. The software contains waveform tables and is capable of dynamic changes. Amplitude envelopes may be specified for each harmonic in the tones used. This permits good simulations of clarinet, banjos, guitar, and even bell-like tones.

MINUTES

EXECUTIVE BOARD MEETING

The Executive Board meeting of January 14, 1981 was called to order at 7:15 PM at the home of the President, with 12 persons in attendance.

The President displayed a letter which he had received from a distant newly-formed users' group requesting information about the WAP library and newsletter, and asked for guidance from the Board. Since WAP has no 'club' memberships, it was suggested that a member of the new group join WAP and purchase the disks. Additionally, since all the software is uncopyrighted and only the anthology itself is our property, the new group must remove references to WAP and must re-edit the collection of programs before redistributing to their members.

Motions were passed establishing that the newsletter would be sold in bulk to clubs outside the immediate area at the cover price, and establishing a policy for periodic publishing of ApNotes at cost plus a reasonable handling fee.

The Vice-President advised the Board on the status of the Club's attempt to become recognized as a non-profit organization. The Board established a 'tax account' containing sufficient funds to cover any tax liability which the club may be obligated to pay for 1980 and 1981. It was also passed that the club forstall any capital expenditures until our tax liability is ascertained or until the finance committee determines that funds are available. Outstanding bills will be paid, however. A resolution was then passed that the club tend to phase out hardware group purchase items.

The Membership Chairman announced that membership cards are ready, and that they will be typed and distributed as the 1981 member list becomes available. The need was voiced for a method of providing a 'hot-line' type question and answer service for the membership, although no acceptable solution was immediately found.

The meeting was adjourned at 10:00 PM.

GENERAL MEMBERSHIP MEETING

The Washington Apple Pi meeting of January 24, 1981 was called to order at 9:30 AM by the President with over 150 persons in attendance. The status of dues collection and membership cards was discussed, and a call for suggested candidates was made by the Nominations Committee headed by John Moon. The Group Purchases Chairman announced the Executive Board decision on reducing future hardware purchases. A new special interest group, SIG/Disabled, was revealed, to be headed by Curt Robbins.

A question and answer session was followed by the feature presentation on the Smarterm Board by Tom Woteki (Dr. Wo).

The meeting was adjourned to SIG meetings at 11:00 AM.

Dana J. Schwartz, Secretary

PASSING ARGUMENT VALUES TO MACHINE-LANGUAGE SUBROUTINES IN APPLESOFT

by C.K. Mesztenyi

Applesoft provides input argument values to a machine language subroutine by POKEing them into fixed memory locations prior to the CALL statement, and provides retrieval of output values of the subroutine by PEEKing into fixed locations after the CALL statement. I not only find these procedures awkward, but if these values are in the internal registers (A,X,Y) of the 6502 processor then I am forced to write an interface program in machine language to pick up or store these values. This becomes especially painful when I only want to test out one of the many 'useful' subroutines published which are in the Monitor, Applesoft, Interface Cards, etc. Hopefully, one of these days some of the entrypoints of DOS will also be published.

As a partial remedy to this problem, I wrote the following somewhat general interface program which sets up the internal registers prior to jumping into any given machine language program. The interface program uses Applesoft subroutines GETBYTC, FRMNUM, GETADR, CHKCOM and memory address LINUM as published by J. Crossley in the March/April 1980 Apple Orchard. The use of this interface program from Applesoft is by the statement:

```
CALL origin,A-expr,X-expr,Y-expr,location
```

where

origin = decimal address of the interface program entry

A-, X-, Y-expr = valid expressions which evaluate to single byte integers (0≤expr≤255) to be loaded into the registers A, X and Y, respectively.

location = decimal address of the machine language subroutine to be used with the above input values.

The interface program in assembly language form is as follows:

```
GETBYTC EQU $E6F5
FRMNUM  EQU $DD67
GETADR  EQU $E752
CHKCOM  EQU $DEBE
LINUM   EQU $0050
        ORG $origin
$origin JSR GETBYTC
        TXA
        PHA
        JSR GETBYTC
        TXA
        PHA
        JSR GETBYTC
        TXA
        PHA
        JSR CHKCOM
        JSR FRMNUM
        JSR GETADR
        PLA
        TAY
        PLA
        TAX
        PLA
        JMP (LINUM)
```

and in hexadecimal: (32 bytes)

```
$origin: 20 F5 E6 8A 48 20 F5 E6
         8A 48 20 F5 E6 8A 48 20
         BE DE 20 67 DD 20 52 E7
         68 A8 68 AA 68 6C 50 00
```

Example:

Assume that the above interface program is loaded into memory location $origin=$6000 (origin=24576) for this example. (Note that you may load it anywhere.) To use the Monitor subroutine PRNTAX ($F941) to output two bytes in hexadecimal, we have to load the registers A and X with these numbers (see Reference Manual p. 61). $F941 is equivalent to 63809 in decimal. Let the two bytes for output be the values of V1 and V2 of Applesoft variables. Now the following Applesoft statements will output those values:

```
V1 = ...
V2 = ...
CALL 24576,V1,V2,0,63809
```

The above statements can either be part of a program or also can be executed in immediate mode. Furthermore, V1 and/or V2 in the CALL statement can be replaced with the appropriate expressions shown with dots above it. Since the subroutine PRNTAX does not require input in register Y, the corresponding entry in the CALL statement contains a zero which is arbitrary.

THE SEARCH FOR THE ALMOST-PERFECT, LOW-COST APPLE WORD PROCESSOR

by Walton Francis

INTRODUCTION. One of the key purposes for which I bought my APPLE was word processing. Although I am reasonably satisfied with the system I got (EasyWriter used with Selectric), which I do not believe could then have been matched at the price, I plan to upgrade in the near future. This article is to share with you the information I have developed in the search for the almost-perfect, low-cost APPLE based word processor.

BACKGROUND. Word processors are computer systems devoted to the manipulation of the written word through electronic editing. Their advantage over the use of typewriters can be expressed in one word--easiness. But this understates their utility. A good word processor so greatly facilitates composition of prose that it transforms the ability of an individual writer to perfect his writing.

Word processing is a revolution in office and clerical functions which will, perhaps, exceed in economic importance any other computer application within another decade. Yet it is among the least understood and most jargon-ridden and confusing computer applications. It is, strangely, perhaps as difficult to understand for the computer pro as for the lay public. This difficulty arises because, unlike actual computing of numbers, the use of a serious word processor cannot be conveyed adequately by reference to other experiences or to the written word (and certainly not by the arcane vocabulary used in most word processing literature). It is one of those subjects which must be experienced to be understood. So, my first recommendation to the APPLE owner interested in word processing is to try it--not any old word processor, but a serious, professional system using a full-screen display and whose most common editing commands can be learned in fifteen minutes or less. You are unlikely to see such a system in your local computer store (with the exception of a Z-80 machine using WordStar), so find a friend whose office uses a Vydek, Lexitron, or other top-flight professional system WHICH DISPLAYS THE TEXT EXACTLY AS IT WILL APPEAR ON THE PRINTED PAGE.

Word processors of like price are often of radically different quality. The difference between a bad and good word processor is roughly the difference between an old hand-cranked phonograph and a top stereo system, or between computer programming by flipping switches on a screenless board versus using an APPLE with a TV screen. Either will get the basic job done--if you are willing to pay the price in aggravation and lower quality. Some word processors for sale today are rather more difficult to learn than programming in a brand new language, and almost as difficult to use in composing and editing text as Applesoft Basic itself. But such a word processor entirely misses the point--the occasional user might as well stick with a typewriter and erasible bond paper, plan to type each page at least twice, and save a lot of money and trouble.

SYSTEM CONFIGURATION. A professional word processing system has five hardware components and a software program on disk (or sometimes ROM resident). The hardware includes a computer, at least one disk drive (or, rarely, a tape drive), a screen, a keyboard, and a printer. Most APPLE owners have all the hardware (though rarely the right kind of screen or printer), and an adequate system can be completed by investing $50 to $200 in any of a dozen or more APPLE based word processing programs widely advertised and sold. Such a system will involve a total cost of perhaps $3000, and seems to have most of the features of a professional system costing $12,000 or more. But the typical APPLE owner who takes this approach will have missed the boat. His system will be a Model T word processor, with some components (the APPLE itself) used far below their capability, other components (the TV screen and the printer) incapable of high quality word processing, and the software itself nowhere near as easy to use as the best now available. And the sad part is that for an extra $200 to $1000 our hypothetical APPLE owner could have had a system which bettered some professional systems and nearly equalled most. (Sorry, but as explained below, an APPLE II based system can never equal the best commercial systems, primarily because of keyboard limitations.)

QUALITY AND PERFORMANCE CRITERIA. The word processing articles you read rarely discuss REAL word processing performance. Perhaps this is because both the systems themselves and the people who write about them are primarily computer-oriented. The average word processing article focuses on counting "features"--seeming dimensions of quality which are, in fact, irrelevant or at most a small part of the truth. For example, it is hard to find an ad or an article about software which does not mention the "global search and replace" feature. The idea is to be able to substitute, at the time of final editing, one word such as "organization" for another such as "institution", in each place in a long document in which the latter appears. Whole paragraphs are devoted to the nuances of this feature, such as whether or not the program will handle words which are capitalized, or have different endings. But in five years of producing or reviewing thousands of documents ranging from short memos to 100 page research reports, in an office which has a professional staff of over 100 people and owns three dozen professional word processors which have this feature, I have never heard of search and replace being used, or even seen a case in which it would have been more than marginally useful. As another example, right

contd.

justification in printing, while more "professional" in appearance, is only needed in a few situations, such as magazine articles printed directly from author-submitted copy, and even then is a marginal feature.

So what does matter? In no particular order, but all important to almost all users except those who don't know what they are missing:

- Print quality--printed letters of office typewriter quality, on correspondence quality paper, without typographical errors and in pleasing type size and text format.
- System speed--to be good, a printer of reasonable speed (preferably either blinding speed or with ability to edit one page while another is being printed). Also, speedy storage and retrieval of text as you move from one function to another (a problem on systems which require frequent disk use while in operation).
- Ease of learning common editing functions such as inserting or deleting characters, setting margins, etc. This means, to be good you can sit down at the keyboard after skimming the manual, and be editing and printing documents within a half hour or less with the system, as compared to the days of study and practice required by some software just to produce a simple document.
- Ease and speed of performing common editing functions in actual use--the criterion most often failed by the systems which use quasi programming languages.

There are many other features which are of importance to some users, including particular capabilities such as automatic page numbering; reliabity of hardware (no business which seriously relies on word processing in day to day operations should ever own less than two machines and a same-day service contract); programmability for specialized uses such as form letters; and text storage capacity in both RAM and disk. (This is no problem for a 48K APPLE and standard disk drive, the former allowing storage of more pages of text than a prudent person would put at risk at one sitting, and the latter holding some 50 pages of text.) Most of these features are far less often needed than software vendors would have you believe, particularly if you have a system which displays the text on the screen exactly as it will appear on the printed page, and/or learn to use your system (in conjunction with the disk drive) to create one document from a prior document which differs only in part.

EASE OF LEARNING AND USE. In my experience, any machine which is easy to use is almost equally easy to learn, so I will combine here these two quality dimensions. Word processing requires telling the computer three things either in typing the basic text or in making changes after a draft is on the screen--where to enter, what action to take (e.g., insert or delete a character or word or phrase), and in what format to print out the completed text. There are two basic approaches to performing these functions--what I will call the "programming" approach and the "visual" approach. The programming approach requires you to instruct the computer by commands, i.e. by typing in instructions such as "indent each paragraph five spaces" or "insert word Q at coordinates X,Y". (In practice, of course, the program uses special key commands to provide such instructions, so you "only" have to memorize a vocabulary of perhaps 100 keyboard commands and a set of rules as to how and when you can use them in various combinations--a mini programming language). The visual approach relies on moving the cursor to the spot on the draft page, as displayed on the screen, on which you want the entry made, and then typing it in. You then go on to the next entry, and when done simply press a key to tell the computer to print the finished text AS IT APPEARS ON THE SCREEN. The programming is still there, but unobtrusively handled by the computer rather than you. For example, the visual approach at its best enables you to avoid using programming commands to embed a paragraph indentation command in the text, since the text as typed puts the beginning of each paragraph exactly where you want it. As another example, instead of giving the computer tab commands on where to locate each heading and entry on a complex table, with a separate command for each entry followed by typing the entry, you simply type the table entries formatted as you want them to appear. Thus, in theory the visual approach enables you to do entirely without print format commands, since the computer automatically prints the text as it is displayed on the screen. Perhaps equally importantly, the visual format greatly facilitates editing, since it is far easier either to proof directly on the screen without a paper copy first, or to edit on a paper copy and then find the place you want to change on the screen.

It is almost impossible to convey in words the difference between the two approaches in terms of ease of use. Those of you who own VisiCalc, which uses the visual approach to mathematical computations and table formatting, will understand this. Suffice it to say that the visual approach is so much simpler, so much more "user friendly", that without exception every person I know who has used it (from secretaries to top professionals) would never consider an alternative. Put most simply, however, the difference is between making changes directly without the intrusion of the computer versus having to enter multiple commands to tell the computer to make the change--roughly the difference between driving a car by punching in coordinates (if you can imagine such a thing) versus turning the steering wheel to tell the car where to go.

In practice, of course, no system is purely one way or another. There is, for example, no simple way to move paragraphs from one page to another without a "move" command. And most of the systems for sale today seem to be at least 50-50 in orientation, by using such features as cursor movement to any point in screen displayed text and "wrap around" automatic carriage returns to allow continuous entry of changes. Beware, however, of any

contd.

program which is a "line" rather than "character" editor--which will not allow you to change individual letters without retyping a whole line. Nonetheless, the degree to which the system has been designed to minimize the need for programming entries varies considerably from system to system.

Lest this all seem excessively abstract, let me put it in concrete terms. A fully visually oriented, professional word processor uses a screen with at least 80 characters of width and at least 60 lines (that is, one full typewritten page can be displayed on the screen), and upper and lower case letters selected by use of the keyboard shift key. Cursor control is usually handled by four arrow keys (right, left, up, down), and there are individual special function keys for such instructions as character and line insert and delete, so that all of the most common editorial functions are handled by one or at most two command key strokes. While there will be "embedded" programming type commands (typically shown on the screen but not printed), such commands will not be necessary for ordinary writing. Thus, the margin, the page length, double spacing, etc. will be set by where you place the cursor, and will not require a "menu" approach to setting format parameters, unless you specifically decide to reformat so much of a document at once that cursor driven changes would be tedious (in which case programming type commands can be used to overrule the parameters set by your original use of the cursor).

PRINTING QUALITY AND SYSTEM SPEED. At present, the almost universally used office printers are the letter quality, daisy wheel printers, such as Qume and Diablo. The NEC Spinwriter is a close cousin. These printers cost about $3000, considerably more than an APPLE with disk drive. In professional systems, these printers are driven through large memory buffers, so that the typist can edit one page while another is being printed, rather than sit and twiddle his thumbs. Even with such "spooling", they run at about the minimum speed tolerable for extended use, some 50 or so characters per second (that is, about 600 words a minute or one single spaced page each 1 1/2 minutes). There are several letter quality printers available for less. For example, the Vista sells for about $2200 in the 45 CPS version (don't consider the slower version). These lower priced letter quality printers are suspect in terms of service availability, and I have seen no feedback on their performance, but they certainly appear to be bargains in comparison to the others.

My printer, also letter quality, is the Anderson Jacobson Selectric, which costs around $1100. This is an option which I would not generally recommend, because the Selectric based printers have two major problems. First, they are intolerably slow at 15 CPS (200 words per minute, or one half hour to type a draft of this article). Second, they are inherently finicky, being based on a mechanism not intended for computer use, and have the nasty habit of creating random typographical errors if not in perfect adjustment. The AJ, however, at least has excellent local service available.

The dot matrix machines have very nice speeds, often of 100 CPS or more. Most of the brands available for under $1000 use 5 by 7 matrices, which do not allow true lower case descenders and hence even a pretense at letter quality printing. However, in the last few months there appear to have been considerable improvements. Several new machines, such as the 9 by 9 matrix Epson MX-80 reviewed in the December, 1980 Apple Pi, cost well below $1000 and offer near letter quality acceptable in most applications. The Epson achieves its high quality not only by a larger matrix but also by using multi-strike or multi-pass techniques which, however, halve or quarter overall speed from 80 CPS to 40 or 20 CPS for the final draft of a document. An equally high quality matrix printer is the Centronics 737, which retails for about $1000. These machines (and the IDS 460) have the major defect that they will not take paper wider than 8 1/2 inches, a significant problem if your word processing needs require large tables. Most matrix printers will print up to 132 columns, but only at the price of tiny typefaces squeezed onto standard paper. A second major defect is that most take only pinfeed paper, which must be hand trimmed or purchased with peel off backing to look like bond. The Centronics, however, will take regular bond paper as well as pin feed.

I would guess that within a year we will have available almost letter perfect quality for under $1000. The new IDS 460 Paper Tiger, selling for about $1300 list but locally available from Mesa for about $1100 including cable, is my best evidence for this belief (see review in November, 1980 Apple Pi). The Paper Tiger uses a 24 by 9 matrix to give virtual letter quality and has a speed of 150 CPS. The brand new IDS 560 is, I believe, otherwise identical but takes wider paper. It is sold by Mesa for about $1300 including cable. The Paper Tigers only have a token buffer (2K) even in the graphics option, not enough for real spooling, and their best type face is very small, but even so they are virtually office quality machines.

A useful descriptive article covering all types of printers, the specifications of many machines, and a number of good buying hints, can be found in the December/January 1981 issue of SMALL BUSINESS COMPUTERS. This article covers a number of machines which are not commonly advertised in microcomputer magazines. One caveat about this article is that some of the data it presents are wrong--the new Diablo 630 daisy wheel lists for $2800, not $900.

LOW COST, ACCEPTABLE QUALITY APPLE BASED SYSTEMS. From the above, it is obvious that the APPLE II cannot quite equal the best professional systems, simply because it does not have an 80 by 60 screen (and many professional systems have a screen capability of 132 by 80), nor a keyboard

contd.

optimized for word processing by four arrow keys for cursor control and a half dozen or more special function keys. But in every other respect the APPLE can potentially match these systems, and its inherent flaws are only marginal. Purchase of an 80 column card gives it a full width, half page screen, which is acceptable visual orientation (width is the key), and the single extra key stroke required to make normal keys do double duty for special functions is, based on my experience, within the bounds of reasonable ease of use.

What then, can be achieved with a total budget of about $4000? First, in addition to an APPLE II with single disk drive ($2000), an 80 column card ($350) is a necessity for real visual orientation. This, in turn, requires a black and white TV monitor ($250) rather than standard set, because a standard set simply has insufficient resolution to support 80 columns of upper and lower case letters. I have just purchased a low-priced Leedex monitor which seems to be adequate for my needs.

Second, if you plan to use word processing for correspondence, you need a virtual letter quality printer. The best printers of which I am aware which can fit within such a budget are the matrix printers discussed above, varying in cost from about $900 to $1500 including parallel printer card. The Epson looks like the best buy, though I am not sure about service. And $250 will buy what is apparently the ONLY fully visual word processor currently available for the APPLE, Information Unlimited's EasyWriter Professional System (see "Word Processing Software Roundup" in January, 1981 PERSONAL COMPUTING). Until recently, it was doubtful whether EasyWriter or any other available software would work with ANY 80 column board (see "The Serious APPLE" in 1980 Number 8 NIBBLE. This article, incidentally, offers much useful advice on APPLE peripherals generally). However, the Easywriter people insist that the Professional system works with the three major 80 column cards, and I have used the Professional system with the Videx card and find the combination excellent. The new APPLE Smarterm card, however, does not work with any word processor, though it is hard to believe that APPLE will not have a new version of Applewriter for it within a few months.

Another option is to buy the Z-80 Softcard from Microsoft, and get WordStar (which is so sure of the superiority of the visual approach that the ad simply states, printed on a screen, "What you see is what you get"). But this involves an extra $300 outlay which is hard to justify unless you want the CP/M system for other reasons.

In sum, it is still touch and go, but at worst very near in the future, before the APPLE will be a near match for the best commercial systems in routine word processing performance. Since the price will be right (either one-third the going commercial rate with a matrix printer or one-half with a daisy wheel), the APPLE based system will be a clear best buy.

For a big drop in ease of use, the $3500 system is here now. Accepting a 40 column format and therefore an inherently limited ability to use the visual approach, I would recommend without hesitation that anyone who needs word processing buy the regular EasyWriter software and use it with the Epson or Centronics printer (or the Paper Tiger if you want to do graphics as well). Numerous reviewers have been quite positive about EasyWriter's ease of learning and use, and it is apparently almost identical in design to the popular Z-80 system, Electric Pencil. I use EasyWriter and find it difficult to believe that any non-fully visual system could beat it in flexibility and ease for normal writing. Its major defect (corrected in the Professional version) is that it is not very handy for table generation and other complex formatting problems. Also, EasyWriter is written in the Forth language and cannot be used to edit Basic and Pascal programs or data files. Of course, other systems with which I am not familiar may match if not beat it, and may have a particular feature vital to your use, but subject to this caveat I see no reason to shop around for a better piece of software. Programma's Apple PIE, for example, apparently requires learning over 150 commands, and involves some other inconveniences in use. And I have read that the Applewriter won't wrap words around. However, both these systems cost under $100 and may well be best buys. Magic Wand and Super Text II are highly recommended by many. The latter, however, is apparently both more complex and harder to use than EasyWriter. For example, to shift into upper case you must embed a control character, compared to EasyWriter's use of the escape key as a shift key, and you must shift modes to change from creating to editing text. I should also caution you that Super Text's screen preview mode is in no way the equivalent of a true visual system, since it neither helps edit nor avoids the need to use print format commands. Nonetheless, so far as I can tell without actual use, these are all reasonable systems and it would be hard to go wrong with any of them if you can't see your way to spending the extra $600 for 80 column card and monitor.

One final option for the occasional user who already has the language card is to buy an 80 column card and use Pascal as a text editor. Pascal is not near optimum for word processing in its editing commands, but I understand that it does have the big plus of visual orientation, so you can print text as it appears on the screen.

Good searching.

THE MICRO COMPUTER AS AN INNOVATION: NOTES ON EDUCATION

by Charles C. Philipp

How should microcomputers be used in K-12 classrooms? This is a legitimate question and deserves some exploratory thinking about education and technological innovations in general.

There is a tendency for people to think about new things in old ways. In the minds of most people, the first automobiles were so closely linked to an old mode of transportation that engines were, and still are, described on the basis of what they replaced, horsepower. Perhaps this kind of thinking is helpful because it creates a connection with the past and provides for continuity. People can temporarily convince themselves that a new technology is really not "new" because it serves an old, familiar purpose. There is a danger in this way of thinking. The danger has to do with the unrecognized qualitative differences that are hidden beneath the surface of the initial uses of an innovation. The first automobile, like the horse, carried at most two people. Because it was possible to make automobiles bigger and bigger, there was an inclination to do so. Until recently, computer technology was tenaciously following the same path, and it was in this climate that the microcomputer appeared.

Large main-frame computers have been used by public school systems for many years. They have served a useful purpose and have saved money in areas such as payroll, accounting and inventory. These kinds of applications have produced a mind-set that is evident in the technocratic language used to describe big computer operations. Terminology such as data processing, thru-put, and mass storage reflect how people think about big computers. Indeed, the capacity of large computers to manipulate, store, and retrieve information is a modern marvel. Most school system people who know about computers know about this kind of computing.

It is easy to understand, then, why the first classroom use of computers involved a data-processing way of thinking about student learning. If big computers can be used to control a complex accounting or payroll system, why not use them to control student learning? This kind of thinking led to the development of computer-assisted instruction (CAI) and to its younger sibling, computer-managed instruction (CMI). Both of these applications represent the use of a new technology to serve an old, familiar function. This made it possible for school systems to be modern - the presence of the big computer proves it - and still feel assured that the old way was, and still is, the right way to educate children. The computer only helps to do it better and faster.

In this fashion, the present is linked with the past, and people do not have to think about subsurface implications or ask fundamental questions about the appropriateness of what is being learned or the quality of the learning environment. The past justifies the present. Advocates for using computers in this way were not difficult to find. B. F. Skinner, the Harvard University trainer of white mice, rats, and pigeons, provided a theoretical basis for CAI. Computer managed instruction was supported by school administrators and supervisors because it provides a way to gather and store extensive, detailed information about individual students, groups of students, and even produces some data about teacher effectiveness. Thus, big computer technology, in the form of CAI and CMI, has provided a way to refocus on the past. The things that some educators would have liked to do twenty years ago are now possible. This raises several important questions: Is it now desirable to do these things? Are the old ways of educating children worth the time, money and effort? Are there more important things, or perhaps ways, to learn today? Most educators, particularly those who run public school systems, are not asking these kinds of questions.

As it turns out, educators do not have to ask critical questions. The questions are being suggested in a powerful way by the presence of microcomputers and by what a few children have clearly demonstrated they can do with small machines. It is apparent that small children and small computers get along together very well. There is now much evidence that this rapport is possible because microcomputers are qualitatively different from large computers. However, rather than looking at the quality and nature of the interaction between children and microcomputers, educators, publishers, and others have focused their attention on the hardware only. What they see is not a microcomputer but a small "main-frame" computer. This illusion is leading to a replay of history. Educators are now trying to make microcomputers do what they have wanted large computers to do. School systems and publishers are now busy producing CAI and CMI software packages for microcomputers. While all of this has been taking place, professional programmers and computer scientists have, understandably, pursued their own interests. For example, those individuals who specialize in microcomputer technology and are interested in graphics, have a legitimate basis to investigate how to generate and how to use the graphics potential of microcomputers. However, when educators see an excellent graphics demonstration, they tend to associate the microcomputer with picture-making - something akin to film strip machines and movie projectors. Most individuals go a step further and silently establish a graphics criterion for judging microcomputers. This is not bad but it could have a Svengali-like effect.

It is not bad because graphics capability does play an important role in the

contd.

educational use of microcomputers. After all, children are attracted to colorful displays, and moving things do capture attention. In addition, graphics capabilities have contributed much toward making the microcomputer a "friendly" machine. In short, microcomputers ought to have good graphics capabilities. However, from the standpoint of educational use, all of this can be seductive. It is seductive because it draws attention away from the purpose behind the graphics. Some animated computer games are just games, while others have great potential to educate and promote intellectual growth. On what basis do educators and parents recognize the difference?

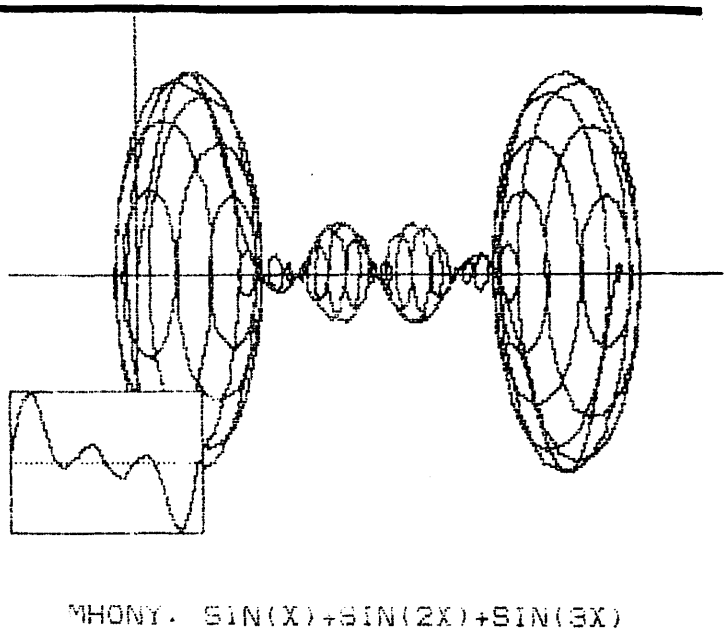
The question is really broader than just recognizing a difference. In fact, before any meaningful questions can be asked, new quality standards must be investigated within the context of the microcomputer's unique potential. Emphasis should be placed on identifying and analyzing the kinds of thinking that take place when children use microcomputers. Most of the assumptions that educators make about how children learn are based on beliefs about how children ought to learn. The growth in language use from birth to the time that a child enters school is evidence of outstanding learning ability. The primary influence during this period is the child's environment. How do children learn to speak their native language without a teacher planning lessons and writing instructional objectives? By telling children what to learn and when to learn it, teachers are also telling children that there are things they cannot learn. Given a chance, children and microcomputers will continue to challenge this point of view.

There is something inherently different about microcomputers, and it is that new and different concept or potential that must be brought to the surface and carefully studied in the search for new quality standards for the use of these machines in education. This task will not be easy because it overlaps three distinctly different disciplines: curriculum and instruction; subject matter content areas such as English, math and science; and the language potential and logic of microcomputer technology.

Today, no reasonable person would suggest spending large sums of money to develop a better horse shoe. However, comparable kinds of things are now happpening in education. When the first Model T rolled off the assembly line, the overall pace of change was slow. At that time, there was relatively little danger in enjoying the luxury of making innovations reflect images from the past. Today, the American automobile industry is a victim of the history it helped to make and refused to relinquish. As serious as this problem is, it is small compared to what will happen in the next century if today's youth cannot cope with the demands of a new and different society. The recent Time Magazine cover-story on robotics in industry is an indicator of things to come.

Thus, questions that have to do with how to use microcomputers in the classroom are not trivial. At this time, the microcomputer is an innovation without a context, and this helps explain why most educators see it primarily as a new tool for solving old problems. This article has suggested that the appearance of the microcomputer has presented some new questions - questions which have not yet been recognized. In a way, teachers are architects helping to build the future. How will quality be defined in the future?

NOTE: Three more articles are planned in this series. The next one will focus on curriculum and instruction and emphasize the different points of view that lead to different uses of computer technology in K-12 classrooms.



ADVERTISING RATES

The following table shows our new advertising rates. Our newsletter distribution is about 1200 copies now, with about 200 of these going around the country. If you would like to advertise please send camera ready ad copy (black and white only, no halftones) by the 10th of the month to our PO Box.

	RATES			
	FULL PAGE	HALF PAGE	QTR PAGE	8TH PAGE
Single issue (or kited series)	$ 40	$ 25	$ 15	$ 8
3-months series (or more)	35	20	10	5
Full-year contract	30	15	10	5

MICROCOMPUTER INFORMATION FOR EDUCATORS

by Robert E. Chesley

The Educations Resources Information Center (ERIC) has information available concerning the use of microcomputers in schools and classrooms. A series of short bibliographies is available concerning microcomputer hardware, microcomputer issues and trends, general microcomputer applications, microcomputer software applications, and microcomputer applications for elementary, middle, secondary and higher education. These bibliographies are available free of charge from the ERIC Clearinghouse on Educational Resources, Syracuse University School of Education, Syracuse, New York 13210.

Another ERIC publication prepared in cooperation with the Association for Educational Communications and Technology contains information about microcomputers for educators. It is entitled "Guide to Microcomputers" by Franz J. Frederick and can be used as a reference by educators without computer experience. It starts with very basic facts about microcomputers, discusses programming, accessories, maintenance, computer assisted instruction, educational applications and projects, as well as other printed resources available to the educator. The publication is available as a paperback publication from the above address for $11.50. It is also available on microfiche for $0.91 plus $0.15 postage from The ERIC Document Reproduction Service, P. O. Box 190, Arlington, Virginia 22210. Persons ordering should specify ERIC Document #192 818.

EPSON MX-80 HORIZONTAL TAB PROBLEM

by Mike Kramer

Those of you who have one of the new Epson MX-80 printers may have experienced some difficulty making it do horizontal tabs as described in the manual. The technique is similar to setting tabs on a typewriter. The following example illustrates how this is supposed to be done:

```
100 PRINT CHR$(27)"E"CHR$(10)CHR$(20)
          CHR$(0)
110 PRINT "ABC"CHR$(9)"DEF"CHR$(9)"GHI"
```

Line 100 specifies that columns 10 and 20 are supposed to be the tab positions. Line 110 should result in printing "ABC" in column 1, "DEF" in column 10, and "GHI" in column 20. Those who have tried this find that "ABCDEF" is printed starting in column 1, and "GHI" is printed in column 10. Through experimentation, it was found that by following the tab specification with printing of CHR$(9), the tab character, and a space, the tabs would work properly. Subsequent tabbing, however, was unpredictable.

A better tabbing method giving absolutely predictable results is POKEing the tab position into memory location 36. The following line prints the same result as the lines above:

```
90 PRINT "ABC";:POKE 36,9:PRINT "DEF";:
        POKE 36,19:PRINT "GHI"
```

EDITOR'S NOTE: The above article and the following letter were received this month from Mike Kramer, Vice President of the Houston Area Apple Users Group (HAAUG). Many thanks, Mike.

Gentlemen:

I read Bill Wurzel's recent article on the Epson MX-80 with interest since I've been the proud owner of one of those fine machines for three months. I have enclosed a copy of a brief article recently submitted to the HAAUG (Houston) Applebarrel. Bill's article briefly mentioned horizontal tabbing, but did not mention any problem. Any of your members trying to use the Epson for printing tabular data past column 40 might benefit from my experience. Bill also mentioned the graphics and Katakana character sets, but did not mention that they can be used only if the Epson parallel card is modified (assuming that card is used). I don't have direct experience using these characters, but Jay Ebner, one of our members, has modified his card and is working on a machine language graphics dump program which we hope to publish in Applebarrel soon.

Your members may also be interested to know that there is a comprehensive manual available from Epson entitled "User's Manual For MX-80 Printer" by David A. Lien. Publisher is Compusoft Publishing, P.O. Box 19669, San Diego, 92119. My copy was obtained at no cost from Epson, so I don't know how much they are. Although written mainly for TRS-80'S, there is a section on the APPLE as well as a lot of general information.

I've enclosed a check for $18 for membership in your group. I'd appreciate reeiving the necessary application forms.

(Ed. The manual mentioned above can be obtained by sending $2.00 to Parsons Pilchard, 1640 Evers Drive, McLean, VA 22101.)

PROBLEM WITH YOUR GAME PADDLES?

by Jay H. Feinstein

If you are having a problem with your game paddle, It may only be a dirty potentiometer. If your game paddle refuses to give you full reading (255), open the controller and spray the contacts (inside the potentiometer) with electric silicone contact cleaner while moving the control knob. Be sure to turn off the computer first.

PSEUDO DATA STATEMENTS FOR INTEGER BASIC

by Jim Kelly

I have read more than one programming article in which the author stated that he had to write a program in Applesoft because of the lack of DATA statements in Integer Basic. The implication is that it is much more difficult to write programs without the DATA statement. This is not the case. There is a relatively simple way to simulate a DATA statement in Integer Basic. The method is based on the fact that Integer Basic allows expressions in GOTO and GOSUB commands. As usual, Integer has an offsetting feature to make up for an apparent lack in relation to Applesoft.

Briefly, the pseudo-DATA statement involves putting the data into a series of short subroutines in successive line numbers after some index line. The accompanying pseudo-READ statement is a GOSUB whose argument is an expression referenced to the index line. Let's look at a simple telephone number look-up program that might be used as a segment of an auto-dial process. Listing 1 shows the more or less standard method of listing the stored phone numbers and allowing the user to select one. The FOR-NEXT loop in lines 50-80 prints out the numbers after reading them using the READ statement. The phone numbers are stored in DATA statements in lines 210-250. Listing 2 accomplishes the same task in Integer Basic. Most lines in the Integer program perform the same or similar function as the same numbered lines in the Applesoft version. The DIM statement in line 40 performs a different function because of the different nature of the string variables in the two Basics. The pseudo-DATA statements are in lines 201-205. Note the lines are numbered consecutively and not in multiples of ten. This simplifies the expression used in the pseudo-READ command. The name and phone number are assigned in each one-line subroutine of the pseduo-DATA statements.

Run the two programs. You will find that the results are exactly the same. The two listings are the same length and of almost equal complexity.

If the FOR-NEXT loop were filling a numerical array, the assignments could be made to temporary variables in the pseudo-DATA subroutines and then reassigned to the desired array elements in the FOR-NEXT loop.

Listings 3 and 4 illustrate a use of the pseudo-DATA statement which I have used for a Hangman game. The Integer program is longer than the Applesoft one since the latter allows all the data to be entered on a single line. The pseudo-DATA method requires many assignments and RETURN statements. However, the Integer version is much more efficient. If the list were 100 words long for instance, and the random choice were the 100th, the Applesoft version must loop through 99 READ statements before arriving at the chosen word. The Integer version goes directly to the correct word.

So you see, you can do anything a READ-DATA combination can do in Integer Basic, albeit with a few more key strokes.

```
] 5 REM LISTING 1
 10 REM PHONE LIST - APPLESOFT
 20 HOME :  PRINT :  PRINT
 30 READ N
 40 DIM NM$(N),PH$(N)
 50 FOR I = 1 TO N
 60 READ NM$(I),PH$(I)
 65 PRINT I;:  HTAB 3
 70 PRINT NM$(I);:  HTAB 20:  PRINT PH$(I)
 80 NEXT I
 90 PRINT
100 INPUT "CHOOSE BY NUMBER ";A
110 PH$ = PH$(A)
120 PRINT
130 PRINT "THE PHONE NO. IS ";PH$
140 END
200 DATA 5
210 DATA DOTTIE,555-2222
220 DATA TIM,555-1234
230 DATA JOE,555-8767
240 DATA JANE,555-5432
250 DATA MAX,555-5555
```

```
> 5 REM LISTING 2
 10 REM PHONE LIST - INTEGER
 20 CALL -936:  PRINT :  PRINT
 30 GOSUB 200
 40 DIM NM$(16),PH$(19)
 50 FOR I = 1 TO N
 60 GOSUB 200+I
 65 PRINT I;:  TAB 3
 70 PRINT NM$;:  TAB 20:  PRINT PH$
 80 NEXT I
 90 PRINT
100 INPUT "CHOOSE BY NUMBER ",A
110 GOSUB 200+A
120 PRINT
130 PRINT "THE PHONE NO. IS ";PH$
140 END
200 N=5:  RETURN
201 NM$="DOTTIE":PH$="555-2222":  RETURN
202 NM$="TIM":PH$="555-1234":  RETURN
203 NM$="JOE":PH$="555-8767":  RETURN
204 NM$="JANE":PH$="555-5432":  RETURN
205 NM$="MAX":PH$="555-5555":  RETURN
```

```
] 5 REM LISTING 3
 10 REM RANDOM DATA SELECTION - APPLESOFT
 20 FOR I = 1 TO 6 * RND (11)
 30 READ WD$
 40 NEXT I
 50 PRINT
 60 PRINT "THE SECRET WORD IS ";WD$;"."
 70 END
200 REM DATA
210 DATA APPLE,BOY,CATTLE,DINGO,ELEPHANT
```

```
> 5 REM LISTING 4
 10 REM RANDOM DATA SELECTION - INTEGER
 20 DIM WD$(20)
 30 GOSUB 200+1+RND (5)
 50 PRINT
 60 PRINT "THE SECRET WORD IS ";WD$;"."
 70 END
200 REM DATA
201 WD$="APPLE":  RETURN
202 WD$="BOY":  RETURN
203 WD$="CATTLE":  RETURN
204 WD$="DINGO":  RETURN
205 WD$="ELEPHANT":  RETURN
```

THE MYSTERIOUS MODEM

by Howard Lefkowitz

My D. C. Hayes Micromodem worked perfectly for six months. What a great "ABBS" we have. Then disaster. I moved my computer to a better location, about twenty feet away. Well, no problem, so let's extend the telephone lines and install a new jack. Goodby modem. Nothing but random junk appearing on the screen. Sometimes it worked, but every now and then, random characters. Much head scratching and suffering. Messages piling up. What could be wrong? I moved the computer back. Modem works perfectly. Twenty feet of wire causing the problem? You bet.

I live two blocks from WTOP. Its signals are everywhere - in the phones, stereo, amateur radio rig, etc. I had installed a filter for my ham radio rig near the old phone jack. It also, unknown to me, had filtered my modem/computer from this evil RFI (Radio Frequency Interference). The solution - make another filter and install it on the new phone jack.

If you get mysterious characters, live near a broadcast station, have a CB'er or ham nearby, then you might also need a filter.

Figure 1. illustrates the filter. Component values are not critical and you can use RF chokes of from one to ten millihenries. The filter should be mounted in a sealed metal enclosure. The enclosure should be grounded to a cold water pipe or, as an alternative, to a metal plate which is under the computer. In a pinch the screw which holds the electric outlet cover plate can be used. This will effectively filter out RFI being picked up by the telephone wires running through your house. The phone company will fix your phones if they are picking up signals (for free).

All parts can be bought at Radio Shack, including phone jacks if needed. You can install a female jack on the enclosure into which you plug your modem. You can also obtain a male plug with various lengths of wire to run from the filter to the already installed wall jack. Just keep the wire short from the modem to the filter (less than 3 feet).

Happy Communicating...

MODEM FILTER

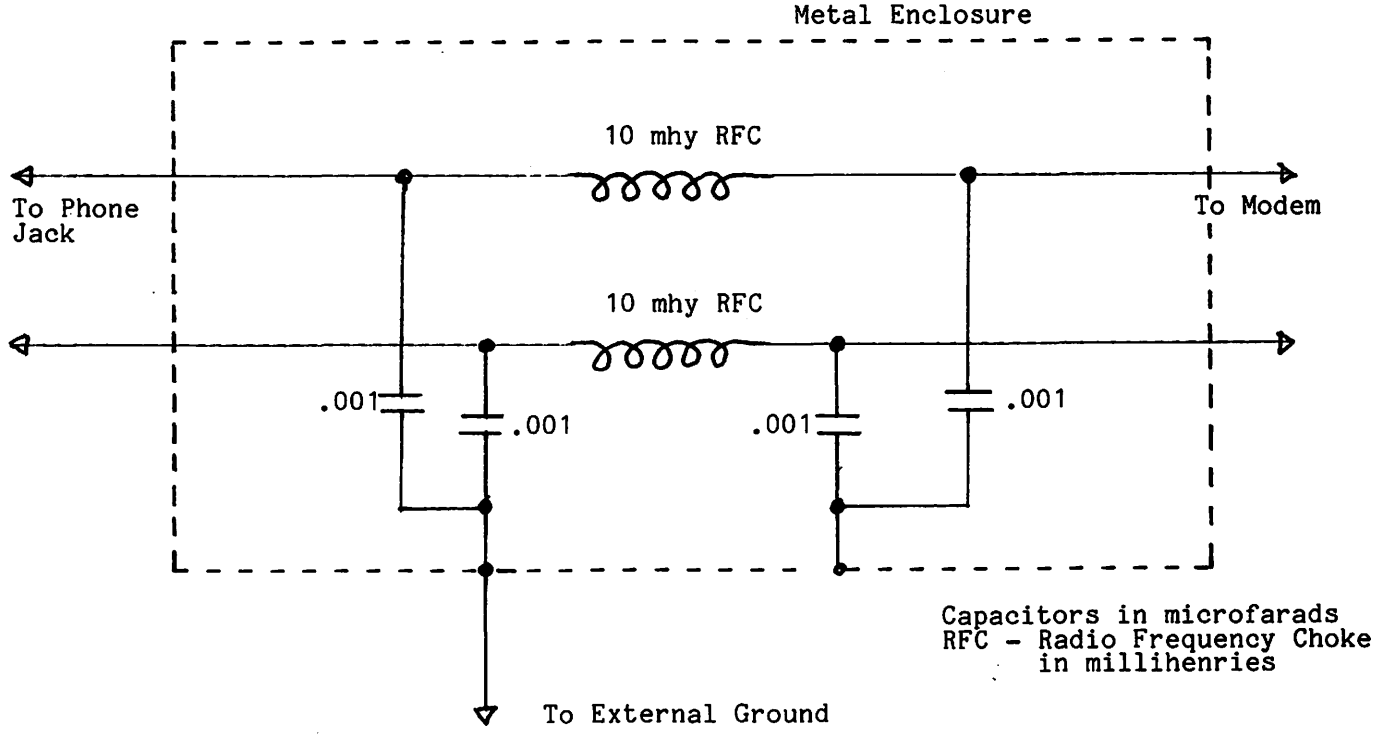


Figure 1.

QUOTABLE QUOTES

PAPER: Tiger by the Tail

UPPERCASE WITH THE SHIFT KEY

Installed 30 JAN 82

by Dr. Who?

>>>>>>>> CRAWL AWAY <<<<<<<<

So you now have upper and lower case capability! Wonderful! Hit Ctrl-A or Ctrl-Whatever and here comes upper case, for just the next letter. Then Ctrl-A and another one, etc.. etc.. etc... Wouldn't it be nice if it were just like a typewriter and we had a real shift key? -- You better believe it.

Easy?? This is not for the fainthearted. A wiring change will provide shift key upper case and also void the warranty. So if ninety days have expired and you have some soldering skills plus the proper tools, read on.

This modification works with the Dan Paymar chip, Smarterm board and most other 80-column upper/lower case boards. The basic (but sketchy) instructions are given in the IAC ApNote G-1. The 80-column boards usually provide firmware (and the Paymar chips software described in ApNote G-1) which permit shift key upper case, if the key is wired to SW2 on the paddle socket. The paddles currently use SW0 and SW1 for games, etc. Usually SW2 is not used for anything and is just sitting idle waiting for this modification.

The APPLE II Reference Manual refers to the paddle socket as the "Game I/O Connector", page 100. Figure 16 illustrates the pinouts. Pin 4 is listed as PB2 and this should be wired to the shift key. The pin numbering is based upon looking at the bottom of the socket (where the solder is). Pin 1 has a circle next to it in Figure 16. This is on the side of the chip with the notch. So make sure you have the right pin. The shift key is illustrated in Figure 17, page 101; lower right-hand corner. Notice that there are two keys shown (left- and right-hand). The shift key is wired to U4 on one side and ground on the other. You wire from the hot side to PB2. Now it gets a little tricky. The keyboard is connected through a ribbon cable to the mother board through a 16-pin connector described on page ~~160~~ 103, Figure 18. To insure that for future service, etc., the keyboard can be unplugged, you should wire using the existing ribbon cable and connector. From Figure 18, notice that pin 4 is unused. The procedure to be used is as follows:

(1) Disassemble keyboard from mother board.
(2) Wire the hot side of the shift key to pin 4 of the ribbon cable where it attaches to the keyboard.
(3) Wire the keyboard connector (Figure 18) pin 4 on the mother board to pin 4 of the game I/O connector (Figure 16) on the mother board.
(4) Put it back together and you are in business.

Thus, you actually install two wires: one on the keyboard and one on the mother board. Remember all sockets are 16-pin. On the "MAIN LOGIC BOARD SCHEMATIC", the game I/O connector (J14, upper part toward the left of center) pin 4 is connected to the keyboard connector (A7, right center) pin 4. Doesn't sound too hard!

Make sure you have the right pins before you start soldering. NOTE: pin 8 of the game I/O connector goes to ground, pin 8 of the keyboard connector goes to ground and one side of the shift key goes to the ground. Use these facts (with a continuity tester if available) to verify that you have the number four pins and the proper shift key connection.

Now soldering. You must use a low wattage iron (25 watts or less). Do your soldering underneath the mother board and keyboard (carefully please). A magnifying glass helps for you weak-eyed types. Use a small size, stranded teflon covered wire and be careful.

Disassembling the computer is a necessity. Unplug the keyboard and remove the bottom plate. This gives you two assemblies. You can wire the keyboard as is. The mother board will have to be removed from the bottom plate. Unplug the power connector, the speaker cable, game paddles and remove all the peripheral boards. Unscrew the nut in the center of the board. The plastic standoffs have small tabs. Gently squeeze the tabs and lift the board. One end at a time. As we said before the wire goes underneath the board. To reinstall just push the board down on the plastic standoffs (gently) till it locks and put the nut and washer back. Assemble the keyboard to the bottom plate and plug everything back in (leave out the peripherals) and test the computer. Then plug peripherals back in one at a time and test them.

Problems? Make sure you didn't make any solder bridges between socket pins. Make sure you wired the correct pins. Are all the chips firmly in their sockets?

The results are worth it. Shift-type-shift-type-Wow! Just like an IBM Selectric.

NOTE: For Smarterm board users Ctrl-A followed by Ctrl-V puts you in the shift key/upper case mode. Ctrl-Z locks you in upper case with Ctrl-A returning you to shift key operation.

Remember the socket pins are numbered from underneath (the soldered side). If this boggles your mind, take it to a dealer. Many dealers will be happy to make the modification for a (small??) fee.

Does Dr. Who? like this modifcation? ..Do apples have worms? Now my Pascal Editor Word Processor works like the Pro's. More about that in a future article. Remember

>>>>>>>> CRAWL AWAY <<<<<<<<

References:

(1) IAC ApNote G1, "Upper/Lower Case and Special Characteristics".
(2) APPLE II Reference Manual, 1979, A2L0001A.

(Howard Lefkowitz)

FLAVORS: LITTLE TIDBITS

by Burton S. Chambers III

(The flavors chosen for each tidbit are not necessarily an indication of content.)

GRAPE: Bug in my version of Apple Pascal Version 1.1 ?

After some experience with the new release of Apple Pascal and its operating system, I can report being quite pleased, although a few weeks ago I thought I had discovered a bug. When attempting to scan a string to see if it contained a certain character with the SCAN routine, which is a byte-oriented built-in, I didn't bother to use a subscript after the string identifier. SCAN starts at the byte containing the array since it was expecting a starting address pointing to the first element. I consider this a documentation problem, not a bug since Apple has repeatedly stated that strings are ESSENTIALLY packed arrays of characters. To repeat don't use:

```
SCAN(numberofchars, <>ch, mystring);
```

Do use:

```
SCAN(numberofchars, <>ch, mystring[1]);
```

When I first thought I had a bug I tore off a letter to Apple, which Mike Kane passed on to Jo Kellner, who in turn answered with a friendly letter within two weeks! Good for Apple. (Incidentally, Mike Kane of Apple assures me that he never got my letter regarding the error in Fortran.) Naturally, within two days of sending off the letter about my SCAN 'problem', my Level I service center (ie. Paul Sand of Computerland, Rockville) figured out what needed to be done to get it to work. Yeah, Paul! Since I haven't had the time to check out all the Apple-provided routines, I would appreciate hearing about any suspected bugs ASAP, but am happy to say that I haven't found any yet myself.

LEMON: Datacomm isn't one !

Datacomm works under Version 1.1 without patches to SYSTEM.APPLE. Necessity must breed solution. Last month I reported that the good folks at D. C. Hayes Microcomputer hadn't interfaced Datacomm with the new BIOS in Apple Pascal Version [1.1] (see item BANANA in the WAP Jan 1981 Flavors). Well they still haven't, but are glad to report they don't have to. The new BIOS doesn't seem to need a patch, Datacomm will automatically work on version [1.1], which must imply that the I/O in version 1.1 runs quicker than the first release. However, there does exist one small problem that the user may have if he forgets which case he is generating his letters in. Since the program was written for the first version, it doesn't recognize lower case in its command set as if it were upper case, therefore, you must set upper case when issuing commands. This isn't a bug, but may be inconvenient. You can, of course, modify the program to suit your own taste, since the source code has been provided. I have tested Datacomm myself, but since my typing is slow, Datacomm didn't get stressed very hard. Others have reportedly used it with version [1.1] with no problems. Naturally, now that you can generate lower case from the Apple keyboard in the Pascal system, someone may wish to modify Datacomm to recognize this, or a similar capability, while sending. That may be a difficult problem, because it may make the program too slow in the sending mode. Oh well, you can always generate the message in the Apple Pascal Editor, and then send it directly with Datacomm.

LIME: Breaking up large TEXT files

You can break up very large TEXT files that won't fit into memory with the Filer. It is a bother, but easier than writing a one-shot program. Some people may not agree with me after reading the rest of this item. The secret is realizing that deleting a file doesn't remove it, it just won't appear in the directory. It stays around until you write over it (e.g. krunching a disk may do this). Next you need to know that a TEXT file has a 2 block header. As an example, to be generalized to as many parts as necessary, let us say you have a file BEAR.TEXT which is 52 blocks long. You got this file from someone else who uses UCSD Pascal with a different version of the Editor, e.g. some USUS-provided programs. To make the exercise easier, assume that is all you have on the diskette containing BEAR, and the drive is the prefixed drive. In the filer, K(runch the disk. Now BEAR starts at block 6. Now, M(ake MARKER[1], so you don't lose your place (trust me!). R(emove BEAR.TEXT and then Make J. Do an E(xtended list to see what happened. Where BEAR.TEXT

contd.

was is <unused> and all the other blocks are filled with MARKER, 1 block, and J, the rest. Now M(ake BEAR.1.TEXT[26], M(ake BEAREND[26], R(emove J, T(ransfer BEAR.1.TEXT to BEAR.1.TEXT and continue to have faith. (You can do an E(xtended list at each step if you like, it may help you see what is happening.) Now M(ake J (he is my favorite label for junk), and then M(ake HEADER[2]. Now, comes the clever part, the rest was sheer agony. K(runch the disk, R(emove J, M(ake J, R(emove HEADER, R(emove BEAREND, and then, M(ake BEAR.2.TEXT[28]. Now, to clean up the blood, R(emove J, R(emove MARKER, and if you like K(runch the diskette. Wait you say, how come 54 blocks now instead of 52. The answer is that this whole exercise is done to duplicate the 2 blocks at the start of a TEXT file that the system expects. I duplicate it by using a removed copy and letting the system concatenate (stick them together) it with the end of the bear. Remember, the number of blocks in a TEXT file are expected to be even, so be careful how you split them up.

To review the process (assumming that only one big file is on the prefixed disc, set below to drive #5:) you would type from the COMMAND mode the following data, under heading COMMANDS, where their meanings are summarized under MEANING:

COMMANDS	MEANINGS
FP#5:	Get in filer, and set prefixed drive to #5:.
K:	Krunch it,
YMMARKER[1]	Yes, and make MARKER (one block).
RBEAR.TEXT	Remove BEAR.TEXT
YMJ	Yes, update directory. Make J, a junk file
MBEAR.1.TEXT[26]	Make BEAR.1.TEXT
MBEAREND[26]	Make BEAREND
RJ	Remove J
YTBEAR.1.TEXT	Yes, update directory. Transfer BEAR.1.TEXT
BEAR.1.TEXT	to BEAR.1.TEXT
MJ	Make J
MHEADER[2]	Make HEADER of 2 blocks (TEXT information)
RJ	Remove J.
YK:	Yes, update directory. Krunch prefixed disc,
YMJ	Yes, and Make J.
RHEADER	Remove HEADER.
YRBEAREND	Yes, update directory. Remove BEAREND.
YMBEAR.2.TEXT[28]	Yes, update directory. Make BEAR.2.TEXT.
RJ	Remove J.
YRMARKER	Yes, update directory. Remove MARKER.
YK:	Yes, update directory. Krunch prefixed disk,
Y	and Yes.

In this example, each new line indicates that a <RETURN> is required. If you have version 1.1 then you may want to M(ake a file with this information on it. This file can be subsequently executed (EXEC/) or edited (E(dit).

ORANGE: Apple ///

I finally have seen and briefly used an Apple ///. And frankly, I am sure one day it will be a useful tool, but I'm in no hurry to get it for one principal reason: the software of interest to me is not yet available. Although the Apple /// has an Apple][emulator mode (through software), it can only emulate some features of our machines. And then, it won't run Pascal and Fortran yet. If anyone has visions of running out and buying one, and then using your available software on it (until you find time to write the Apple /// specific software) you may be seriously disappointed. Since, Apple has introduced their Information Analyst version first, you'll be able to run Visicalc ///, which incidentally looks pretty good.

I think I'll wait until after Pascal has been introduced on the Apple ///, and some brave pioneers have mapped out the territory before jumping in. If you decide to be one of the brave ones, you'll find that most of the Apple ///'s will come with 128K bytes. (Historical note: IBM 7040, 7044, 7090, and 7094 computers were 32K 36-bit word machines - an Apple /// is 89% of that in bits. Anybody, remember the prices for the IBM mainframes ?). Since, Computerland Rockville has sold at least two already, I hope our trusty pioneers will tell us how neat their Apple ///'s are. Good computing!

BLAISE AWAY

by Dr. Wo

TAKING A FLING AT A FILER

OR

REINVENTING A BIG WHEEL

In this month's column I'd like to share with you some of my recent efforts to duplicate some of the functions of the system filer: a library UNIT, 'filer', you can reference in any program of your choosing.

We will be making use of the knowledge gained in our peek at Apple's memory using the program 'pagedumper' described here last time. Please refer to the December issue of Washington Apple Pi for the program and related dscussion. Especially important is the discussion of the location of the table of physical devices and the names of the boot and prefix (system default) volumes we found in our exploration of higher memory. We will also make use of the declarations for a directory as supplied by Apple and reprinted here last. They appear again in the appropriate places in the declarations for the UNIT.

For those of you who have no desire to key in the approximately 800 lines of code presented below, the program is in our club's library. See below for more information.

UNIT filer: What it does

UNIT filer is a regular library unit which can be 'used' by any of your programs. It supports six functions familiar to you from the system filer: list the contents of a directory; set the name of the prefix (system) volume; list the volumes on line; zero a directory; crunch a disk from the high end; and remove files. The six procedures are housed in the principal procedure of the UNIT, 'minifiler'. The six are supported by a variety of primitive procedures such as fetching a directory into memory, searching the table of physical devices and parsing volume and file name strings.

In addition to 'minifiler', the UNIT sports three utilities which are also available to the program which hosts the UNIT. They are 'fetchdir', which fetches into memory the directory of a selected unit, 'getfileinfo', which retrieves the characteristics of a file, and 'getvolinfo', which does the same for a volume.

Similar to the system filer, 'minifiler' supports several short hand and wild card notations:

```
:  -for the default volume
*  -for the boot volume
#n -to denote physical unit n
=  -wild card for the 'remove'
    procedure
```

In addition 'minifiler' is supported by a volume/file name parser which recognizes pascal naming syntax. For example, the following are recognized as equivalent when the boot disk and default disk are the same, namely APPLE1: on unit #4:

```
          JUNK.TEXT
         *JUNK.TEXT
        *:JUNK.TEXT
       #4:JUNK.TEXT
  APPLE1:JUNK.TEXT
```

Reinventing the Wheel?

To some of you this may smack of reinventing the wheel. To me it was a challenge and a great deal of fun. In addition, it has turned out to be a very useful program which has greatly increased the flexibility of a communications program I have written for the D.C. Hayes Micromodem. Another possibility which comes to mind is an interactive statistics program I am developing. In general, any program (or system, if you will)

contd.

which you do not, or are not able, to conveniently exit, but for which you want filer capabilities, is a candidate application.

Here's hoping you enjoy it, and find it as useful as I have!

Behind Every Silver Lining

Unfortunately the UNIT is heavily system (i.e. Apple) and version dependent. The program reads the table of physical devices and the names of the boot and default disks from memory. And it updates the name of the default disk in memory so that the system recognizes it. The addresses for these data are obviously system dependent and it turns out they also vary from version 1.0 to version 1.1 Apple Pascal.

The trouble then is that this library is in jeopardy whenever Apple revises its Pascal. At the very least you may have to update the program to refer to new adresses. Worse, Apple could decide to change the declarations for the table of devices and so on.

Here are the current adresses, found using 'pagedumper':

Name	Version 1.0	Version 1.1
tableaddrs	-22136	-21874
booteraddrs	-22262	-22000
prefixaddrs	-22270	-22008

Fortunately, system revisions are not an every day occurence so these procedures could serve well for quite a while. However, like any bad habit, their use can be overdone. Enjoy now and pay later perhaps?

A Word About UNITs

Library UNITs are a powerful way to combine related or frequently referred to procedures which can then be referred to by other, host, programs as a logical unit. In addition they are compiled once and linked to their hosts.

There are three major syntactical components to a UNIT, the INTERFACE section, the IMPLEMENTATION section and the initialization code.

The INTERFACE section contains all of the declarations (CONSTants, data TYPES, VARiables, PROCEDURES etc.) which are available to the host program. In effect, these become global declarations in the the host program.

The IMPLEMENTATION section contains the remaining declarations and procedures which the UNIT needs to get its job done, as well as the code which defines the procedures in the INTERFACE. The IMPLEMENTATION section is private to the UNIT; it is not seen nor can it be accessed by the host program. Declarations made in the INTERFACE will not conflict with any made in the host program.

The initialization code for a UNIT is used to initialize (what else!?) the unit and is executed before any code from the host is executed. The initialization can set up variables or execute code from either the INTERFACE or the IMPLEMENTATION or it can be null--'BEGIN END.'

There are two types of units, REGULAR units and INTRINSIC units. Even though the Apple Pascal manuals leave something to be desired in their explanation of the differences, I will not go into them here. Let's save that for another time.

Filer's INTERFACE: Utilities

Filer's INTERFACE starts with the declarations for an Apple Pascal directory. These were discussed briefly in my last article and are fairly well documented here again. These declarations were supplied by Apple and, I understand, are available upon request from them.

The remainder of the INTERFACE comprises three utilities and the

contd.

procedure 'minifiler'. Let's start with the utilities.

Procedure 'fetchdir' fetches into memory the directory of a designated unit. It starts off by verifying that the unit has a directory and then does a 'unitread' to haul the directory into memory. Provided everything has gone well it assigns TRUE to the variable OK and the contents of the directory to the variable 'dir'.

Procedure 'getvolinfo' gets the information stored in the zeroeth, or volume information entry of the directory associated with the volume name or unit number passed to it. First it calls 'parsevid' to parse the volume/unit name passed to it, then it searches the table of devices for the volume. If everything goes all right it allocates space for a directory, fetches the directory in, extracts the zeroeth entry setting 'volinfo' equal to it, and sets OK to TRUE. Otherwise it sets OK to FALSE and returns garbage in 'volinfo'. It ends by releasing the space reserved for the directory.

The procedure 'getfileinfo' performs the task of retrieving the directory information for a specified file. First it calls 'parsetidvid' to parse the file name passed to it, then it searches the unit table for the designated volume. If all is well it makes space for a directory, fetches it, and searches the directory for the designated file. If it finds the file it assigns TRUE to OK and the file information to 'fileinfo'; otherwise it assigns FALSE to OK. Finally, it releases the space allocated to the directory.

INTERFACE Continued: 'minifiler'

The procedure 'minifiler' consists of three pieces, a routine which reserves space for a directory and a crunch buffer, a REPEAT loop, and a routine to release space back to the system.

The procedure 'reservespace' allocates space for a directory and for the buffer which is used for crunching disks. The procedure takes account of the limits to memory. It starts off by marking the top of the heap, then it allocate s space for the directory and sets 'dirp' to point there via 'new(dirp)'. Then it marks the heap again via 'mark(heap)' and repeatedly allocates space for blocks of 512 bytes until either the space available between the top of the heap and the top of the stack is less than 1K words, or until 63 blocks have been set aside. (Sixty-three is the limit since the Apple can't count as high as 64*512. It won't get up to 63 either because memory is taken up by programs. I heard that once!)

Note the declarations:

```
TYPE byte=PACKED ARRAY[0..0] OF 0..255;
VAR  heap:^byte;
```

Therefore, 'heap' can point to a byte in memory.Thanks to 'reservespace' it points to the beginning of the Krunch buffer. Note also that heap^[0] is the first byte of the Krunch buffer and that with compiler range checking off, heap^[n] can be used to refer to the n-th byte of the buffer. This trick is used in the procedure 'Krunch'.

The procedure 'selfree' merely returns to the system the space allocated by 'reservespace'. Upon exit, 'minifiler' returns the system to its original state so far as dynamically allocated memory is concerned.

The Heart of the Matter

The heart of 'minifiler' is the REPEAT loop. We'll take a look at each procedure callable within it but not in great detail. First the generalities.

Each of the procedures except 'volumes' accepts an input string which is then parsed. Provided no syntax error is uncovered, each procedure eventually searches for the directory named or implied in the input. If no I/O error is commited

contd.

reading the directory, each then goes onto to its main tasks.

Each procedure is constructed in a top down fashion. Thus each calls lower and lower level procedures until the original task is complete.

'listing'
=========

The procedure 'listing' produces an extended directory listing formatted for a 40 column screen. If there are more files than can be listed on one screen, the routine pauses to read the keyboard;typing <ESC> then ends the listing, any other key continues it.

After parsing the input string, 'listing' searches for the needed directory; if it finds it calls 'displaydir'. The latter routine repeatedly calls 'writefileinfo' which handles the chore of formatting information on the screen. Lastly, 'displaydir' keeps track of whether the screen id full.

'setprefix'
==========

The procedure 'setprefix' sets the name of the system default volume. The system will remember the change after you exit 'minifiler'. This link into the system is one reason why the UNIT is version dependent.

The procedure begins by parsing the input string. If the parse is succesful it either sets the prefix to 'volid' directly or reads in a directory and then sets it. What happens depends on whether the '#n' notation is used or not.

'volumes'
========

The procedure 'volumes' lists the volumes on line. It does this by searching 'unittable', the table of on line devices which is maintained by the operating system. This is another example of version dependency.

'zero'
======

The procedure 'zero' blanks out the selected directory. It does this simply by setting the field 'dnumfiles' in the zeroeth entry of the directory to 0. It will ask for a confirmation before going ahead. It begins by parsing the input string and searching for the specified directory. If everything is OK it then 'zaps' the directory. The procedure 'zap' is declared outside 'zero' because it is also called by 'remove'.

'Krunch'
========

The procedure 'Krunch' crunches a disk from the end, block 280, and updates its directory. It leaves blocks 0-5 alone of course, since these are system blocks. It uses the buffer set up in 'reservespace' and the compiler range checking option mentioned above.

The number of blocks a file has to be moved is computed. If non-zero, the file is broken into chunks of size 'buffblkcount' (the size of the buffer) and succesive reading and writing of chunks takes place.

'Krunch' calls 'squash' which calls 'move' and 'gapsize'. The latter function computes the distance between files; 'move' actually moves the files around. 'Squash' keeps track of whether any moving takes place (BOOLEAN VARiable 'squashflag'), and only if it does is the directory updated.

'remove'
========

The procedure 'remove' deletes files from a directory. It asks for confirmation before taking any action. Specifying '=' in response to the query 'Remove which files?' is equivalent to zeroing the directory.

The procedure parses input and looks for a directory. If everything is OK it calls 'delete'. The latter searches the directory and sets the 'status' field of each directory entry to TRUE if that file is to be deleted.

In other words, delete makes sense of the '=' wildcard.

contd.

After flagging everything, 'delete' calls 'dr@pfiles' which updates the directory, subject to confirmation.

IMPLEMENTATION

The UNIT is supported by several low level or utility procedures. Very briefly they are as follows:

'readdir'

The procedure 'readdir' hauls into memory the directory of the specified unit. It contains error checking code for making sure the unit has a directory and for flagging disk I/O errors. If the read is successful, the 'unitname' field of the specified entry in 'unittable' is updated.

'refreshtable'

This procedure updates 'unittable' by looping through a series of calls to 'readdir'.

'foundir'

This BOOLEAN function searches for the directory of a specified volume. It accepts any valid volume specification, e.g. *,:,#4,APPLE1:, etc. If the search is by volume number, it calls 'numberfound', else it calls 'namefound'. These routines incorporate I/O error checks.

'capitalize'

This routine capitalizes the input string.

'getvid', etc.

The procedure 'getvid' supports 'parsevid' and 'parsetidvid'. These are the volume and file name parsers.

All contain code to flag syntax errors.

'flagparserr', 'flagioerr'

Lastly, 'flagparserr' and 'flagioerr' provide error messages. The former gives parser errors, the latter I/O errors that result from bad directory reads etc.

Initialization

The initialization of the UNIT is simple and lives in the procedure 'initfiler'. This procedure sets up 'booter' and 'prefix' and the table of units, 'unittable'. It also sets up the values of clear-to-end-of-screen and two other characters.

Using the UNIT

In order to use the UNIT you need only compile it and store it in a library. If you decide to put it in SYSTEM.LIBRARY, you might want to make it into an intrinsic unit. This strikes me as a waste of valuable boot disk space, so I store the code in a library called FILER.LIB. When I want to use it, I make sure that FILER.LIB is on line at compile time and use the 'U' option of the compiler. After compilation comes linking to the host program.

Take a fling with 'filer' and

B L A I S E A W A Y !!!!!!!

Dr. Wo

contd.

```
(*$S#)
UNIT filer;
(*************************************
*
* A library unit to duplicate some of
* the functions of the system filer.
*
*         CONTRIBUTED TO
*
*              the
*       Washington Apple Pi
*
*           Pascal SIG
*
*               by
*
*             Dr. Wo
*
*
*            Feb 1981
*
*        Written by Dr. Wo
*
*************************************)

INTERFACE
CONST maxdir=77;    (maximum number of entries in directory)
      vidleng=7;    (number of characters in volume id)
      tidleng=15;   (number of characters in title id)
      fblksize=512; (standard disk block length)
      dirblk=2;     (directory starts at this disk block address)

      maxunit=12;

TYPE  daterec=PACKED RECORD
                month:0..12;(0 implies meaningless date)
                day:0..31;
                year:0..100;(100 implies dated volume is temporary)
                END;

      (volume id)
      vid=string[vidleng];

      dirrange=0..maxdir;

      (title id)
      tid=string[tidleng];

      filekind=(untypedfile,xdskfile,codefile,textfile,infofile,datafile,
                graffile,fotofile,securedir);

      (directory layout)
      direntry=PACKED RECORD
                dfirstblk:integer;(first physical disk address)
                dlastblk:integer; (points at bock following last used block)
                CASE dfkind:filekind OF
                 securedir,untypedfile:
                  (only in dir[0], this is volume info)
                  (filler1:0..2048; (13 bits for downward compatibility ??)
                   dvid:vid;
                                                                  contd.
```

```
                  deovblk:integer; {number of blocks in this volume}
                  dnumfiles:dirrange; {number of files in directory}
                  dloadtime:integer; {time of last access ??}
                  dlastboot:daterec); {most recent date setting}
                xdskfile,codefile,textfile,infofile,
                datafile,graffile,fotofile:
                 (filler2:0..1024; {12 bits for downward compatibility}
                  status:BOOLEAN; {for filer wildcards}
                  dtid:tid; {title of file}
                  dlastbye:1..fblksize; {numbr of bytes in file's last block}
                  daccess:daterec); {date last modified}
                END;
    directory=ARRAY[dirrange] OF direntry;

    unitrange=0..maxunit;

VAR dirp:^directory; { global directory pointer }

PROCEDURE minifiler;

PROCEDURE fetchdir(unitnum:integer;VAR OK:BOOLEAN;VAR dir:directory);

PROCEDURE setvolinfo(title:string;VAR OK:BOOLEAN;VAR unitnum:unitrange;
                     VAR volinfo:direntry);

PROCEDURE setfileinfo(title:string;VAR OK:BOOLEAN;VAR volid:vid;
                      VAR unitnum:unitrange;VAR fileinfo:direntry);

IMPLEMENTATION
CONST null='' ; { null string }

      notblocked=-1; { used by readdir and flagioerr }

      tableaddrs= -21874; { system adress of table of physical units }
      booteraddrs=-22000; { system address of name of boot disk }
      prefixaddrs=-22008; { system address of name of prefix disk }
      (**********************************
      *
      * THE PRECEEDING ARE THE VERSION 1.1
      * ADDRESSES. THE VERSION 1.0 DATA
      * ARE:
      * tableaddrs= -22136
      * booteraddrs=-22262;
      * prefixaddrs=-22270;
      *
      **********************************)

      maxblocks=63; { maximum block size of "Krunch" buffer }

TYPE tablentry=RECORD
                 unitname:vid;
                 CASE blkdunit:BOOLEAN OF
                  TRUE:(totalblks:INTEGER);
                 end;
    table=ARRAY[unitrange] OF tablentry;

    { the type of variable heap used in initfiler }
    byte=PACKED ARRAY[0..0] OF 0..255;

    { vid/tid parsing ewrrors used by flagparserr }
```

contd.

```
    perr=(pnull,pvidnmlng,pnotdigit,pnotunitnum,ptidnmlng,pcolon);

VAR ch:CHAR; { global character variable }
    gindex:INTEGER; {global index variable}
    iorslt:INTEGER; { result of last disk i-o operation }
    prslt:perr; { result of last parse }
    physunit:unitrange; { number of physical unit }
    cleos,cleoln,bell:CHAR;
    volid,booter,prefix:vid;
    fileid:tid;
    title:STRING;
    unittable:table; { table of units on line, initialized by initfiler }
    heap:^byte; { pointer into heap to mark Krunch buffer, see initfiler }
    dirheap:^INTEGER; { pointer into heap to mark beginning of directory }
    buffblkcount:INTEGER; { size of Krunch buffer in blocks }
    { used by initfiler to initalize booter and prefix
      and by setprefix to reset prefix }
    stringptr:RECORD CASE BOOLEAN OF
                TRUE:(addrs:INTEGER);
                FALSE:(contents:^vid);
                END;


PROCEDURE initfiler;
VAR tableptr:RECORD CASE BOOLEAN OF
                TRUE:(addrs:INTEGER);
                FALSE:(contents:^table);
                END;

BEGIN
 tableptr.addrs:=tableaddrs;
 unittable:=tableptr.contents^;
 stringptr.addrs:=booteraddrs;
 booter:=stringptr.contents^;
 stringptr.addrs:=prefixaddrs;
 prefix:=stringptr.contents^;
 cleos:=chr(11);
 cleoln:=chr(29);
 bell:=chr(7);
END;

PROCEDURE readdir(unitnum:unitrange;VAR iorslt:INTEGER);
BEGIN
 (*$I-*)
  IF unittable[unitnum].blkdunit THEN
   BEGIN
    unitread(unitnum,dirp^,sizeof(directory),dirblk);
    iorslt:=ioresult;
    IF iorslt=0 THEN
     WITH unittable[unitnum] DO
      BEGIN
       unitname:=dirp^[0].dvid;
       totalblks:=dirp^[0].deovblk;
      END;
   END
   ELSE iorslt:=notblocked;
 (*$I+*)
END;

PROCEDURE refreshtable;
```


contd.

```
BEGIN
 FOR sindex:=maxunit DOWNTO 1 DO
  readdir(sindex,iorslt);
END;

FUNCTION foundir(VAR volid:vid;VAR physunit:unitrange;
                 VAR iorslt:INTEGER):BOOLEAN;

 FUNCTION numberfound(VAR volid:vid;physunit:unitrange;
                      VAR iorslt:INTEGER):BOOLEAN;
 BEGIN
  numberfound:=FALSE;
  readdir(physunit,iorslt);
  IF iorslt=0 THEN
   BEGIN
    volid:=dirp^[0].dvid;
    numberfound:=TRUE;
   END;
 END;

 FUNCTION namefound(volid:vid;VAR physunit:unitrange;
                    VAR iorslt:INTEGER):BOOLEAN;
 VAR foundit,volintable:BOOLEAN;
 BEGIN
  foundit:=FALSE;
  volintable:=FALSE;
  physunit:=0;

  REPEAT
   physunit:=physunit+1;
   volintable:=volid=unittable[physunit].unitname;
  UNTIL volintable OR (physunit=maxunit);

  IF volintable THEN
   BEGIN
    readdir(physunit,iorslt);
    foundit:=(iorslt=0) AND (volid=dirp^[0].dvid);
   END;

  IF NOT foundit THEN
   BEGIN
    physunit:=maxunit;
    REPEAT
     readdir(physunit,iorslt);
     foundit:=(iorslt=0) AND (dirp^[0].dvid=volid);
     physunit:=physunit-1;
    UNTIL (physunit=0) OR foundit;
    IF foundit THEN physunit:=physunit+1;
   END;
  namefound:=foundit;
 END;

BEGIN foundir
 IF volid='#' THEN foundir:=numberfound(volid,physunit,iorslt)
  ELSE foundir:=namefound(volid,physunit,iorslt);
END;

PROCEDURE capitalize(VAR title:STRING);
CONST space=' ';
      smalla=97;
```

contd.

```
      smallz=122;
BEGIN
 sindex:=1;
 WHILE sindex<=length(title) DO
  BEGIN
   ch:=title[sindex];
   IF ch<=space THEN delete(title,sindex,1)
   ELSE BEGIN
    IF ch IN [chr(smalla)..chr(smallz)] THEN
     title[sindex]:=chr(ord(ch)-smalla+ord('A'));
    sindex:=sindex+1;
   END;
  END;
END;

PROCEDURE getvid(VAR title:STRING;VAR volid:vid;
                 VAR physunit:unitrange;VAR prslt:perr);

 PROCEDURE getname;
 BEGIN
  sindex:=pos(':',title);
  IF sindex<=vidleng+1 THEN
   BEGIN
    IF sindex=0 THEN BEGIN volid:=prefix;insert(':',title,1);END
     ELSE BEGIN
      volid:=copy(title,1,sindex-1);
      delete(title,1,sindex-1);
     END;
   END
   ELSE prslt:=pvidmalng;
 END;

 PROCEDURE getunitnumber;
 BEGIN
  volid:='#';
  delete(title,1,1);
  IF length(title)>0 THEN
   BEGIN
    ch:=title[1];
    IF ch IN ['0'..'9'] THEN
     BEGIN
      REPEAT
       physunit:=10*physunit+ord(ch)-ord('0');
       delete(title,1,1);
       IF length(title)>0 THEN ch:=title[1];
      UNTIL (length(title)=0) OR NOT (ch IN ['0'..'9']);
      IF NOT (physunit IN [1..maxunit]) THEN prslt:=pnotunitnum;
     END
     ELSE prslt:=pnotdigit;
   END
   ELSE prslt:=pnotdigit;
 END;

BEGIN getvid
 ch:=title[1];
 IF ch='*' THEN BEGIN volid:=booter;delete(title,1,1);insert(':',title,1); END
 ELSE IF ch=':' THEN volid:=prefix
 ELSE IF ch='#' THEN getunitnumber
 ELSE getname;                                                    contd.
END;
```

```
PROCEDURE parsevid(title:STRING;VAR volid:vid;
                   VAR physunit:unitrange;VAR prslt:perr);
BEGIN
 volid:=null;
 physunit:=0;
 prslt:=pnull;
 capitalize(title);
 getvid(title,volid,physunit,prslt);
 IF prslt=pnull THEN
  BEGIN
   IF pos(':',title)=1 THEN delete(title,1,1);
   IF length(title)>0 THEN prslt:=pvidnmlng;
  END;
END;

PROCEDURE parsetidvid(title:STRING;VAR volid:vid;VAR physunit:unitrange;
                      VAR fileid:tid;VAR prslt:perr);
BEGIN
 volid:=null;
 physunit:=0;
 fileid:=null;
 prslt:=pnull;
 capitalize(title);
 getvid(title,volid,physunit,prslt);
 IF prslt=pnull THEN
   IF pos(':',title)<>1 THEN prslt:=pcolon
   ELSE BEGIN
    delete(title,1,1);
    IF length(title)>tidleng THEN prslt:=ptidnmlng
     ELSE fileid:=title;
   END;
END;

PROCEDURE flagparserr(prslt:perr);
BEGIN
 CASE prslt OF
  ptidnmlng:writeln('File name is too long');
  pvidnmlng:writeln('Volume name is too long');
  pnotdigit:writeln('Unit number expected');
  pnotunitnum:writeln('Invalid unit number');
  pcolon:writeln('Colon expected in vol name');
  END;
END;

PROCEDURE flagioerr(iorslt:INTEGER);
BEGIN
 IF iorslt IN [1..16] THEN
   CASE iorslt OF
    1:writeln('Bad block');
    2:writeln('Bad unit number');
    3:writeln('Bad mode');
    4:writeln('Undefined hardware error');
    5:writeln('Lost unit');
    6:writeln('Lost file');
    7:writeln('Bad title, illegal file name');
    8:writeln('No room, insufficient space');
    9:writeln('No unit, no such volume');
    10:writeln('No such file on line');
    11:writeln('Duplicat file');
```

contd.

```
    12:writeln('Not closed');
    13:writeln('Not open');
    14:writeln('Bad format');
    15:writeln('Ring buffer overflow');
    16:writeln('Write protect error');
   END
  ELSE IF iorslt=notblocked THEN writeln('Not a blocked volume')
  ELSE writeln('Undefined I-O error');
END;

PROCEDURE fetchdir;
VAR physunits:SET OF unitrange;
BEGIN
 physunits:=[0..maxunit];
 OK:=FALSE;
 IF unitnum IN physunits THEN
  IF unittable[unitnum].blkdunit THEN
   BEGIN
    (*$I-*)
    unitread(unitnum,dir,sizeof(directory),dirblk);
    IF ioresult=0 THEN OK:=TRUE;
    (*$I+*)
   END;
END;

PROCEDURE setvolinfo;
BEGIN
 mark(dirheap);
 new(dirp);
 OK:=FALSE;
 parsevid(title,volid,unitnum,prslt);
 IF prslt=pnull THEN
  IF foundir(volid,unitnum,iorslt) THEN
   BEGIN
    volinfo:=dirp^[0];
    OK:=TRUE;
   END;
 release(dirheap);
END;

PROCEDURE setfileinfo;
BEGIN
 mark(dirheap);
 new(dirp);
 OK:=FALSE;
 parsetidvid(title,volid,unitnum,fileid,prslt);
 IF prslt=pnull THEN
  IF foundir(volid,unitnum,iorslt) THEN
   BEGIN
    sindex:=0;
    WHILE NOT ((sindex=dirp^[0].dnumfiles) OR OK) DO
    BEGIN
     sindex:=sindex+1;
     OK:=fileid=dirp^[sindex].dtid;
    END;
    IF OK THEN fileinfo:=dirp^[sindex];
   END;
 release(dirheap);
END;
```

contd.

```
PROCEDURE minifiler;

PROCEDURE reservespace;
CONST onekilo=1024;
TYPE block=PACKED ARRAY[1..fblksize] OF 0..255;
VAR heapblk:^block;
BEGIN
 mark(dirheap);
 new(dirp);
 buffblkcount:=0;
 mark(heap);
 REPEAT
  new(heapblk);
  buffblkcount:=buffblkcount+1;
 UNTIL (buffblkcount=maxblocks) OR (memavail<=onekilo);
END;

PROCEDURE prompt(VAR ch:CHAR);
BEGIN
 gotoxy(0,0);
 write(cleoln,bell,'List,Vols,Prfx,Zero,Krunch,Rmov,Quit?');
 read(keyboard,ch);
 writeln;
END;

PROCEDURE listing;

 PROCEDURE displaydir;
 CONST fullscree=21;
       esc=27;
 VAR freeblks,maxgap,linecount:INTEGER;

  PROCEDURE writefileinfo(entrynum:INTEGER;
                          VAR linecount,freeblks,maxgap:INTEGER);
  VAR gap:INTEGER;
  BEGIN
   gap:=0;
   WITH dirp^[entrynum] DO
    BEGIN
     write(dtid);
     write(dlastblk-dfirstblk:tidlen+4-length(dtid));
     WITH daccess DO
      BEGIN
       write(day:3,'-');
       CASE month OF
        1:write('Jan-');
        2:write('Feb-');
        3:write('Mar-');
        4:write('Apr-');
        5:write('May-');
        6:write('Jun-');
        7:write('Jul-');
        8:write('Aug-');
        9:write('Sep-');
        10:write('Oct-');
        11:write('Nov-');
        12:write('Dec-');
        0:write('???-');
        END;
       write(year:2,' ');
```

contd.

```
      END;
     write(dfirstblk:4);
     CASE dfkind OF
      xdskfile:writeln(' BAD');
      codefile:writeln(' Code');
      textfile:writeln(' Text');
      infofile:writeln(' Info');
      datafile:writeln(' Data');
      graffile:writeln(' Graf');
      fotofile:writeln(' Foto');
      END;

     IF entrynum=dirp^[0].dnumfiles THEN gap:=dirp^[0].deovblk-dlastblk
      ELSE gap:=dirp^[entrynum+1].dfirstblk-dlastblk;
     IF gap>0 THEN
      BEGIN
       writeln('<UNUSED> ',gap:tidlens-5,dlastblk:15);
       linecount:=linecount+1;
       freeblks:=gap+freeblks;
       IF gap>maxgap THEN maxgap:=gap;
      END;
     linecount:=linecount+1;
    END;
  END;

BEGIN displaydir
  page(output);
  gotoxy(0,1);
  writeln(dirp^[0].dvid,':');
  linecount:=1;
  maxgap:=0;
  freeblks:=0;
  gindex:=1;
  WHILE gindex<=dirp^[0].dnumfiles DO
   BEGIN
    writefileinfo(gindex,linecount,freeblks,maxgap);
    IF (linecount MOD fullscreen=0) THEN
     BEGIN
      writeln(gindex,' of ',dirp^[0].dnumfiles,' files.');
      read(keyboard,ch);
      IF ch=chr(esc) THEN exit(displaydir)
       ELSE BEGIN
        page(output);
        gotoxy(0,1);
        writeln(dirp^[0].dvid);
        linecount:=1;
       END;
     END;
    gindex:=gindex+1;
   END;
  write(dirp^[0].dnumfiles,' files, ');
  write(freeblks,' unused blocks, ');
  writeln(maxgap,' in largest');
END;

BEGIN listing
  write(cleos,'Directory listing of? ');
  readln(title);
  IF length(title)>0 THEN
```

contd.

```
  BEGIN
   parsevid(title,volid,physunit,prslt);
   IF prslt<>pnull THEN flagparserr(prslt)
    ELSE IF foundir(volid,physunit,iorslt) THEN displaydir
    ELSE flagioerr(iorslt)
  END;
END;

PROCEDURE volumes;
BEGIN
 writeln(cleos);
 writeln('Volumes on line:');
 writeln;
 refreshtable;
 FOR gindex:=1 TO maxunit DO
  IF length(unittable[gindex].unitname)>0 THEN
   writeln(gindex:2,' ',unittable[gindex].unitname,':');
  writeln;
  writeln('Prefix is: ',prefix,':');
  writeln;
  writeln('Root is: ',booter,':');
END;

PROCEDURE setprefix;
BEGIN
 write(cleos,'Set prefix to? ');
 readln(title);
 IF length(title)>0 THEN
  BEGIN
   parsevid(title,volid,physunit,prslt);
   IF prslt=pnull THEN
    BEGIN
     IF pos('#',volid)=1 THEN
      IF unittable[physunit].blkdunit THEN
       BEGIN
        readdir(physunit,iorslt);
        IF iorslt=0 THEN volid:=dirp^[0].dvid
         ELSE flagioerr(iorslt);
       END ELSE volid:=unittable[physunit].unitname;
     IF length(volid)>0 THEN
      BEGIN
       prefix:=volid;
       stringptr.addrs:=prefixaddrs;
       stringptr.contents^:=prefix;
      END;
     writeln;
     writeln('Prefix is ',prefix,':');
    END
    ELSE flagparserr(prslt);
  END;
END;

PROCEDURE zap(volid:vid;physunit:unitrange);
BEGIN
 writeln;
 write('Destroy ',volid,': ?');
 read(keyboard,ch);
 writeln;
 IF ch IN ['Y','y'] THEN
  BEGIN
```

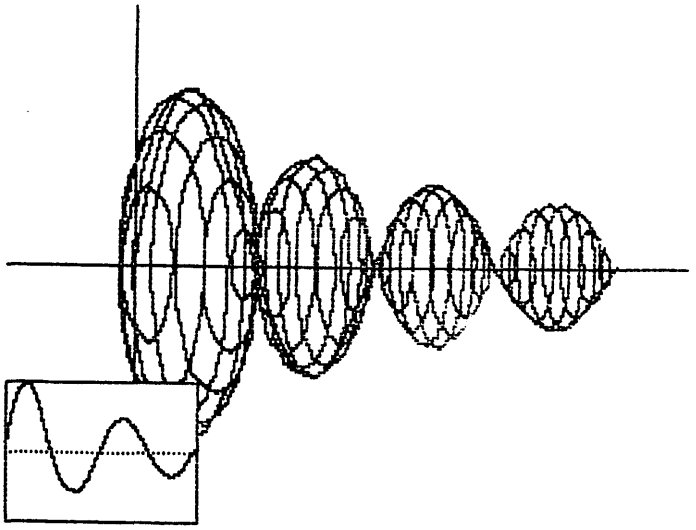
contd.

```
   dirpt[0].dnumfiles:=0;
   unitwrite(physunit,dirpt,sizeof(directory),dirblk);
   writeln('Directory zeroed');
  END
  ELSE writeln('No action. Directory preserved.');
END;

PROCEDURE zero;
BEGIN
 write(cleos,'Zero the directory of? ');
 readln(title);
 parsevid(title,volid,physunit,prslt);
 IF prslt<>pnull THEN flagparserr(prslt)
  ELSE IF foundir(volid,physunit,iorslt) THEN zap(volid,physunit)
  ELSE flagioerr(iorslt);
END;

PROCEDURE Krunch;

 PROCEDURE squash(volid:vid;physunit:unitrange);
 VAR filenum,freeblocks:INTEGER;
     squashflag:BOOLEAN;

  FUNCTION gapsize(entry1,entry0:dirrange;VAR freeblks:INTEGER):INTEGER;
  BEGIN
   freeblks:=dirpt[entry1].dfirstblk-dirpt[entry0].dlastblk;
   gapsize:=freeblks;
  END;

  PROCEDURE move(entry:dirrange;distance:INTEGER;VAR squashflag:BOOLEAN);
  VAR relblk,length:INTEGER;
  BEGIN
   WITH dirpt[entry] DO
    BEGIN
     relblk:=dfirstblk;
     length:=dlastblk-relblk;
     writeln('Moving ',dtid,', ',length,' blocks');
    END;

   (*$R-  **************)
   REPEAT
    IF length>buffblkcount THEN gindex:=fblksize*buffblkcount
     ELSE gindex:=fblksize*length;
    unitread(physunit,heapt[0],gindex,relblk);
    unitwrite(physunit,heapt[0],gindex,relblk-distance);
    relblk:=relblk+buffblkcount;
    length:=length-buffblkcount;
   UNTIL length<=0;
   (*$R+  ******************)

   WITH dirpt[entry] DO
    BEGIN
     dfirstblk:=dfirstblk-distance;
     dlastblk:=dlastblk-distance;
    END;
   squashflag:=TRUE;
  END;

 BEGIN squash
  squashflag:=FALSE;
```

contd.

```
  FOR filenum:=1 TO dirpt[0].dnumfiles DO
   IF gapsize(filenum,filenum-1,freeblocks)>0
    THEN move(filenum,freeblocks,squashflag);
  IF squashflag THEN unitwrite(physunit,dirpt,sizeof(directory),dirblk);
  writeln(volid,':',' crunched' );
 END;

BEGIN krunch
 write(cleos,'Crunch what volume? ');
 readln(title);
 IF length(title)>0 THEN
  BEGIN
   parsevid(title,volid,physunit,prslt);
   IF prslt<>pnull THEN flagparserr(prslt)
    ELSE IF foundir(volid,physunit,iorslt) THEN squash(volid,physunit)
    ELSE flagioerr(iorslt);
  END;
END;

PROCEDURE remove;

 PROCEDURE delete(volid:vid;physunit:unitrange;fileid:tid);
 VAR x,y:STRING;
     ieql:INTEGER;
     flag:BOOLEAN;

  PROCEDURE dropfiles;
  VAR K:INTEGER;
  BEGIN
   page(output);
   writeln('Endangered file(s) on ',volid,':');
   writeln;
   FOR gindex:=1 TO dirpt[0].dnumfiles DO
    WITH dirpt[gindex] DO
     IF status THEN writeln(dtid);
   writeln;
   write('Remove file(s) and update directory?' );
   read(keyboard,ch);
   writeln;
   IF ch IN ['Y','y'] THEN
    BEGIN
     writeln;
     FOR gindex:=dirpt[0].dnumfiles DOWNTO 1 DO
      IF dirpt[gindex].status THEN
       BEGIN
        writeln('Removing ',dirpt[gindex].dtid);
        dirpt[gindex].dtid:=null;
        FOR K:=gindex+1 TO dirpt[0].dnumfiles DO
         dirpt[K-1]:=dirpt[K];
        dirpt[0].dnumfiles:=dirpt[0].dnumfiles-1;
       END;
     unitwrite(physunit,dirpt,sizeof(directory),dirblk);
    END
    ELSE writeln('No action taken');
  END;

 BEGIN delete
  ieql:=pos('=',fileid);
  x:=fileid;
  y:=fileid;                                                    contd.
```

```
  IF ieql=0 THEN
   FOR sindex:=1 TO dirpt[0].dnumfiles DO
    WITH dirpt[sindex] DO
      status:=fileid=copy(dtid,1,length(dtid))
   ELSE BEGIN
    x:=copy(fileid,1,ieql-1);
    y:=copy(fileid,ieql+1,length(fileid)-ieql);
    IF (length(x)=0) AND (length(y)=0)
     THEN BEGIN zap(volid,physunit);exit(delete) END
     ELSE FOR sindex:=1 TO dirpt[0].dnumfiles DO
           WITH dirpt[sindex] DO
            status:=(x=copy(dtid,1,length(x)))
                                AND
                    (y=copy(dtid,length(dtid)-length(y)+1,length(y)));
   END;

  flag:=FALSE;
  sindex:=dirpt[0].dnumfiles;
  WHILE (NOT flag) AND (sindex>0) DO
   BEGIN
    flag:=dirpt[sindex].status;
    sindex:=sindex-1;
   END;

  IF flag THEN dropfiles
   ELSE writeln('Can''t find ',concat(volid,':',fileid));
 END;

BEGIN remove
 write(cleos,'Remove what file(s)? ');
 readln(title);
 IF length(title)>0 THEN
  BEGIN
   parsetidvid(title,volid,physunit,fileid,prslt);
   IF prslt<>pnull THEN flagparserr(prslt)
    ELSE IF foundir(volid,physunit,iorslt) THEN delete(volid,physunit,fileid)
    ELSE flagioerr(iorslt);
  END;
END;

PROCEDURE setfree;
BEGIN
 release(heap);
 release(dirheap);
END;

BEGIN (* minifiler *)
 reservespace;
 page(output);
 REPEAT
  prompt(ch);
  CASE ch OF
   'l','L':listing;
   'p','P':setprefix;
   'v','V':volumes;
   'z','Z':zero;
   'k','K':Krunch;
   'r','R':remove;
  END;
 UNTIL ch IN ['q','Q'];
 setfree;
END;


BEGIN
 initfiler;
END.
```



MAHONEY SIN(2X)/(X/2+1)

HOWIE MITCHELL
408 JACKSON AVENUE
LEXINGTON, VIRGINIA
24450

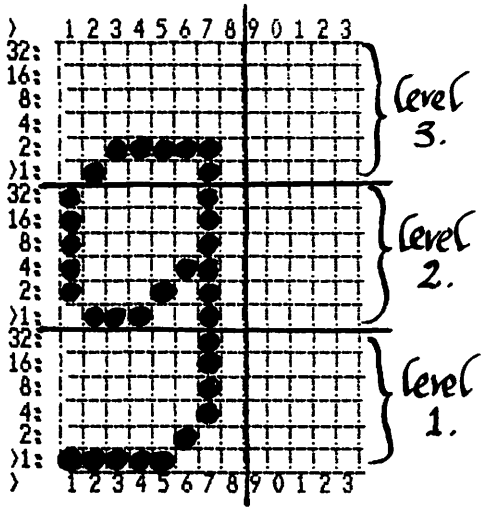
January 22, 1981

Dear Mr. Editor,

Several years ago, *I experienced the thrill of learning a bit about Chancery Cursive script.*

I recently acquired one of the magnificent Anadex DP-9501 printers, which offers a graphics capability. For several months, I have been intrigued with writing a program which will translate ordinary text into this fancy script. I was held back by knowing that this would require an enormous amount of shape-detail for each letter (e.g. 30 data bits for one letter; or about 780 separate amounts for the alphabet - upper & lower case). It seemed to be too much of an undertaking, until I tried my hand at designing JUST ONE LETTER. By shifting my attention from the entire-job-all-at-once to a small portion of it, the work became ordered, relatively pleasant, and not too time-consuming.

By printing out little guide grids (see below), the encoding of shape - to binary - to decimal data became easy and aesthetically pleasing to do.



It is most curious to realize that, although I'd intended to share with you the delight of the <u>experience</u>, all I have to show for it is a complex program listing, and a few words and a diagram.

I also have an increased respect and admiration for the elegant HEADLINE LETTERS in the Apple Pi magazine. They're <u>handsome</u>! My best to whoever designs them.

Sincerely,

Howie

contd.

*** RIGHT-JUSTIFIED SCRIPT WRITER ***

```
10  TEXT : HOME
15  HTAB 6
    PRINT "*** CURSIVE TEXT WRITER ***                    ------------------"
30  PRINT : PRINT "  THIS PROGRAM WILL TRANSLATE ORDINARY  LETTERS INTO RIGHT-JUSTIFIED SCRIPT     (SIMILAR TO CHANCARY
CURSIVE WRITING).": PRINT
35  PRINT "  SINGLE CAPITALS ARE INITIATED BY PRE- FACING THE LETTER WITH CONTROL B. GROUPSOF CAPITALS APPEAR IF CONTROL
 A IS USED (GROUP-CAPS ARE TURNED OFF BY A SECOND  CONTROL A).": PRINT
40  PRINT "  THE SIZE OF THE LETTERS VARIES, BUT 64IS ABOUT THE LIMIT PER LINE.  ONE  PAGE WILL HOLD A MAXIMUM OF 45 LIN
ES OF TEXT."
50  PRINT "  CHARACTERS AVAILABLE ARE THE LETTERS  OF THE ALPHABET (UPPER & LOWER CASE),   THE NUMERALS, AND THE FOLLOWI
NG PUNCTU- ATION MARKS:  .,-'!()?:;"
52  PRINT
55  PRINT "-INITIALIZING- (ONE MOMENT PLEASE.)"
100  REM *************************     CHARACTER DATA       ******************************
105  REM ************************* DATA FORMAT:             ** # VERT. STRIPS, LOWEST   ** LEVEL, DATA FOR TOP,     ** -
SPACE- DATA FOR MIDDLE, ** -SPACE- DATA FOR BOTTOM  ** (IF NEEDED).            ******************************
120  DATA 72: REM ***************** THIS INCLUDES THE LETTERS** (UPPER & LOWER CASE), THE** NUMERALS, AND PUNCTUATION** M
ARKS: .,-'!()?:;        ******************************
123  REM *************************    LOWER CASE LETTERS    ******************************
125  DATA 11,2,0,1,2,2,2,2,3,0,0,0,0,    62,1,1,1,1,2,63,1,2,4,8: REM : LOWERCASE A.
130  DATA 8,2,15,17,34,34,34,34,33,0,     63,1,1,1,1,2,60,0: REM : LOWERCASE B.
135  DATA 9,2,0,1,2,2,2,2,1,0,0,     60,2,1,1,1,1,34,4,8: REM : LOWERCASE C.
140  DATA 11,2,0,1,2,2,2,2,31,32,32,32,32,     62,1,1,1,1,2,63,1,2,4,8: REM : LOWERCASE D.
145  DATA 9,2,0,1,2,2,2,2,1,0,0,     62,9,9,9,9,17,34,4,8: REM : LOWERCASE E.
150  DATA 8,1,0,0,0,15,16,32,32,32,     16,16,16,63,16,16,16,16,     1,1,1,62,0,0,0,0: REM : LOWERCASE F.
155  DATA  8,1,0,1,2,2,2,2,3,0,     62,1,1,1,2,4,63,0,     1,1,1,1,1,2,60,0: REM  : LOWERCASE G.
160  DATA 11,2,15,16,33,34,34,34,33,0,0,0,0,     63,32,0,0,0,0,63,1,2,4,8: REM : LOWERCASE H.
     DATA 7,2,0,0,27,0,0,0,0,     16,32,62,1,2,4,8: REM : LOWERCASE I.
170  DATA 6,1,0,0,27,0,0,0,     16,32,63,0,0,0,     1,1,62,0,0,0: REM : LOWERCASE J.
175  DATA 10,2,15,17,34,34,34,34,33,0,0,0,     63,4,4,4,4,10,49,2,4,8: REM : LOWERCASE K.
180  DATA 6,2,15,16,32,32,32,32,     63,1,1,2,4,8: REM : LOWERCASE L.
185  DATA 14,2,0,0,3,1,2,2,1,1,2,2,1,0,0,0,  16,32,63,0,0,0,63,0,0,0,63,2,4,8: REM : LOWERCASE M.
190  DATA 13,2,0,0,3,0,1,2,2,2,1,0,0,0,0,16,32,63,32,0,0,0,0,63,1,2,4,8: REM  : LOWERCASE N.
195  DATA 8,2,0,1,2,2,2,1,0,0,     60,2,1,1,1,2,60,0: REM : LOWERCASE O.
200  DATA 10,1,0,0,3,0,1,2,2,2,1,0,     16,32,63,34,1,1,1,1,62,0,     1,1,62,0,0,0,0,0,0,0: REM : LOWERCASE P.
205  DATA 8,1,0,1,2,2,2,2,3,0,     62,1,1,1,2,4,63,0,     0,0,1,3,5,9,49,2: REM : LOWERCASE Q.
210  DATA 8,2,0,0,3,0,1,2,2,1,     16,32,63,32,0,0,0,0: REM : LOWERCASE R.
215  DATA 8,2,1,2,2,2,2,2,1,0,     38,17,17,17,17,17,14,0: REM : LOWERCASE S.
220  DATA 9,2,2,10,18,63,2,2,2,0,0,     0,0,0,62,1,1,2,4,8: REM : LOWERCASE T.
225  DATA 13,2,0,0,3,0,0,0,0,0,3,0,0,0,0,     16,32,62,1,1,1,1,2,62,1,2,4,8: REM : LOWERCASE U.
230  DATA 11,2,0,0,3,0,0,0,0,0,0,3,2,3,     16,32,56,4,2,1,2,4,56,0,0: REM : LOWERCASE V.
235  DATA 12,2,0,0,3,0,0,0,0,0,0,0,3,2,     16,32,62,1,1,2,28,2,1,1,62,0: REM : LOWERCASE W.
240  DATA 11,2,0,0,2,0,0,0,0,1,2,0,0,     17,33,34,20,8,20,34,1,2,4,8: REM : LOWERCASE X.
245  DATA 10,1,0,0,3,0,0,0,0,0,3,0,     16,32,62,1,1,1,1,2,63,0,     0,1,1,1,1,1,1,2,60,0: REM : LOWERCASE Y.
250  DATA 8,2,3,2,2,2,2,3,2,0,     19,21,25,17,49,17,3,0: REM : LOWERCASE Z.
255  REM *************************    UPPER CASE LETTERS    ******************************
260  DATA  14,2,0,0,16,32,35,44,48,15,0,0,0,0,0,0,     1,3,13,49,16,16,16,16,62,3,1,1,2,0: REM : CAP A.
265  DATA 13,2,16,32,32,63,33,33,33,33,33,17,14,0,0,     1,1,1,63,1,1,1,1,1,1,34,28,0: REM  : CAP B.
270  DATA 10,2,15,16,32,32,32,32,32,16,56,0,     60,2,1,1,1,1,1,2,7,0: REM : CAP C.
275  DATA  13,2,16,32,32,63,32,32,32,32,32,16,8,7,0,     1,1,1,63,1,1,1,1,1,2,4,56,0: REM  : CAP D.
280  DATA  12,2,16,32,32,63,33,33,33,33,32,48,0,0,     1,1,1,63,1,1,1,1,1,1,3,0: REM  : CAP E.
285  DATA  11,2,16,32,32,63,33,33,33,33,32,48,0,     0,0,1,63,1,0,0,0,0,0,0: REM  : CAP F.
290  DATA 10,1,15,16,32,32,32,32,32,16,56,0,    60,2,1,1,1,49,33,34,63,0,     0,0,3,3,1,1,1,2,60,0: REM : CAP G.
     DATA  14,2,16,32,32,63,33,1,1,1,1,33,63,32,32,0,     0,1,1,63,1,0,0,0,0,1,63,1,1,0: REM  : CAP H.
305  DATA  9,2,0,0,16,32,32,32,32,63,0,     1,1,1,1,1,1,2,60,0: REM  CAP I.
310  DATA  10,1,0,0,16,32,32,32,32,32,63,0,     0,0,0,0,0,0,0,0,63,0,     1,1,1,1,1,1,1,2,60,0: REM  : CAP J.
315  DATA  13,2,16,32,32,32,63,1,2,4,8,16,32,32,0,     0,1,1,1,62,32,16,8,4,2,1,1,0: REM  : CAP K.
```


contd.

```
320 DATA 11,2,0,0,15,16,32,32,32,32,0,0,0,     1,3,61,1,1,1,1,1,1,2,0: REM : CAP L.
325 DATA 15,2,24,32,32,63,12,2,1,0,1,2,12,63,32,32,0,     0,1,1,63,1,0,0,48,0,0,1,63,1,1,0: REM : CAP M.
330 DATA 14,2,16,32,32,63,24,6,1,0,0,0,31,32,32,0,     1,1,1,62,0,0,32,16,12,2,63,1,1,0: REM : CAP N.
335 DATA 10,2,15,16,32,32,32,32,32,16,15,0,     60,2,1,1,1,1,1,2,60,0: REM : CAP O.
340 DATA 12,2,16,32,32,63,32,32,32,32,32,17,14,0,     0,0,1,63,33,32,32,32,32,0,0,0: REM : CAP P.
345 DATA 11,2,15,16,32,32,32,32,32,16,15,0,0,     60,2,1,9,9,9,5,2,61,1,1: REM : CAP Q.
350 DATA 13,2,16,32,32,63,32,32,32,32,32,17,14,0,0,     1,1,1,63,33,32,32,32,56,4,2,1,1: REM : CAP R.
355 DATA 11,2,14,17,33,33,33,33,33,17,56,0,0,     7,2,1,1,1,1,1,1,34,28,0: REM : CAP S.
360 DATA 11,2,48,32,32,32,32,63,32,32,32,32,48,     0,0,0,0,1,63,1,0,0,0,0: REM : CAP T.
365 DATA 14,2,16,32,32,63,0,0,0,0,0,0,0,63,32,32,     0,0,0,60,2,1,1,1,1,1,2,60,0,0: REM : CAP U.
370 DATA 13,2,16,32,32,62,33,0,0,0,0,0,33,62,32,     0,0,0,0,32,24,6,1,6,24,32,0,0: REM : CAP V.
375 DATA 18,2,16,32,32,60,3,0,0,0,0,0,7,0,0,0,0,0,3,28,32,32,     0,0,0,0,48,15,2,12,48,0,48,12,2,15,48,0,0,0: REM : CAP W.
380 DATA 13,2,16,32,32,56,36,2,1,1,2,36,56,32,0,     0,0,1,7,9,16,32,32,16,9,7,1,0: REM : CAP X.
385 DATA 13,2,16,32,32,56,36,2,1,0,1,2,36,56,32,     0,0,0,0,0,0,1,63,1,0,0,0,0: REM : CAP Y.
390 DATA 11,2,48,33,33,33,33,35,37,41,48,0,0,     7,9,17,33,1,1,1,1,1,3,0: REM : CAP Z.
400 REM ************************     PUNCTUATION MARKS    **        ,.-'!()?:;        ****************************
405 DATA 6,1,0,0,0,0,0,0,     0,0,3,3,0,0,     0,8,16,32,0,0: REM : COMMA.
410 DATA 6,2,0,0,0,0,0,0,     0,0,3,3,0,0: REM : PERIOD.
415 DATA 15,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,     0,0,0,4,8,24,24,24,24,24,16,32,0,0,0: REM : HYPHEN.
420 DATA 5,2,0,52,56,0,0,     0,0,0,0,0: REM : APOSTROPHE.
425 DATA 8,2,0,0,0,63,63,0,0,0,     0,0,0,51,51,0,0,0: REM : EXCLAMATION MARK.
430 DATA 10,2,0,0,0,0,7,8,16,32,0,0,0,     0,0,0,56,4,2,1,0,0,0: REM : L PARENTHESIS.
435 DATA 10,2,0,0,0,0,32,16,8,7,0,0,0,     0,0,0,1,2,4,56,0,0,0: REM : R PARENTHESIS.
440 DATA 14,2,0,0,0,0,0,12,16,16,16,16,17,14,0,0,0,     0,0,0,0,0,0,0,0,27,32,0,0,0,0,0: REM : QUESTION MARK.
450 REM ************************          NUMERALS          ****************************
452 DATA 11,2,7,8,16,16,16,17,10,7,0,0,0,     60,6,9,17,33,1,2,60,0,0,0: REM : #0.
454 DATA 7,2,4,8,31,0,0,0,0,     1,1,63,1,1,0,0: REM : #1.
456 DATA 11,2,4,8,16,16,16,16,16,9,6,0,0,0,     15,17,33,33,33,33,1,3,0,0,0: REM : #2.
458 DATA 11,2,4,8,16,16,16,16,16,9,6,0,0,0,     4,2,33,33,33,33,18,12,0,0,0: REM : #3.
460 DATA 12,2,0,0,1,2,4,8,31,0,0,0,0,0,     16,48,16,16,16,16,63,16,16,0,0,0: REM : #4.
462 DATA 11,2,31,17,17,17,17,17,16,0,0,0,0,     2,1,1,1,1,1,34,28,0,0,0: REM : #5.
464 DATA 11,2,7,8,17,17,17,17,8,0,0,0,0,     60,34,1,1,1,1,34,28,0,0,0: REM : #6.
466 DATA 10,2,24,16,16,16,17,18,28,0,0,0,     0,0,31,32,0,0,0,0,0,0: REM : #7.
468 DATA 11,2,6,9,16,16,16,16,16,9,6,0,0,0,     12,18,33,33,33,33,18,12,0,0,0: REM : #8.
470 DATA 11,2,7,8,16,16,16,16,16,8,7,0,0,0,     0,34,17,17,17,17,34,60,0,0,0: REM : #9.
475 REM ************************     COLON & SEMI-COLON     ****************************
480 DATA 7,2,0,0,1,3,1,0,0,     0,0,2,39,2,0,0: REM : COLON.
485 DATA 6,1,0,0,0,0,0,0,0,     0,0,27,27,0,0,     0,8,16,32,0,0: REM : SEMI-COLON.
500 REM ************************     INITIALIZE STRINGS.    ****************************
505 BLANK$ = "@@@@@@@@@@@@@@@@@@@@"
510 REM ************************ FORMAT FOR STRINGS:      ** LTR$(CHARACTER #, LEVEL) ****************************
515 READ CHARS: DIM LTR$(CHARS,3),SIZE(CHARS)
520 FOR LTR = 1 TO CHARS
525 READ SIZE,LIMIT:SIZE(LTR) = SIZE: IF LIMIT = 2 THEN LTR$(LTR,1) =  LEFT$ (BLANK$,SIZE): REM : CREATE GRAPHICAL BLAN
K STRINGS FOR LETTERS HAVING NO DESCENDERS.
530 FOR LEVEL = 3 TO LIMIT STEP  - 1
535 FOR N = 1 TO SIZE
540 READ DAYTA:LTR$(LTR,LEVEL) = LTR$(LTR,LEVEL) +  CHR$ (64 + DAYTA): NEXT N
545 NEXT LEVEL
550 NEXT LTR
560 HTAB 1
570 VTAB 23: PRINT  CHR$ (7);"(PRESS ANY KEY TO CONTINUE.)        ";: GET A$
600 REM ************************     GET INPUT TEXT       ****************************
602 HOME
605 DIM MESSAGE$(45)
610 PRINT "  THERE IS ROOM FOR ABOUT 45 LINES OF   TEXT. TYPE IN '*' TO BEGIN PRINTOUT."
615 PRINT : PRINT "PUNCTUATION AVAILABLE = ,.-'!()?:;": PRINT
617 PRINT "  USE ')' TO INDENT FOR ADDRESS HEADING."
620 FOR N = 1 TO 45: PRINT "LINE #";N;":": FOR DASH = 1 TO 64: PRINT "-";: NEXT DASH: PRINT
625 VTAB  PEEK (37) - 1: INPUT MESS$(N)
```

contd.

```
630  IF ME$(N) = "*" THEN 640
635  NEXT N
640 EOF = N - 1:ACAP =  - 1: FOR N = 1 TO 5: NEXT N
699  REM ************************      PRINT RESULTS      ****************************
700  PR# 1: PRINT  CHR$ (28)
702 BLANK$ = "@@@@@@@@@@@@@@@@@@@@": REM : 20 @'S FOR RIGHT-JUSTIFYING.
705  FOR MESS = 1 TO EOF:MESSAGE$ = MESSAGE$(MESS)
707 ACAP =  - 1
710  IF  LEN (ME$) = 0 THEN  PRINT 9: NEXT MESS
715 SPACE = 0:DOTS = 0: FOR LOOK = 1 TO  LEN (MESS$):A$ =  MID$ (MESS$,LOOK,1):SPACE = SPACE + (A$ = " "):A =  ASC (A$)
718 CAP = 0: IF A = 2 THEN CAP = 1: NEXT LOOK
720  IF A = 1 THEN ACAP = ACAP *  - 1: NEXT LOOK
722  IF A = 32 THEN 730
724  IF (A ) 32 AND A ( 65) THEN  GOSUB 900: GOTO 727
725 A = A - 64
727 CFLAG = (CAP ) 0 OR ACAP ) 0) AND (A ) 0 AND A ( 27):DOTS = DOTS + SIZE(A + 26 * (CFLAG = 1))
730  NEXT LOOK: IF SPACE ) 0 THEN GSPACE = (599 - DOTS) / SPACE: REM : GSPACE WILL BE USED FOR INSERTING RIGHT-JUSTIFYIN
G BLANKS.
732 JUST$ = "@@@@@@@@@": IF GSPACE ( 20 THEN JUST$ =  LEFT$ (BLANK$,GSPACE)
735  FOR LEVEL = 3 TO 1 STEP  - 1
737 ACAP =  - 1
740  FOR N = 1 TO  LEN (MESS$):A$ =  MID$ (MESS$,N,1):A =  ASC (A$): IF A = 32 AND N (  ) LEN (MESS$) THEN  PRINT JUST$
;: NEXT N
745  IF A ) 64 THEN 780
750  IF (A ) 32 AND A ( 65) THEN  GOSUB 900: GOTO 790
760  IF A = 2 THEN CAP = 1: NEXT N
765  IF A = 1 THEN ACAP = ACAP *  - 1: NEXT N
780 A = A - 64 + ((CAP ) 0) OR (ACAP ) 0)) * 26
790  IF A ( 0 THEN A = 0
800  PRINT LTR$(A,LEVEL);:CAP = 0: NEXT N: PRINT 6
820  NEXT LEVEL
825  PRINT 2
830  NEXT MESS
840  PRINT  CHR$ (29)
860  PR# 0
870  END
900  REM ************************ SET A TO FETCH SPECIAL   ** CHARACTERS.            ****************************
910  IF A$ = "," THEN A = 53: RETURN
915  IF A$ = "." THEN A = 54: RETURN
920  IF A$ = "-" THEN A = 55: RETURN
925  IF A$ = "'" THEN A = 56: RETURN
930  IF A$ = "!" THEN A = 57: RETURN
935  IF A$ = "(" THEN A = 58: RETURN
940  IF A$ = ")" THEN A = 59: RETURN
945  IF A$ = "?" THEN A = 60: RETURN
950  IF (A ) 47 AND A ( 60) THEN A = A + 13: RETURN
955  IF A$ = ")" THEN  PRINT ";375";:A = 0: RETURN
1000  PR# 0: HOME : PRINT  CHR$ (7): FOR N = 1 TO 10: HTAB 5: PRINT "*** SOMETHING WENT WRONG! ***": PRINT : NEXT N


26000  REM **********************
    *                        *
    *  HOWIE MITCHELL *      *
    *  408 JACKSON AVE.      *
    *  LEXINGTON, VA. 24450  *
    *  JANUARY, 1981         *
    *                        *
    **************************
```

Character set - upper and lower case letters:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz

Numerals and Punctuation: 1234567890 ,.-'!():;

Happy 1981! Take a byte from your APPLE for good cheer. Perhaps a little APPLE jack...

A quick fill-in on the plans and activities of some of our SIGs and committees.

Ted Perry, Chairman of our Education SIG, is in the process of collecting project information from all State funding offices and is inventorying educational software developed with the support of Federal grants. These include projects funded under Title 4C, Bureau of Education for the Handicapped, National Science Foundation and others. He says that Greg Vanderheisen with TRACE, University of Madison, Wisconsin can input BLISS symbols which are a standardized set of symbols for sign language communication.

James E. Hassler's Ham Radio SIG meets every Sunday night on 14.329 MHZ at 0100 ZULU (8 PM East Coast time) net control WB7TRQ located in Cheyenne, Wyoming. His group boasts a membership of over 200 Ham's. He's getting calls from Sweden, Africa, etc. They run an exchange library and will exchange with any club.

Dave McFarling of Lincoln, Nebraska, (402) 467-1878, has generously picked up the ball I dropped and is now the Chairman of the Handicapped SIG. Dave was one of the feature writers in the October issue of Softalk. He's a quadraplegic who runs a computer business in his home. He's very interested in getting volunteers to handle areas dealing with specific special needs, e.g. deaf/hearing impaired, blind, learning disabled, etc.

Neil Lipson, who is responsible for the IAC Software Exchange, tells me that the software submissions are getting better. Next to come will be an education disk with math and engineering programs. Joe Budge, our Secretary, confirms that this IAC Disk No. 4 has gone to the "mailing house". This will be followed shortly with another utilities disk. Neil needs public domain software with documentation on the disks.

Major Terry N. Taylor, who runs the IAC Newsletter library, reports that the newsletter portion of the library has over 650 newsletters representing some 9000 pages of APPLE material. He's averaging about 3 - 4 requests per week for information from the library. Also available are 305 different articles (1400 pages worth). Listings are available to member clubs and he will provide copies at 6 cents per page. Present address is 1646 Harper Ave., Redondo Beach, Colo. 90278. In his capacity as librarian for the Denver Apple Pi, he will help new clubs by providing them public domain programs recorded on their two blank diskettes. These should be sent either to: Denver Apple Pi Librarian, P.O. Box 17467, Denver, Colo. 80217, or to Terry Taylor at his address listed above.

Randy Field, Chairman for New Club Assistance, will provide some materials, starting this month, and lots more in February. There will be a sample constitution and by-laws, club organization recommendations, suggested officers and committees, as well as ideas for activities and ways to attract new members.

Craig Vaughan of Telecommunications is working together with Mark Robbins' Standards Committee on the development of standards for transferring files. He also reports that SAUG (Source Apple Users Group) is working on the development of on-line libraries with the capability to transfer articles. They are considering publishing an electronic magazine.

Tom Woteki of Washington Apple Pi, (202) 547-0984, has started a Languages SIG. He writes our Pascal "Blaise Away" articles under the pseudonym "Dr. Wo". He's looking for volunteers who will handle APPLE Forth, Pilot, Fortran or whatever. Hopes to have a series of lectures or panels prepared in time for the IAC annual meeting in May in Chicago.

To get additional information on these or other committee activities, check your Winter 1980-1981 Apple Orchard, page 4, for their phone numbers.

Have you heard...?

....We are now officially incorporated within the State of California. Our membership continues to grow - latest count is 200 clubs.

....New subscription address - please note that all subscription requests for the Apple Orchard should now be sent to P.O. Box 1493, Beaverton, Oregon 97075, rather than to Seattle.

....Pascal 1.1 is rumored to be out and available to new Pascal purchasers. Be sure to ask for 1.1, otherwise you may end up with the old version. Manuals are being sent to all Pascal 1.0 registered owners, i.e. those who sent in their warranty cards. Send your master and $15 to Apple, Inc. to get your updated software. The update and manuals will be available for $65 at your local dealers for non-registered owners. Apropos of this is the following summary of the differences between versions 1.0 and 1.1, excerpted with permission from the Apple Pascal Update, Apple Part #030-1084-00.

CHANGES THAT AFFECT OVERALL OPERATION:

- All files from 1.0 are included within 1.1 but are not necessarily interchangeable.
- Text files with more than 40 blocks must be divided using the old Editor prior to being used by Version 1.1.
- All read commands will work much more quickly with 1.1.

VERSION 1.1:

- Includes a system swapping option that allows you to maximize available memory space.

- Allows you to shift the keyboard into lower case.
- Sets up consistent rules on uses of suffix .TEXT or .CODE.
- Modifies the one-drive startup.
- Allows you to create Exec files.
- Provides a SAVE command other than through the Filer.
- Provides new editor prompts.
- Includes expanded find and replace commands.
- Adds optional automatic spacing and improves hyphenation.
- Corrects and improves COPY From and WRITE commands and will no longer clobber your file when the buffer is nearly full.

CHANGES THAT AFFECT USER PROGRAMS:

- Upper and lower case letters are interchangeable.
- A new "V" option to check the length VAR parameters of type STRING.
- Program code file can contain up to 16 segments.
- New "Next Segment" and "SWAPPING" options.
- Automatic return to text mode on termination while in graphics mode.
- Improved LIBRARY utility prompt lines.
- A new UNIT called "CHAINSTUFF" to "chain" to another program.
- LIBRARY.CODE used instead of FORTLIB.CODE.

All this and lots more (witness 31. listed problems for 1.0 which have been fixed in 1.1) which suggests that your $15 buys quite a bit.

....M&R Enterprises says that Supr-Fan is just about ready for marketing and distribution. It's supposed to be whisper quiet.

....RAM card wars - Last summer Andromeda introduced its 16K Expansion Board. Microsoft has just announced a similar board but as of now none have been shipped. We've heard rumors that Apple is about to unbundle the Language Card from Pascal so that it can be purchased separately for about $250.

....Is there a DOS 4.0 in your future? A new APPLE motherboard?

Bernie Urban - Ed.
January 10, 1981

FROM THE INTERNATIONAL APPLE CORE:

This is the EXEC File Creator which should have accompanied the Program List Formatter article in the Winter Apple Orchard.

```
*
JLIST

10 D$ = CHR$ (13) + CHR$ (4): PRINT
     D$"OPENFP LIST"D$"WRITEFP LI
     ST"
15  PRINT "POKE 0,PEEK( 103 ):POKE1
     ,PEEK( 104 ):POKE2,PEEK( 175 ):P
     OKE3,PEEK( 176 )"
20  PRINT "POKE104,PEEK( 176 )"
30  PRINT "IF PEEK( 175 )=255 THEN
     POKE 104,PEEK( 104 )+1"
40  PRINT "POKE103,( PEEK( 175 )+1 )-
     INT(( PEEK( 175 )+1 )/256 )*256"
50  PRINT "POKE PEEK( 103 )+PEEK( 10
     4 )*256-1,0"
60  PRINT "POKE PEEK( 103 )+PEEK( 10
     4 )*256,0"
70  PRINT "POKE PEEK( 103 )+PEEK( 10
     4 )*256+1,0"
80  PRINT "RUN PROGRAM LISTING FO
     RMATTER"
90  PRINT D$"CLOSE"
100  END
```

WASHINGTON APPLE PI
MAIL ORDER FORM

Washington Apple Pi now has a program library, and disks are available for purchase by anyone. The price to members is $5.00 per disk and $8.00 to non-members. These disks are chock full of exceptional programs - the utilities are especially useful. The games are some of the best - not just simple and uninteresting ones. You may pick them up at any meeting or have them mailed for $2.00 per disk additional. (If you order five or more the additional charge will be $10.00 total.) They will come in a protective foam diskette mailer.

PROGRAM DISKETTES

Members: $5.00 picked up at meeting
$7.00 mailed to you (for the first five, remainder at $5.00)

Non-members: $8.00 per disk picked up at meeting
$10.00 mailed to you (for the first five, remainder at $8.00)

Volume 1	Utilities I	( )	Volume 29	Utilities VIII	( )
Volume 2	Utilities II	( )	Volume 30	Games X	( )
Volume 3	Games I	( )	Volume 31	Plot Utilities	( )
Volume 4	Games II	( )	Volume 32	Games XI	( )
Volume 5	Games III	( )	Volume 33	Accounting	( )
Volume 6	Games IV	( )	Volume 34	Solar Tutor	( )
Volume 7	Games V	( )	Volume 35	Garden Management	( )
Volume 8	Utilities III	( )	Volume 100	DOS 3.3 Utilities A	( )
Volume 9	Educational I	( )	Volume 180	Dungeon Designer	( )
Volume 10	Math/Science	( )	Volume 181	Beginner's Cave	( )
Volume 11	Graphics I	( )	*Volume 182	Lair of Minotaur	( )
Volume 12	Games VI	( )	*Volume 183	Cave of the mind	( )
Volume 13	Games	( )	*Volume 184	Zyphur Riverventure	( )
Volume 14	IAC Utilities IV	( )	*Volume 185	Castle of Doom	( )
Volume 15	Games VII	( )	*Volume 186	Death Star	( )
Volume 16	Utilities V	( )	*Volume 187	Devil's Tomb	( )
Volume 17	Graphics II	( )	*Volume 188	Caves of Treas. Isl.	( )
Volume 18	Educational II	( )	*Volume 189	Furioso	( )
Volume 19	Commumications	( )	*Volume 190	The Magic Kingdom	( )
Volume 20	Music	( )	*Volume 191	The Tomb of Molinar	( )
Volume 21	Apple Orchard	( )	*Volume 192	Lost Island of Apple	( )
Volume 22	Utilities VI	( )			
Volume 23	Games VIII	( )			
Volume 24	Games IX	( )			
Volume 25	Utilities VII	( )			
Volume 26	Stocks/Investments	( )			
Volume 27	Math	( )			
Volume 28	Planetfinder	( )			

*Vol. 181 required with these disks.

TOTAL ORDER = $ ------

Check here if you want these shipped---

NAME ----------

ADDRESS ----------

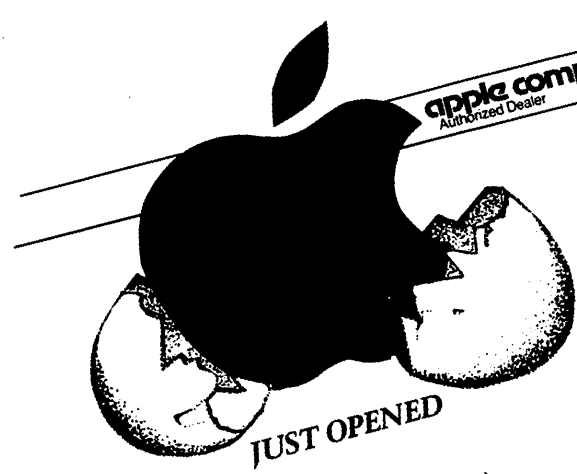
CITY, STATE, ZIP ----------

TELEPHONE ----------

Membership No.(1st three digits after WAP on mailing label) --------

Make checks payable to "Washington Apple Pi"

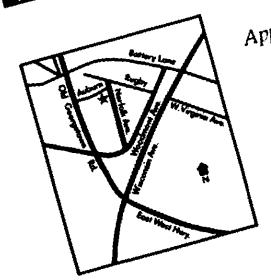
Send order to: Washington Apple Pi- ATTN: Librarian
PO Box 34511
Washington, DC 20034



JUST OPENED

We are pleased to announce
the opening of
Bethesda Computers,
to specialize in the sale of
microcomputers, software,
peripheral equipment & accessories
for use in
home, business and educational
environments.
Major lines will include
Apple, Atari and Hewlett Packard.
Please drop by
for a
"hands-on" demonstration
and professional analysis
of your computer needs

BETHESDA COMPUTERS



Hours
10 am–6 pm Monday thru Saturday
Open till 9 pm on Thursday
8020 Norfolk Ave., Bethesda, MD 20014
(301) 657-1992