$1

Washington Apple Pi

Volume 2 July 1980 Number 7

Highlights

In This Issue

Computerland® and apple II

For the best in personal computing

SOFTAPE™ Personal Software™

D. C. Hayes Associates, Inc.
MICROCOMPUTER PRODUCTS

CENTRONICS®

Mountain Hardware, Inc.

Integral Data Systems, Inc.

MUSE

THE SOURCE
Information Utility

PDI

houston instrument

Heuristics INC.

Automated Simulations

SANYO

Computerland®
We Know Small Computers.
ComputerLand/Tysons Corner
8411 Old Courthouse Road at Rt. 123 — 893-0424

VISA

master charge

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) 725-6281
Members-at-Large	-Scooter Conrad	(301) 725-6251
	-Mark Crosby	(202) 488-1980
	-Sandy Greenfarb	(301) 674-5982
Editor	-Bernard Urban	(above)
Associate Editor	-Rich Wasserstrom	(above)
	-Genevie Urban	(301) 229-3458
Librarian	-David Morganstein	(301) 972-4263

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

Membership dues for Washington Apple Pi are $12.00 per calendar year. If you are interested in joining our club, call our number and leave your name and address. An application form will be mailed to you. Or if you prefer, write us at the above PO Box.

EVENT QUE

Washington Apple Pi meets on the 4th Saturday of each month at 9:30 AM at George Washington University School of Engineering, 23rd and H Streets, N.W. Call the club telephone for exact meeting location. The August meeting will be held on the 23rd.

NOVAPPLE meets on the 2nd Thursday at 7:30PM at Computers Plus in Franconia, and on the 4th Thursday at 7:30 PM at Computerland of Tysons Corner.

CLASSIFIEDS

For Sale; Apple Pascal Board, $400; Apple Serial I/O Board, $150; Apple Communications Board, $175; or all for $700. Call Kirk Balcom, 385-8389.

For Sale: 48K Apple II (integer), cassette based (cassette deck optional), with much software including: data base management system, games, bartender program, Applesoft cassette, and others. Make offer. Hugh Develin, 734-1447.

For Sale: Applesoft Firmware Card. $135.00. Fred Henry, (301) 422-6539

MINUTES

The Washington Apple Pi meeting of June 28, 1980 was called to order at 9:40 am by the Vice President. The new planned meeting structure was announced and suggestions were requested from the membership.

The following items of interest were discussed: Dave Efron discussed and requested assistance in producing abstracts of Apple related articles on a regular basis. Dave Morganstein announced several new additions to the library and methods for stimulating contributions were suggested. Bob Peck pre sented some information on group purchases.

A representative from Acorn publications announced they were preparing a new magazine for Apple owners, and were looking for authors. Chuck (TCA 257) reminded the membership that he was the Apple Source coordinator. Tom Jones, Membership Chairman, asked for names of prospective new members to encourage them to join the club. John Moon announced that he was about to begin testing an Apple Bulletin Board System (ABBS) which he is writing.

The meeting was then turned over to Paul Sand of Computerland (Rockville) for a slide presentation on the Apple III.

Dana J. Schwartz, Secretary

GROUP PURCHASE POWER--DOS 3.3

---A membership benefit---

Those members who now have a Disk II will certainly want to upgrade their systems with DOS 3.3. Normally selling for $60 plus tax, your club will offer it for a substantial discount. Your deposit check for $20 is required before an order can be placed. For further information, call the club number, (301) 468-2305.

NEWSIG

Just bought your Apple? Is it all you can do to turn it on? Join the rest of us at NEWSIG--a special interest group for new Apple owners. Al Weiner, an experienced computer programmer, but a new Apple owner , will lead this group from basics through computer problem solving strategies. Call the club number for further info.

Novapple Minutes For 11 June, 1980

The meeting was called to order by the President at 7:30PM. He announced that Henry Tannenbaum of PM Magazine is looking for people with new and unusual uses for their Apples. He plans to do a show which will discuss the use of computers. If anyone is interested in being on TV and they feel they have some interesting programs to show contact the studio for further details. No date was given for the airing time. Gerald Eskelund indicated he was going to attempt to start a documentation book for Apple Pi and Novapple programs. As the need arises he will call upon members to assist in the documentation. Most of the current programs can be found in Call-Apple and other magazines which are probably available to the members. If the file looks good we may then have a saleable product for the club. Nick Cirrilo made a motion that the club dues be established at $12.00 for the 1980-81 year beginning in July. An amendment was made to include the cost for the Apple Orchard in addition to the dues. The amendment was defeated after some discussion but the basic motion passed with a unanimous vote. Dues will be taken with the meetings beining in July.

The program for the night was a continuation of the Applesoft tutorial presented by Nick Cirillo. He covered such topics as Counting Routines, FOR/NEXT, Loops and READ/DATA/RESTORE. He plans to cover Arrays, DIM and String Manipulation for the next session in July. These sessions are good for both the Novice and the Expert since everyone is expected to contribute to the knowledge of the members. It allows you to show off what you know or listen to the experience of others.

Future meetings will contain the following programs subject to last minute changes as needed:

June 19---Open Forum for Problems
July 9 ---Applesoft Tutorial
July 24---Stockmarket Programs
August 13-Applesoft Tutorial
August 21-Adventure Fest
For up to date notice of what's going on come to the meetings and be part of the happening.

NOVAPPLE NEWS NOTES

As I promised I will try to put newsworthy items in this section of the Newsletter, but you the members will have to help. So far we are batting zero. Two items are worthy of note this time. First we want to document the various programs in Apple Pi's disk library. If you have copies of articles which tell how a program is used or give examples, bring them to the meetings and we will begin to assemble a book for club members. As soon as we have enough to publish I will see that the documentation is made available to our members. Remember if each person brings only one program documentation article we will have over fifty programs. I will anounce at meetings what programs have been gathered and if possible I will publish a list. Duplicates will be made available to members to copy.

The Officers got together and have decided to attempt a Computer Day. The present plans are to have one Saturday a quarter devoted to a demonstration and trading session. As it is planned groups would be set up to discuss topics of interest, while others will be set up to trade programs and build a Novapple program library. Participants will be asked to bring their own machine or make arrangements with someone to share. We will not use a computer store if we can find a suitable place such as a school or large meeting room at reasonable cost. Think about it and let the Officers know what you would like to see.

BLAISE AWAY!

A Primer on Pascal Pointers

by

Dr. Wo

One of the salient features of Pascal which distinguishes it from the other high-level languages available for microcomputers, notably BASIC and FORTRAN, is the variety of data types the language supports. One of these types is the pointer type. Used in conjunction with a user defined record type, pointers make it easy for the programmer to implement dynamic lists, or even trees, with applications from systems programming to list processing. As implemented on the Apple, pointers even permit direct addressing of absolute blocks of memory, a trick which makes saving hi-res graphics images to disk a triviality.

Before dealing with such arcane matters as memory addressing, we first need to familiarize ourselves with pointers and their declarations, pointer assignments and the dynamic allocation of memory.

Declaring Pointers

The type 'pointer' is a simple type, like INTEGER and CHAR, meaning a pointer variable has no substructure as do ARRAYs and RECORDs. Unlike the other simple types however, 'POINTER' is not a standard identifier. Rather, pointers are declared as in the folowing examples:

```
TYPE personrec=RECORD
                 (we will define this type in detail below)
               END;
     link=↑personrec;
VAR  p,q:↑INTEGER;
     alfa:↑CHAR;
     firstperson,lastperson:link;
```

These declarations establish a data type personrec, and several variables p,q,alfa,firstperson and lastperson. p and q are pointers to INTEGERs, alfa is a pointer to CHARacters and first- and lastperson are of type link making them pointers to the type personrec. In fact, p and q can only point to INTEGERs; they are said to be 'bound' to the type INTEGER. Analogous comments apply to the other variabes.

Pointers differ from other data types in a way more significant than the manner in which they are declared. Unlike all the other data types in Pascal, the value of a pointer is hidden from the programmer; only the object pointed to can be accessed. This is because a pointer is an address for another variable or structure and the internal representation of the address is heavily machine dependent, whereas the specifications for Pascal are machine independent. That is, the definition of Pascal attributes no properties to pointers except that they should provide an ability to indirectly refer to other data structures. The only operations permitted with pointers are assignment and comparisons for equality.

Assignments Involving Pointers

The preceeding remarks indicate the need to distinguish between a pointer and its referant, the thing pointed to. Notationally this is accomplished as follows:

```
p  (the pointer p)
p↑ (its referant)
```

This distinction is observed in assignments involving pointers.

For example, suppose the situation is as depicted in Figure 1 where p points to a location whose contents are the integer x (so p↑=x), and q↑=y, and alfa↑='a'. The result of the assignment p↑:=q↑ is depicted in Fig. 2. Note that p still points to the same location; only the contents of the location are changed. Following this the assignment p:=q is depicted in Figure 3. Now p points to a new location-- and the old location is lost unless the job done by p has been taken over by another pointer. The figures emphasize that the location pointed to is anonymous; only the contents of the location can be accessed.

The assignments p:=alfa, p↑:=alfa↑ are never correct because of the binding of pointers in the first case, and the data type mismatch in the second. Likewise the assignments p↑:=q and p:=q↑ are errors.

'NIL' and the Dynamic Allocation of Memory

Sometimes it is convenient to have a pointer point at nothing. This situation typically arises when we wish to mark the end of a dynamic list; beyond the end there is 'nothing'. Pascal provides for this through the assignment of 'NIL' to the value of a pointer as in

```
p:=NIL;
firstperson:=NIL;
```

NIL is a reserved word in UCSD Pascal.

In order to implememt dynamic processing we need a method of allocating space in memory during program execution. Pascal provides for this through the intrinsic procedure 'new'. The call to the procedure new takes the form

```
new(z)
```

where z is any pointer. The effect of new is three-fold:

i) it allocates space for a variable of the type that z points to;

ii) it generates a pointer to this new variable;

iii) it assigns the pointer to the variable z.

The use of new in the sequence

```
p:=NIL;
new(p);
p↑:=x;
```

is depicted in Figures 4, 5 and 6. In words: space for p is allocated at compile time; the assignment p:=NIL sets p to point at nothing (Fig 4); new(p) allocates space for an integer and sets p to point at it; and p↑:=x sets the contents of the location pointed to.

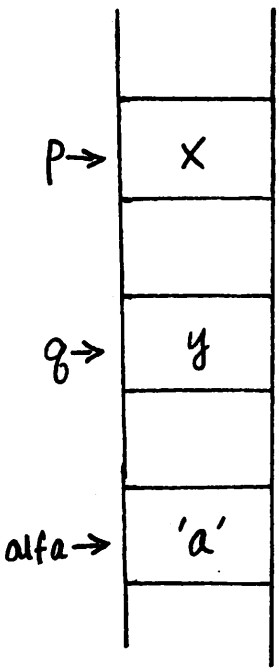

Fig. 1

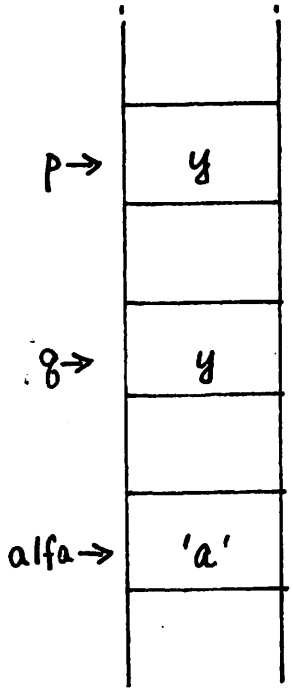

Fig. 2
p↑ := q↑

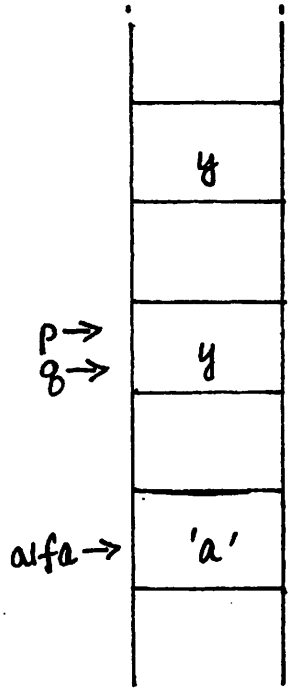

Fig. 3
p := q

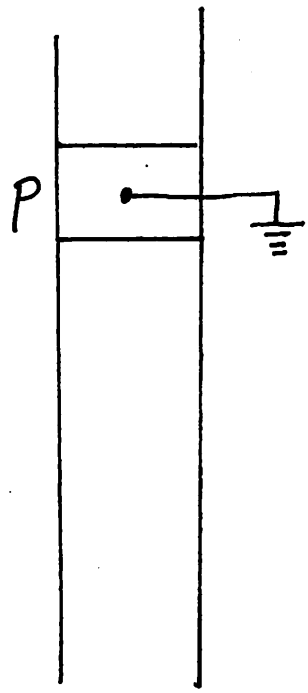

Fig. 4
p := NIL

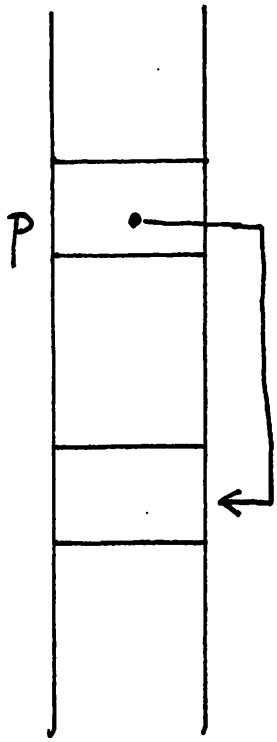

Fig. 5
new(p)

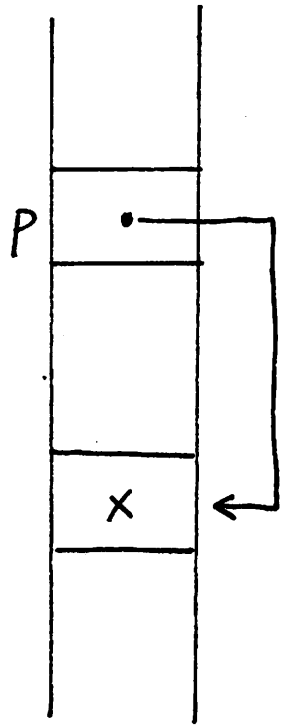

Fig. 6
p↑ := x

A Simple Dynamic List Processing Example

Let us illustrate these concepts with a simple program which creates and prints a dynamic list of names and phone numbers. The global declaration for the type personrec sets up a record type with three fields including 'next', of type link, a pointer to records of type personrec. The pointer field of personrec is used to chain together the records dynamically created by the program.

The procedure 'createlist' kicks off by establishing space for a record pointed to by 'lastperson'. In effect lastperson is used to mark the end of the list. The role of 'firstperson' is to mark the top of the list. Since the list is initially void, firstperson is set to lastperson.

We then enter a REPEAT...UNTIL control loop which assumes that at least one record will be entered. First we allocate space for a new record and collect the information to be stored there; newperson points to the record. Then we set the pointer field of newperson↑ to point to the current top of the list. This forges the link between the new record and the list. Since the new record is now at the top of the list, we adjust firstperson to point at the new record. In other words, we are creating a stack of records in memory linked together by pointers with firstperson playing the role of the pointer to the top of the stack. Lastperson remains fixed and marks the bottom of the stack.

As we enter 'printlist' we know where the two ends of the list are. In order to print the list we simply move down the list following the links in the chain. Note that the length of the list need not be passed between the two procedures and that the two versions of the local variable 'personcount' are used simply to inform the programmer.

The whole program is a simple illustration of first in-last out processing. We could just have easily set it up for first in-first out processing or used the pointers to dynamically sort the list. More on that some other time.

References

Two general references with good treatments of pointers are 'Pascal Programming' by Grogono, Addsison-Wesley; and 'Pascal: an Introduction to Methodical Programming' by Findlay and Watt, Computer Science Press. Both are chock full of examples. A very well written but rather more sophisticated treatment is 'Algorithms+Data Structures=Programs' by Wirth, Prentice-Hall, also full of examples.

Fooling the Compiler or How to Know Where Your'e Pointing

Did I say you don't know the value of a pointer? I did, but you can know, and even set the value of a pointer, by fooling the compiler. The technique, which depends on the variant record facility of Pascal, smacks of trickery which contravenes the spirit of Pascal. But why pass up what can be a handy device just because of that?

That said, I'm going to hold off on this little goodie until next time. The hour is late and Dr. Wo grows weary.

Post-script to Lower Case Input

I gather there was a little bit of confusion over how to implement the programs for lower case input which appeared in my last piece. It seems I committed two of the 13 deadly sins: assuming readers know all (?) I know, and not writing proper documentation. I hope the following sets you straight.

Three program listings were provided: two assembly language programs and one Pascal program. The first assembly listing is for reference for the hard-core and can be ignored as far as implementing lower case. That brings us to the assembly program .PROC SYSGEN. Don't ignore this. In fact, Key in the text of the program, assemble it and svae in a file called SYSGEN, say.

Now key in the Pascal program STARTUP and compile it. Save the results in a file, say STARTUP.

Now run the linker. When it asks for a 'host file', respond STARTUP. When it asks for a 'lib file' answer SYSGEN. When it asks for another lib file, just hit return and when it asks for where to send the linker map, also hit return. Finally, when it asks for the name of your output file respond STARTUP.CODE.

STARTUP.CODE now contains compiled, linked code. Simply execute the code and you will have lower case input and output. If you have any problems, turn off your Apple, turn it on again, and execute STARTUP.CODE again. Voila!

If you want to get lower case automatically each time you power up, simply change the name STARTUP.CODE to SYTEM.STARTUP and deposit the code on the boot disk.

Blaise Away!!!!

Dr. Wo

```
PROGRAM first_in_last_out;
TYPE link=^personrec;
     personrec=RECORD
                 name:STRING;
                 phone:STRING;
                 next:link;
                 END;
VAR  firstperson,lastperson:link;

 PROCEDURE createlist(VAR firstperson,lastperson:link);
 VAR newperson:link;
     personcount:INTEGER;
     answr:CHAR;
 BEGIN
  personcount:=0;

  {set up a pointer to mark the end of the list,
   and set up firstperson to point there as well}
  new(lastperson);
  firstperson:=lastperson;
```

```
 REPEAT
  page(output);
  gotoxy(0,5);

  {create space for a new record}
  new(newperson);

  {enter the new record}
  WITH newperson^ DO
   BEGIN
    writeln(personcount,' records entered.');
    writeln;
    writeln('Enter the record of the next person:');
    writeln;
    write('Name: ');
    readln(name);
    write('Phone: ');
    readln(phone);
   END;

  {insert the new record at the 'top' of the list:
    set the pointer part of the new record to point to
    the record currently at the top of the list,
    then adjust the top-of-list pointer to account for the new entry}
  newperson^.next:=firstperson;
  firstperson:=newperson;

  personcount:=personcount+1;
  writeln;
  write('Enter another record?');
  read(answr);
  writeln;
 UNTIL answr IN ['n','N'];

END;

PROCEDURE printlist(firstperson,lastperson:link);
VAR nextperson:link;
    personcount:INTEGER;

BEGIN
 page(output);
 gotoxy(0,5);
 personcount:=0;

 {set nextperson to point to the top of the list}
 nextperson:=firstperson;

 WHILE nextperson<>lastperson DO
  BEGIN

    personcount:=personcount+1;
    WITH nextperson^ DO
     BEGIN
      writeln('[',personcount:2,'] ',name);
      writeln('      ',phone);
      writeln;
     END;

    {move 'down' the list:
      set nextperson.to point to the next record in the list}
    nextperson:=nextperson^.next;

  END;
END;

BEGIN
 createlist(firstperson,lastperson);
 printlist(firstperson,lastperson);
END.
```

FILEMOVER

by Dana J. Schwartz

The D.C. Hayes Micromodem II opens the world of telecommunications to the Apple owner. However, I found shortly after installing mine that the Micromodem firmware did not universally facilitate the transfer of data (notably tokenized Basic progarms) from one owner to another. It was for this purpose that I wrote Filemover.

This program will allow the exchange of almost any type of disk file (Basic, binary, or text) between two Micromodem II users. Files may be sent in either direction at the rate of about six sectors per minute. A checksum is also included with each sector transmitted in order to detect errors from noise on the telephone lines.

In addition, Filemover is able to "boot" a copy of itself to another Micromodem II. Although this process takes slightly longer (about 15 to 20 minutes), it is usually much faster than physically transporting the program.

Although the Micromodem is well suited for use as a remote terminal and for transmittal of ASCII data, it is not a simple task to transfer a full eight bits of data at a time. Even if the modem registers are configured to send eight bits, the most significant bit always emerges from the firmware as a one. Thus, in order to efficiently transmit the data found in the more common disk files, I had to go outside of the supplied firmware and manipulate the ACIA registers (the heart of the modem) myself.

Basically, a typical transmission will proceed as follows: the transmitter will send the filename, size, and type to the receiver. The receiver will then use DOS to create a dummy file using the given parameters. Finally, the track/sector list is used to read the file sector by sector on the transmit side, machine language routines perform the transmission, and the newly created track/sector list on the receive side is used to store the sectors.

There are only two restrictions on the type of file which may be transmitted. First, files greater than 122 sectors are not accepted, since I did not think they were common enough to require the extra code to link to a second track/sector list. Second, random-access text files which do not have data stored on contiguous virtual sectors will not be transmitted correctly, since I also thought these to be too uncommon to justify the software overhead.

Operating Scenario

1. Both users establish voice contact and determine which will be Originate or Answer. If one user must "boot", go to step 6.

2. Establish modem contact by immediate "pickup" or by entering number (on Originate side) and "answer" (on Answer side).

3. Enter terminal mode and confirm contact. Determine who will "send" or "receive", or "talk" (drop carrier). Exit terminal mode by one user typing [CTRL-A] [CTRL-X] [RTN] to initiate transfer. If "talk", drop carrier, then either stop or go to step 3.

4. Transmitter chooses file from catalog and sends file data (name, type, size) to receiver. Receiver optionally renames file, then dummy file is created on disk.

5. File transfer begins. On completion go to step 3. If an error occurs at any time, report and go to step 3.

6. "Boot": Transmitter informs receiver of proper sequence of commands (given on screen), establishing modem contact. Enter terminal mode. Transmitter starts Boot by [CTRL-A] [CTRL-X] [RTN]. On completion, transmitter goes to step 3 and stops, receiver saves Filemover on disk.

Notes

1. Whenever a filename is requested, a single [RTN] will produce a Catalog and prompt the user again.

2. This program will probably have to be modified to work with the upcoming DOS 3.3.

3. The disk errors printed correspond with the codes listed on page 97 of the DOS 3.2 manual.

4. The LOMEM in line 0 must be poked by hand or always entered before Filemover is executed.

Memory Usage

$800 (2048) RWTS access *
$840 (2112) Input single byte through firmware.
$848 (2120) Output single byte through firmware.
$880 (2176) Transfer data block (type, name, size)--32 bytes.
$900 (2304) Directory, track/sector list buffer.
$A00 (2560) File sector buffer.
$B00 (2816) Output buffer from $A00. *
$B80 (2944) Input buffer to $A00. *
$C00 (3079) LOMEM:

*See listings below

```
]LIST
    0 LOMEM:3072: DIM N$(30): REM MUST
      BE FIRST
   10 CALL -936: VTAB 5: TAB 15: PRINT
      "FILEMOVER": VTAB 9: TAB 4:
      PRINT "HAYES MODEM DISK FILE TR
      ANSFER"
   20 VTAB 13: TAB 10: PRINT "BY DANA
      J. SCHWARTZ": VTAB 15: TAB
      10: PRINT "WASHINGTON APPLE PI"
      : VTAB 22
   30 DIM HEX$(100): GOTO 50
   40 HEX$( LEN(HEX$)+1)=" N E88AG"
      : FOR I=1 TO LEN(HEX$): POKE
      511+I, ASC(HEX$(I)): NEXT I:
      POKE 72,0: RETURN
   50 HEX$="0800: A9 08 A0 12 20 D9 03
      B0 01 60 AD 1F 08 8D 11 08"
      : GOSUB 40: CALL -144
   60 HEX$="0810: 60 00 01 60 01 00 11
      00 24 08 00 09 00 00 01 00"
      : GOSUB 40: CALL -144
   70 HEX$="0820: 00 60 01 00 00 01 EF
      D8": GOSUB 40: CALL -144
   80 HEX$="0840: 20 07 C3 8D 28 08 60
      00 AD 28 08 20 05 C3 60": GOSUB
      40: CALL -144
   90 HEX$="0B00: A2 00 8E 29 08 A9 02
      2C B6 C0 F0 FB BD 00 0A 8D"
      : GOSUB 40: CALL -144
  100 HEX$="0B10: B7 C0 4D 29 08 8D 29
      08 E8 D0 EA 60": GOSUB 40:
      CALL -144
  110 HEX$="0B80: A2 00 8E 29 08 A9 01
      2C B6 C0 F0 FB AD B7 C0 9D"
      : GOSUB 40: CALL -144
  120 HEX$="0B90: 00 0A 4D 29 08 8D 29
      08 E8 D0 EA 60": GOSUB 40:
      CALL -144
  130 DIM M$(5),TP$(8)
  140 TP$="TIA?B???"
  150 D$="": REM CTRL-D
  160 PRINT D$;"NOMON C,I,O"
  170 MSLOT=3: REM MODEM SLOT #
  180 POKE 2114,192+MSLOT: POKE 2125
      ,192+MSLOT
  190 POKE 2824,134+16*MSLOT: POKE
      2832,135+16*MSLOT: POKE 2952
      ,134+16*MSLOT: POKE 2957,135
      +16*MSLOT
  200 GOSUB 7000: GOSUB 7030: PRINT
      : REM INIT MODEM
  210 INPUT "ORIGINATE/ANSWER/BOOT (O/
      A/B)? ",M$
  220 IF M$="O" THEN 370
  230 IF M$#"A" AND M$#"B" THEN 210

  240 BOOT=0: IF M$#"B" THEN 290
  250 BOOT=1: CALL -936: PRINT "INSTRU
      CT RECEIVER:": PRINT
  260 PRINT "  (BOOT DOS)": PRINT
      "  >INT": PRINT "  >IN#'SLOT'"
      : PRINT "  >PR#'SLOT'": PRINT
      "  CTRL-A CTRL-H"
  270 PRINT "  CTRL-A CTRL-Q [‘NUMBER’
      ] RTN": PRINT : PRINT "    (OMIT
      NUMBER IF ‘PICKUP’)"
  280 VTAB 14: PRINT "NOTE:": PRINT
      "    HIMEM: MUST BE IDENTICAL"
      : PRINT "    EXPECT DOUBLE CHARS
      IN TERM MODE": VTAB 20
  290 INPUT "PICKUP/ANSWER (P/A)? "
      ,M$
  300 IF M$="P" THEN 340
  310 IF M$#"A" THEN 290
  320 IF PEEK (-16251+16*MSLOT)>127
      THEN 320: REM WAIT FOR RING
  330 IF PEEK (-16251+16*MSLOT)<128
      THEN 330: REM WAIT TILL END
  340 POKE 1656+MSLOT,128+8+2+1: POKE
      2040+MSLOT,21: POKE -16250+
      16*MSLOT,21: PRINT
  350 IF PEEK (-16250+16*MSLOT) MOD
      8<4 THEN 390
  360 I= PEEK (-16249+16*MSLOT): GOTO
      350
  370 INPUT "NUMBER (RTN IF PICKUP)? "
      ,N$
  380 PRINT "";N$: REM CTRL-Q
  390 CALL -936: GOSUB 7010: GOSUB
      7020: POKE 1912+MSLOT,8+2
  400 INPUT M$
  410 GOSUB 7000: IF M$="" THEN 430
      : REM CTRL-R
  420 GOSUB 7030: PRINT "": GOSUB
      7010: REM CTRL-R
  430 POKE 1912+MSLOT,128+4: POKE
      1784+MSLOT,0
  440 IF BOOT=1 THEN 9000
  450 CALL -936: PRINT "INSERT DISK FO
      R TRANSFER": PRINT : PRINT

  460 MODE=0: REM ASSUME SEND
  470 INPUT "SEND/RECEIVE/TALK (S/R/T)
      ? ",M$
  480 IF M$="S" THEN 1000
  490 IF M$="R" THEN 2000
  500 IF M$#"T" THEN 470
  510 POKE -16251+16*MSLOT,136
  520 INPUT "CONTINUE/HANG UP (C/H)? "
      ,M$
  530 IF M$="H" THEN 570
  540 IF M$#"C" THEN 520
  550 GOSUB 7000: GOSUB 7030: PRINT

  560 GOTO 350
  570 POKE -16251+16*MSLOT,0
  580 END
 1000 PRINT D$;"CATALOG"
```

```
>LIST 1010,30000
 1010 PRINT : INPUT "FILE NAME? "
      ,N$
 1020 IF LEN(N$)<1 THEN 1000
 1030 POKE 2075,9: POKE 2078,1: GOSUB
      6000: REM T/S TO $0900, FILE INF
      O TO $0880
 1040 IF PEEK(2065)#0 THEN 1000
 1050 TPI=TY MOD 8+1
 1060 PRINT "TYPE = '";TP$(TPI,TPI)
      ;"'  SIZE = ";SZ;"  ( ";SZ-
      1;" TO SEND )"
 1070 IF SZ<124 THEN 1090
 1080 PRINT "SIZE > 123": GOTO 1000

 1090 POKE 2088,134: CALL 2120: REM SY
      NC
 1100 GOSUB 3000
 1110 PRINT "SENDING FILE INFO"
 1120 FOR I=0 TO 31: POKE 2088, PEEK
      (2176+I): CALL 2120: NEXT I
 1130 POKE 2075,10: POKE 2078,1: REM R
      EAD TO $0A00
 1140 FOR I=1 TO SZ-1
 1150 TR= PEEK (2314+I*2):SE= PEEK
      (2315+I*2): GOSUB 5000: REM READ
       SECTOR
 1160 GOSUB 3000
 1170 IF PEEK(2065)#0 THEN 1900:
       REM ERROR ON READ
 1180 PRINT "SENDING SECTOR ";I
 1190 CALL 2816: REM SEND SECTOR
 1200 POKE 2088, PEEK (2089)/16: CALL
      2120: POKE 2088, PEEK (2089
      ) MOD 16: CALL 2120: REM SEND CK
      SUM
 1210 NEXT I
 1220 GOSUB 3000
 1230 PRINT : PRINT "TRANSFER COMPLETE
      ": PRINT
 1240 INPUT "HIT RETURN TO CONTINUE IN
       TERM MODE",M$
 1250 GOTO 390
 1900 PRINT : PRINT "SENDING BAD SECTO
      R AND CKSUM"
 1910 CALL 2816
 1920 POKE 2088,255: CALL 2120: CALL
      2120
 1930 CALL 2112: IF PEEK (2088)<>
      255 THEN 1930
 1940 GOTO 8010
 2000 MODE=1: GOSUB 3000: REM SYNC
 2010 POKE 2088,134: CALL 2120
 2020 PRINT "RECEIVING FILE INFO"

 2030 FOR I=0 TO 31: CALL 2112: POKE
      2176+I, PEEK (2088): NEXT I:
       REM GET FILE DATA
 2040 TYR= PEEK (2176):SZR= PEEK
      (2207)-128: REM ALWAYS LOCKED
 2050 FOR I=0 TO 29: POKE 3077+I,
       PEEK (2177+I): NEXT I: POKE
      3107,30: REM FILE NAME TO N$
 2060 IF SZR>1 THEN 2080
 2070 PRINT "BAD FSI RCVD": GOTO
      8000
 2080 PRINT "FILE TO TRANSFER : "
      ,N$: PRINT
 2090 INPUT "RENAME (Y/N)? ",M$
 2100 IF M$="N" THEN 2150
 2110 IF M$#"Y" THEN 2090
 2120 PRINT D$;"CATALOG"
 2130 PRINT : INPUT "NEW FILE NAME? "
      ,N$
 2140 IF LEN(N$)<1 THEN 2120
 2150 TPI=TYR MOD 8+1
 2160 PRINT "ALLOTTING FILE : ";N$
      : PRINT "TYPE = '";TP$(TPI,
      TPI);"'  SIZE = ";SZR;"  ( "
      ;SZR-1;" TO SEND )"
 2170 PRINT D$;"OPEN ";N$;",L256"

 2180 FOR I=0 TO SZR-2: PRINT D$;
      "WRITE ";N$;",R";I: PRINT N$
      : NEXT I
 2190 PRINT D$;"CLOSE ";N$
 2200 POKE 2075,9: POKE 2078,1: GOSUB
      6000: REM GET T/S IN $0900; UPDA
      TE TYPE
 2210 IF PEEK (2065)#0 THEN 8000
 2220 POKE 2075,10: POKE 2078,2: REM W
      RITE FROM $0A00
 2230 FOR I=1 TO SZR-1
 2240 TR= PEEK (2314+I*2):SE= PEEK
      (2315+I*2)
 2250 IF TR>2 AND TR<35 AND TR<>17
       AND SE<13 THEN 2270
 2260 PRINT "BAD TR/SE PAIR": GOTO
      8000
 2270 POKE 2088,134: CALL 2120: REM SE
      ND ACK
 2280 PRINT "RECEIVING SECTOR ";I
 2290 CALL 2944: REM RCVE SECTOR
 2300 CALL 2112:CK= PEEK (2088) MOD
      16: CALL 2112:CK=CK*16+( PEEK
      (2088) MOD 16): REM GET CKSUM
 2310 IF CK= PEEK (2089) THEN 2330

 2320 PRINT "BAD CHECKSUM": GOTO
      8000
 2330 GOSUB 5000: REM WRITE SECTOR
 2340 IF PEEK (2065)#0 THEN 8000
 2350 NEXT I
 2360 POKE 2088,134: CALL 2120: REM SE
      ND ACK
 2370 PRINT : PRINT "TRANSFER COMPLETE
      ": PRINT
```

```
2380 INPUT "HIT RETURN TO CONTINUE IN
     TERM MODE",M$
2390 GOTO 390
3000 CALL 2112
3010 IF PEEK (2088)=255 THEN 8000

3020 IF PEEK (2088)<>134 THEN 3000

3030 RETURN
5000 POKE 2070,TR: POKE 2071,SE
5010 POKE 2065,0
5020 CALL 2048
5030 IF PEEK (2065)=0 THEN RETURN

5040 PRINT : PRINT "DISK ERROR "
     ; PEEK (2065)
5050 PRINT "TRACK = ";TR;"  SECTOR =
     ";SE
5060 RETURN
6000 TR=17:SE=12: GOSUB 5000: REM DIR
     ECTORY
6010 IF PEEK (2065)#0 THEN RETURN
     : REM ERROR
6020 ST=2304:LN= LEN(N$)
6030 FOR I=ST+13 TO ST+223 STEP
     35
6040 IF PEEK (I-2)=0 THEN 6260
6050 FOR J=1 TO LN
6060 IF PEEK (I+J)<> ASC(N$(J)) THEN
     6200
6070 NEXT J
6080 IF LN=30 THEN 6120
6090 FOR J=LN+1 TO 30
6100 IF PEEK (I+J)<>160 THEN 6200

6110 NEXT J
6120 TY= PEEK (I):SZ= PEEK (I+31
     )
6130 FOR J=0 TO 31: POKE 2176+J,
      PEEK (I+J): NEXT J
6140 IF MODE=0 OR TY=TYR THEN 6180

6150 POKE I,TYR
6160 POKE 2078,2: GOSUB 5000: POKE
     2078,1: REM UPDATE TYPE
6170 IF PEEK (2065)#0 THEN RETURN
     : REM ERROR
6180 TR= PEEK (I-2):SE= PEEK (I-
     1): GOSUB 5000: REM T/S LIST
6190 RETURN
6200 NEXT I
6210 TR= PEEK (ST+1):SE= PEEK (ST+
     2)
6220 IF TR=0 AND SE=0 THEN 6260
6230 GOSUB 5000: REM NEXT DIR SECT
6240 IF PEEK (2065)#0 THEN RETURN
     : REM ERROR
6250 GOTO 6030
6260 PRINT "FILE NOT FOUND"
6270 POP : IF MODE=1 THEN 8000
6280 FOR I=1 TO 1000: NEXT I
6290 GOTO 1000
7000 PRINT D$;"IN#0": RETURN
7010 PRINT D$;"PR#0": RETURN
7020 PRINT D$;"IN#";MSLOT: RETURN

7030 PRINT D$;"PR#";MSLOT: RETURN

8000 POKE 2088,255: CALL 2120: REM ER
     ROR CODE
8010 PRINT : PRINT "ABNORMAL TERMINAT
     ION": PRINT
8020 IF MODE=0 OR SZR<2 THEN 8080

8030 INPUT "DELETE NEW FILE (Y/N)? "
     ,M$
8040 IF M$="N" THEN 8080
8050 IF M$#"Y" THEN 8030
8060 PRINT D$;"UNLOCK ";N$
8070 PRINT D$;"DELETE ";N$
8080 GOSUB 7000: GOSUB 7010
8090 PRINT : INPUT "HIT RETURN TO CON
     TINUE IN TERM MODE",M$
8100 GOTO 390
9000 CALL -936
9010 STRH= PEEK (203):STRL= PEEK
     (202)/8*8
9020 STPH= PEEK (77):STPL=( PEEK
     (76)+7)/8*8
9030 IF STPL<256 THEN 9040:STPH=
     STPH+1:STPL=STPL-256
9040 GOSUB 7030: PRINT "POKE 203,"
     ; PEEK (203): GOSUB 9500
9050 GOSUB 7030: PRINT "POKE 202,"
     ; PEEK (202): GOSUB 9500
9060 GOSUB 7030: PRINT "CALL -151"
     : GOSUB 9500
9070 CALL -936
9080 POKE 60,STRL: POKE 61,STRH:
      POKE 62,STRL+7: POKE 63,STRH:
      CALL -589: PRINT
9090 FOR I=0 TO 28: POKE 3077+I,
      PEEK (1152+I): NEXT I: POKE
     3106,30: REM INTO N$
9100 POKE 3081,186: REM ":"
9110 GOSUB 7030: PRINT N$: GOSUB
     9500
9120 STRL=STRL+8: IF STRL<256 THEN
     9130:STRH=STRH+1:STRL=STRL-
     256
9130 IF STRH<>STPH OR STRL<>STPL THEN
     9070
9140 GOSUB 7030: PRINT "3D0G": GOSUB
     9500
9150 CALL -936: PRINT "INSTRUCT RECEI
     VER:": PRINT
9160 PRINT "  (PICK UP PHONE IF 'PICK
     UP')": PRINT "  CTRL-A CTRL-Z"
     : PRINT "  CTRL-A CTRL-X RTN"
     : PRINT "  >SAVE FILEMOVER"
```

```
     : VTAB 15
 9170 GOSUB 7030: PRINT "": GOSUB
     7010: REM CTRL-T
 9180 GOSUB 7020: POKE 1912+MSLOT,
     8+2
 9190 BOOT=0: GOTO 400
 9500 GOSUB 7010: GOSUB 7020: INPUT
     M$: GOSUB 7000: RETURN
20000 REM ***************************
20010 REM *                         *
20020 REM *         FILEMOVER       *
20030 REM *                         *
20040 REM *      DANA J SCHWARTZ    *
20050 REM *       (301)725-6281     *
20060 REM *                         *
20070 REM *          5/15/80        *
20080 REM *                         *
20090 REM ***************************
20100 REM      0- 999   SET-UP
20110 REM   1000-1999   SEND
20120 REM   2000-2999   RECEIVE
20130 REM
20140 REM   3000-3999   ACK RCVR
20150 REM   5000-5999   CALL RWTS
20160 REM   6000-6999   FIND T/S LIST
20170 REM   7000-7999   IN/PR SUBS
20180 REM   8000-8999   ERROR HANDLER
20190 REM
20200 REM   9000-9999   SELF-TRANSFER
20210 REM    MODEM SLOT # LINE 170
```

>

```
        ORG $800
BUF     EQU $0900
RWTS    EQU $03D9
*
START   LDA #>IOB    HI ORDER
        LDY #<IOB    LO ORDER
        JSR RWTS
        BCS ERROR
        RTS
*
ERROR   LDA ERCD     ERR HANDLER
        STA SVCD
        RTS
*
SVCD    DS  1
*
IOB     HEX 0160     I/O CTL BLOCK
        HEX 0100
        HEX 1100
        DA  DCT
        DA  BUF
        HEX 0000
        HEX 01       READ
ERCD    HEX 00
        HEX 00600100
*
DCT     HEX 0001     DEV CHAR TBL
        HEX EFD8
        END
```

```
        ORG  $0B80
BUF     EQU  $0A00
STATUS  EQU  $C0B6
DATA    EQU  $C0B7
CKSUM   EQU  $0829
*
INPUT2  LDX  #$0
        STX  CKSUM
LOOP    LDA  #$1
TEST    BIT  STATUS
        BEQ  TEST
        LDA  DATA
        STA  BUF,X
        EOR  CKSUM
        STA  CKSUM
        INX
        BNE  LOOP
*
        RTS
*
        END  INPUT2
```

```
        ORG  $0B00
BUF     EQU  $0A00
STATUS  EQU  $C0B6
DATA    EQU  $C0B7
CKSUM   EQU  $0829
*
OUTPUT2 LDX  #$0
        STX  CKSUM
LOOP    LDA  #$2
TEST    BIT  STATUS
        BEQ  TEST
        LDA  BUF,X
        STA  DATA
        EOR  CKSUM
        STA  CKSUM
        INX
        BNE  LOOP
*
        RTS
*
        END  OUTPUT2
```

WRITING A WORD PROCESSOR

John L. Moon

INTRODUCTION

This article is about writing a word processor program. This will serve as a good example of a number of programming techniques from design through implementation and testing. However, It is neither the best word processor nor the best example of good programming techniques; rather it is a personal example of doing a useful job using the APPLE II.

This article will identify program development stages which correspond to: problem definition, design, implementation stages and the remaining "wish list".

PROBLEM DEFINITION

My first "problem" was to define the problem. I knew that I would like to use my APPLE and its new Paper Tiger to replace my old Smith-Corona. I was somewhat familiar with word processors having used such systems at work as well as having bought APTYPE and played with Easywriter a little. I suppose, if all I was really interested in was word processing, that I could have found a system that would have satisfied my requirements. However, I had this nifty full-screen text editor written by a friend of mine in Connecticut that was better than any other text editor that I have seen for the APPLE (barring a few minor aspects easily forgiven since it was free software, written in Applesoft so I could update it, and used text files that were compatible with other programs I have written including another text editor, a graphics package and a disk oriented assembler.)

I do not like to learn a different text editor for each application that I have that requires generation of a text file. So to me, it was a hard requirement that the text processor be standalone in that it would take inputs from a standard text file and either print on the printer or generate a listing file that could then be printed to the printer by a list utility.

Within this environment, my requirements began to evolve. The word processor should:

1) Operate in a standalone manner, independent of the text editor used.

2) Use standard text files.

3) Be written in Applesoft for ease of change and transportability to other APPLE computers.

For its word processing capabilities, I had my own favorite list of commands I always used and Genivie Urban's list of what she used for generating the club newsletter. I made up numerous lists, but finally settled on the following for an inital implementation:

Line break: cause a line to be outputted whether or not the line is full.

Center lines.

Comments to allow documenting change levels and stubbing of commands.

Indentation of lines left and right.

Definition of page length.

Page ejection.

Definition of line length.

Control over formatting, concatenation and justification.

Spacing of empty lines.

Begin a paragraph.

This is not necessarily a complete list of all the capabilities that I wanted (see "wish list" below) but represented an adequate set of initial capabilities. To represent capitalization, I used the up arrow (shift-n) to designate which letters should be capitalized. A whole line can be capitalized by a double up arrow combination. This method of capitaliztion is a necessary compromise - by using a general text editor, I couldn't be guaranteed of being able to use the escape key and the inverse video that many other APPLE word processors use. However, this is a fairly straightforward way to implement capitalization. If the text editor can create lower case then this subterfuge doesn't have to be used.

DESIGN

Having decided (over a period of weeks to months for some items) what it was that I wished to implement, it remained to figure out how to implement it. This consisted of laying out overall block diagrams of data and control flow including trial codings of the key problems to be faced. The major areas that had to be carefully thoughtout beforehand were:

The initialization and termination processing.

File formats, opening, closing, reading, and writing.

Input line aquisition.

Command decode and processing.

Line concatenation and justification.

Of course, to build a fully functional word processor, each of these areas would eventually have to be represented as actual Basic programming statements. But in the design stage, I wrote just enough down to explore how I would do each of these areas. When I felt like the rest was just detail that could easily be filled in, then I went on to a different part of the problem. Considering that this word processor would be something less than 300 - 400 statements, I didn't formally write up any of the design documentation, but rather depended on my memory to recall or reconstruct the techniques as needed.

The bulk of the implementation was then written in one night. I started with one of my simple text editors that already had the general structure i was looking for:

Initialization.

Command decode.

Command and processing routines.

Termination.

I then modified it piecemeal, doing some testing along the way. Having identified major subsections that needed to be written, I picked them in an order that allowed me to incrementally add capability to the program and then test each piece.

First, the file initialization was written along with part of the command decode. Since I would be testing it manually, the I/O was originally configured to only be from the keyboard and to the display.

Then the commands were stubbed by writting a IF C$ = ".xx" THEN RETURN statement was written for each command. The output routine was stubbed by just having it print out the current string. The input routine was then written to get individual characters and concatenate them together to prevent commas and colons from terminating the input incorrectly. Traps were put in the routine to allow a quick exit anytime I wanted. (With this kind of routine, a ctl-C for break won't work!)

At this point the program could be run. This checked for syntax errors as well as logic errors.

I wrote the line justification next. It works on the principle of deciding how many blanks should be added to the line and then trys to scatter the blanks into the line equally from both sides. It was debugged by typing in lines and watching it work. By using the quick escape in the input mode, intermediate variables could be inspected with the Basic immediate mode.

From this point onwards, I wrote commands and tested them one by one. Finally, I rewrote the input and output sections so as to be able to access files and printer. The former ability to input test data via the keyboard was retained.

Capitalization was the last capability put in, added as an extra step in the input line aquisition.

DOCUMENTATION

Documentation is almost always left to last. In this case, my excuse was that I could use the finished word processor in order to write the documentation. So my test file became my user guide. Incidentally, any mistakes the processor made in processing the user guide served to document the software errors that needed fixing.

THE "WISH LIST"

Like every software product, this one doesn't do all I'd like it to

do. My current list for future enhancements is:

Page numbering.

Table of contents generation from heading indicators.

Automatic index generation from identified phrases.

Symbols that can have a value supplied later.

A stacked file imbed capability to allow inclusion of files within files by reference.

Page headings and trailers.

Generalization of printer I/O, multiple fonts on the Paper Tiger.

SUMMARY

Included as attachments are the program listing and the user guide. You are invited to change it as required for your usage. There are a number of other types of programming techniques used that I didn't describe above. When I get the chance, I'll put a copy in the club library along with the text editor that I use with it.

(listing begins on p. 19.)

P A S C A L S I G
P A S C A L S I G
P A S C A L S I G

Share your Knowledge of Apple/UCSD Pascal and the Apple Language System. Help others and yourself! Bootstrap your way into the wonderful world of Pascal.

J O I N T H E P A S C A L S I G
J O I N T H E P A S C A L S I G
J O I N T H E P A S C A L S I G

No prior Knowledge of Pascal required.
All members cordially invited

First meeting after the regular August meeting.
SIG Leader: Tom WoteKi (aKa Dr. Wo)
547-0984

J O I N T H E P A S C A L S I G
J O I N T H E P A S C A L S I G
J O I N T H E P A S C A L S I G

```
]LIST

10 D$ =  CHR$ (4)
20  GOSUB 2000
30  GOSUB 3000
40  GOSUB 6000
100  GOSUB 2100
101  IF  LEN (A$) < 3 THEN 100
102 C$ =  LEFT$ (A$,3)
105  GOSUB 110: GOTO 100
110  IF C$ = ".BR" THEN  GOSUB 10
     00: GOSUB 1990:B$ = "": RETURN

120  IF C$ = ".CE" THEN  GOSUB 10
     00: GOSUB 1990: GOSUB 4000:C
     E = AN: RETURN
125  IF C$ = ".CO" THEN  GOSUB 10
     00: GOSUB 1990: GOSUB 4000:C
     O = AN: RETURN
130  IF C$ = ".CM" THEN  RETURN
135  IF C$ = ".FO" THEN  GOSUB 10
     00: GOSUB 1990: GOSUB 4000:F
     O = AN: RETURN
138  IF C$ = ".JU" THEN  GOSUB 10
     00: GOSUB 1990: GOSUB 4000:J
     U = AN: RETURN
140  IF C$ = ".IL" THEN  GOSUB 10
     00: GOSUB 1990:AN = IL: GOSUB
     4100:IL = AN: RETURN
150  IF C$ = ".IR" THEN  GOSUB 10
     00: GOSUB 1990:AN = IR: GOSUB
     4100:IR = AN: RETURN
160  IF C$ = ".LL" THEN  GOSUB 10
     00: GOSUB 1990:AN = LL: GOSUB
     4100:LL = AN: RETURN
170  IF C$ = ".PA" THEN  GOSUB 10
     00: GOSUB 1990:B$ =  CHR$ (1
     2): GOSUB 3100:B$ = "": RETURN

180  IF C$ = ".PL" THEN  GOSUB 10
     00: GOSUB 1990:AN = PL: GOSUB
     4100:PL = AN: RETURN
190  IF C$ = ".PP" THEN  GOSUB 10
     00: GOSUB 1990:B$ = " ": GOSUB
     3100:B$ = "     ": RETURN
195  IF C$ = ".SP" THEN  GOSUB 10
     00: GOSUB 1990:AN = 0: GOSUB
     4100:B$ = " ": FOR I = 1 TO
     AN: GOSUB 3100: NEXT I:B$ =
     "": RETURN
200  IF  LEFT$ (C$,1) = "." THEN
      RETURN
205  IF  LEN (A$) > 0 THEN  GOSUB
     7000
207  IF  LEN (A$) > 0 AND FO = 0 THEN
     B$ = A$: GOSUB 1200: RETURN

209  IF  LEN (A$) > 0 AND  LEN (B
     $) = 0 THEN B$ = A$: GOTO 21
     5
210  IF  LEN (A$) > 0 THEN B$ = B
     $ + " " + A$
215  IF CE > 0 OR CO = 0 THEN  GOSUB
     1000:B$ = "": RETURN
220  IF  LEN (B$) > (LL - IL - IR
     ) THEN  GOSUB 1000:B$ = E$: GOTO
     220
230  RETURN
500  GOSUB 2060: GOSUB 3050
550  END
1000  REM  PUT OUT THE CURRENT LI
     NE
1010  REM  USE ROUTINE AT 3000 TO
      OUTPUT TO FILE
1011  REM   PUT LEFTOVER INTO E$
1012 E$ = ""
1020  IF  LEN (B$) <  > 0 THEN CR
      = 0
1030  IF  LEN (B$) = 0 THEN  RETURN

1100 L1 = LL - IL - IR
1110 L2 = L1
1115  IF  LEN (B$) < L2 THEN L2 =
      LEN (B$) - 1:E$ = "": IF CE
      = 0 THEN  GOTO 1200
1119  IF CE > 0 THEN L3 = L1 -  LEN
     (B$): GOTO 1400
1120  IF  MID$ (B$,L2 + 1,1) <  >
     " " AND L2 > 5 THEN L2 = L2 -
     1: GOTO 1120
1130 E$ =  MID$ (B$,L2 + 2)
1140 B$ =  LEFT$ (B$,L2):L3 = L1 -
     L2: REM  L3=#OF BLANKS TO AD
     D
1142  IF JU = 0 THEN 1200
1145 L4 = H:L5 = L2
1150  FOR I = 1 TO L3
1160 L4 = L4 + 1
1170  IF  MID$ (B$,L4,1) = " " THEN
     L5 = L5 + 1:L2 = L2 + 1:B$ =
      LEFT$ (B$,L4) + " " +  MID$
     (B$,L4 + 1):L4 = L4 + 1: GOTO
     1190
1179  IF L5 - 1 = H - 1 THEN L5 =
     L2
1180 L5 = L5 - 1: IF  MID$ (B$,L5
     ,1) = " " THEN L2 = L2 + 1:B
     $ =  LEFT$ (B$,L5) + " " +  MID$
     (B$,L5 + 1): GOTO 1190
1184  IF L5 = H THEN L5 = L2
1185  IF L4 = L2 THEN L4 = H
1186  GOTO 1160
1190  NEXT I
1200  IF IL > 0 THEN  FOR I = 1 TO
     IL:B$ = " " + B$: NEXT I:B$ =
      LEFT$ (B$,LL)
1300  GOSUB 3100:B$ = "": RETURN
```

```
1400 J =  INT ((L3 - 1) / 2): FOR
     I = 1 TO J:B$ = " " + B$: NEXT
     I:CE = CE - 1: GOTO 1300
1990  IF CR = 0 THEN B$ = E$: GOSUB
     1000:CR = 1: RETURN
1991  RETURN
2000  INPUT "INPUT FILENAME?";F$
2005  IF F$ = "KEYBOARD" THEN D$ =
     CHR$ (13)
2010  PRINT D$;"OPEN ";F$
2030  RETURN
2060  PRINT D$;"CLOSE ";F$
2070  RETURN
2100  PRINT D$;"READ ";F$: GOSUB
     10000: PRINT D$
2110  IF  LEN (A$) = 0 THEN  GOTO
     500
2120  RETURN
3000  REM
3005  INPUT "OUTPUT FILENAME?";F2
     $
3010  IF F2$ <  > "PRINTER" THEN
      PRINT D$;"OPEN ";F2$
3020  RETURN
3050  IF F2$ = "PRINTER" THEN  RETURN

3051 B$ = "": GOSUB 3100: PRINT D
     $;"CLOSE ";F2$: RETURN
3100  IF F2$ = "PRINTER" THEN  PRINT
     D$;"PR#1"
3101  IF F2$ <  > "PRINTER" THEN
      PRINT D$;"WRITE ";F2$
3110  PRINT B$
3120  IF F2$ = "PRINTER" THEN  PRINT
     D$;"PR#0"
3121  IF F2$ <  > "PRINTER" THEN
      PRINT D$
3122  RETURN
4000  REM  LOOK FOR 1,N,ON,OFF RE
     TURN AN=1,N,32000,0
4010  IF  MID$ (A$,5,2) = "ON" THEN
     AN = 32000: RETURN
4020  IF  MID$ (A$,5,3) = "OFF" THEN
     AN = 0: RETURN
4030 AN =  VAL ( MID$ (A$,5)): RETURN

4100  REM  LOOK FOR 0,H,+H,-H
4110  IF  LEN (A$) = 3 THEN AN =
     0: RETURN
4120  IF  MID$ (A$,4,1) = " " THEN
     AN =  VAL ( MID$ (A$,5)): RETURN

4130 AN = AN +  VAL ( MID$ (A$,4)
     ): RETURN
6000 CR = 1
6010 CE = 0
6015 CO = 32000
6018 JU = 32000
6019 FO = 32000
6020 IL = 0
6030 IR = 0
6040 LL = 60
6050 PL = 60
6060 H = 7
6100  RETURN
7000 UP = 0:G$ = ""
7010  FOR I = 1 TO  LEN (A$)
7014  IF  MID$ (A$,I,1) = "↑" AND
     UP = 1 THEN UP = 1000
7015  IF  MID$ (A$,I,1) = "↑" AND
     UP = 0 THEN UP = 1
7016  IF  MID$ (A$,I,1) = "↑" THEN
     7100
7019 J = 0: IF UP > 0 THEN UP = U
     P - 1: GOTO 7030
7020  IF  ASC ( MID$ (A$,I,1)) >
      =  ASC ("A") AND  ASC ( MID$
     (A$,I,1)) <  =  ASC ("Z") THEN
     J = 32
7030 G$ = G$ +  CHR$ ( ASC ( MID$
     (A$,I,1)) + J)
7100  NEXT I:A$ = G$
7110  RETURN
10000 A$ = ""
10010  GET C$
10011  IF C$ = "%" THEN  END
10015  IF DB THEN  PRINT C$;
10020  IF C$ =  CHR$ (13) THEN  RETURN

10030 A$ = A$ + C$: GOTO 10010

]
```

USER GUIDE
FOR THE
JOHN MOON TEXT PROCESSOR

John L. Moon

May 10,1980

This document will describe the commands that make up the John L. Moon TEXT PROCESSOR. This is just a quick description so that I will not forget what the commands are that I used during the time that I am not using the word processor.

First, a little preamble on how to prepare files for processing with the text processor. Files may be created using either my editor, or Brad's ED (I would suggest ED since it has much nicer editing features The only requirement is that the file be a text file with the last record in the text file a null line (one with just a carriage return).

The syntax of a line is merely commands must start with a period in column one and are two letters followed by optional parameters, and everything else is just text. To represent capitalization, a special character, the up arrow (shift-n) is used. A single up arrow will capitalize the following letter. Two up arrows will capitalize all the following characters in that source line. The total list of commands at this point are: .br (break), .ce (center), .cm (comment, .il (indent line left), .ir (indent line right), .ll (line lenth), .pa (page eject), .pl (page length), and .pp (paragraph).

For all commands that take parameters, the parameter must follow exactly one blank after the command - except for the signed numeric parameters. They must have the sign (+ or -) as the fourth character with the number immediately adjacent.

Break will cause all accumulated words up to this time to be printed and a new line started. It is possible that the line may not be right justified.

Center takes either a numeric paramter, or ON or OFF. The following lines will be centered on the page.

Comment can be used to put comments in the source file that will never be printed. Note that any unrecognized command will be ignored and nothing printed but comment will always be available for this purpose.

Indent line left takes a numeric or signed numeric parameter. If an unsigned number, the left margin is set to that number. If the number is signed, then it is added to the current margin.

Indent line right is similar to .il but it indents from the right. It should be noted that both of these commands essentially subtract from the set linelength.

Line length is set to a numeric value. If the value is signed, it is added to the current value.

Page eject causes the paper in the printer to advance.

Page length will set a page length value similar to how line length works, but at the moment, it does not actually affect anything.

LIBRARIAN'S CORNER--A Page From The Stack
by David Morganstein

New additions to our library include two disks from the Denver Apple Pi of interest to adventure game players. These disks, numbered 181 and 182 are unchanged from the original copies. The programs build on a system entitled AEMON. You need 181 to build characters in a text file called FRESH MEAT (chuckle). The programs are described in detail in the latest issue of Recreational Computing.

Regarding contributions to the library, please bring disks with your programs to the monthly meetings. We will set up a machine there and copy your offerings. To encourage contributions, the club will give back a library disk of your choice in return for your contribution, if your program is selected for inclusion in the library. Keep those programs coming.

Disk Speed and Other Woes

I know there are some strange incompatibility problems in reading disks from other systems. Disks initialized on a Language System, for example, appear not to boot on non-Language Systems until UPDATED. For those who are unfamiliar with this term, there is a program on your MASTER DISK entitled UPDATE x.x.x, where the x's denote the particular version (e.g. 3.2.1). If you BRUN this program, it will place a new image of the DOS (disk operating system) on any disk. It is the DOS which is read during boot-up.

At the last meeting someone turned in six disks with the message, "These don't work...." When I got home, I found that five would catalog just fine and only one gave me an I/O ERROR. Please check all your disks and keep in mind the drive speed and update tips before you return library disks.

We have talked about drive speed problems before. Here's what I've recently learned. There is an old and a new version of the disk analog card (found inside your drive). The newer versions are more tolerant of speed variations. I recently had an "I/O ERROR" with my oldest drive which, when checked, had a speed error of 10 to 12 below optimal when checked with the library disk speed program. (Apple Pi Vol. 1.) After correcting this, the reading problems cleared up. Meantime, I checked another drive with the new analog card. It measured low by about 15, yet had no read problems.

Good Buys In Software

A revised version of the CALL A.P.P.L.E Program Line Editor is available. If you write or enter programs, this is an invaluable tool. It allows you to EDIT individual lines with ease. It is also a key-board filter in that you may define quick two-stroke entries to do complicated and useful things. For example, the sequence "ESC","1" causes the message "CATALOG, D1" to be sent to the DOS, affecting a cataloging of the disk in drive 1. Other built-in functions like "ESC","W" give you the starting address and length of the most recently BLOAD'ed

binary program. However, the power of the program is that you may build your own key-board macros to send anything you commonly use to the Apple nerve center.

"Rescue at Rigel" is a delightful sci-fi adventure. Similar to "Temples", "Datestones", and "Morloc", it is a journey through a maze of rooms on various levels. The object is to locate the captives (including the fair princess) being held by sinister monsters (the TOLLAHS) -- pictured as grasshopper-like characters. There are sentry-robots, plasmoids and thornets to avoid. Yes indeed, the leaders of the TOLLAHS are the HIGH TOLLAHS. (Groan.... ed.)

On that note, I will reset....

SIGAMES--A Small Group of People Having Fun
by Alban Gass.

SIGAMES is a small group of people who meet once or twice a month and discuss their mutual interests in games. The diversity of interest among the members on this subject is amazing. Yet, there is enough cohesion among the group to make most comments relevant to all. Some members are interested in artificial intelligence and LISP processing. Others prefer to develop game software while still others are looking to find an opponent to play a particular program.

Tom Lucas has been directing a group of several SIGAMES members in developing a table-driven Super Wompus or Adventure-like game. Technically, the structure is similar to that used in Scott Adams' Adventures and EAMON. Also being developed are the utilities to make it easy for someone to generate and enter new data into the program. The versatility of this technique is that everyone can develop their own scenario for an Adventure game with minimal effort. As an added feature, this structure will support real time combat.

Currently, portions of this program are being developed by different members and integrated into a unified program. Several areas of the project must still be designed in detail and no code has been developed for them. It may require several iterations of portions of the code before a documented program will be available. SIGAMES plans to make the program available through the APPLE PI Library to the entire membership.

The SIGAMES group meets immediately after the WASHINGTON APPLE PI meeting at a place specified during the general session. The SIGAMES agenda is free wheeling and the topic of the day may be only partially discussed on the appointed day. However, members do have fun and conversation is brisk. Product reviews. computer science topics, and classical game issues are discussed, subject to the availability of speakers. If games is one of your interests, come along and join the fun. If some other topic is your passion, form your own special interest group and share your interest with others.

YOUR AD HERE !

RATES		
	$30	full
	$15	half
	$10	quarter
	$ 6	eighth

(line copy only - no half-tones or colors)

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. They will come in a protective foam diskette mailer.

Also available for purchase by members at a discount price is the new APPLE II REFERENCE MANUAL (replaces the Red Reference Manual). The price of this manual is $17.00. You may pick it up at a meeting or have it mailed to you at no extra charge.

Amount

1. New APPLE II REFERENCE MANUAL - $17.00 each ------

2. PROGRAM DISKETTES
   Members: $5.00 per disk picked up at meeting
   $7.00 mailed to you...
   Non-members: $8.00 per disk picked up a meeting
   $10.00 mailed to you...

Volume 1--Utilities I	()
Volume 2--Utilities II	()
Volume 3--Games I	()
Volume 4--Games II	()
Volume 5--Games III	()
Volume 6--Games IV	()
Volume 7--Games V	()
Volume 8--Utilities III	()
Volume 9--Educational I	()
Volume 10-Math/Science	()
Volume 11-Graphics I	()
Volume 12-Games VI	()
Volume 13-Games VII	()
Volume 14-IAC Utilities IV	()
Volume 15-Games VIII	()
Volume 16-Utilities V	()
Volume 17-Graphics II	()
Volume 18-Education II	()
Volume 19-Communications	()
Volume 20-Music	()
Volume 21-Apple Orchard	()

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

Washington Apple Pi
P.O. Box 34511
Washington, D.C. 20034

First Class