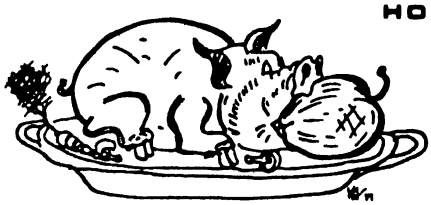
HOUSTON AREA APPLE USERS GROUP



News from the Apple Barrel

VOLUME 3 NO. 4 MAY, 1980

President, Bruce Barber Vice President, Bob Stout

Editor, Ed Seeger

<<< EDITOR'S DILEMMA >>>

'Most any Apple club anywhere aims to put out a newsletter with "something for everyone": beginner, advanced hobbiest, professional computer jock. Let me be honest with all...this issue misses the mark widely! With around 250 members now in H.A.A.U.G., we truly have representatives of "everybody," including junior high students, Pascal FORTRAN COBOL Assembler BASIC polyglots, wealthy dilettantes, EE's, a number of physicians, businesspeople of all types, and even a Presbyterian minister. And lots of beginners among us. Lots of them! Lots of US, for that matter, since so many of us will never get over the feeling that we're still just beginning to appreciate all the Apple is and all it can do.

This issue will therefore frustrate a lot of us. DO NOT THROW IT AWAY! Save it. Bring it out again in six months. Re-read it in ANOTHER six months. Unless you are strictly one of the dilettantes, curious but not actually interested, you will not always feel like a rank beginner. If you were a person who does not learn and grow, you'd have bought the Texas Instruments machine and parked your intelligence at the keyboard. Although this issue of APPLE BARREL is over a lot of heads, use it as a challenge rather than as a put-down. Bob Stout will teach us a bit of assembler language on the last Saturday of each month. David Black and Pat McGee have offered to teach some Pascal. There is a business sub-group within H.A.A.U.G. quite interested in file-keeping and sorting. Join them if you'd like to learn. And it's not so improbable at all that a year from now, some who are beginners today will not be able to live without CP/M or Z-80 80000 800000-based software, and will be glad to know your Apples can handle it! Hang in there, beginners. Even the minister says he understands a little of this!'

-EBS

```
0367: 9D 7E 03    118          STA LN.X    AND LOOP TO BUMP THE DIGIT
036A: CA          119          DEX         TOO ITS LEFT.
036B: 10 EE       120          BPL LOOP2   (UNCONDITIONAL BRANCH)
036D:             121 *
036D: 90 05       122 CKSP     BCC OUT     BRANCH IF WE DID NOT BUMP A SPACE.
036F: A9 01       123          LDA #$01    OTHERWISE, REPLACE THE BUMPED
0371: 9D 7E 03    124          STA LN.X    SPACE WITH A ONE.
0374:             125 *
0374:             126 *                    THE NEXT LINE NUMBER IS NOW
0374: 68          127 OUT      PLA         IN ARRAY LN.  RESTORE X.
0375: AA          128          TAX
0376: 68          129          PLA         RESTORE A (THE CR)
0377: 60          130          RTS         GIVE THE CR TO THE MONITOR.
0378:             131 *
0378:             132 * THE ILN ARRAY IS SET TO: SP.SP.1.0.0.SP.
0378:             133 *
0378: F0 F0 01
037B: 00 00 F0    134 ILN      DFB $F0.$F0.1.0.0.$F0
037E:             135 *
037E:             136 * JUST IN CASE THE USER STARTS WITH A CALL TO 787
037E:             137 * RATHER THAN TO 770. INITIALIZE LN TO THE SAME THING.
037E:             138 *
037E: F0          139 LN       DFB $F0     (894 DEC) 10000 DIGIT.
037F: F0          140          DFB $F0     (895 DEC)  1000 DIGIT.
0380: 01          141          DFB 1       (896 DEC)   100 DIGIT.
0381: 00          142          DFB 0       (897 DEC)    10 DIGIT.
0382: 00          143          DFB 0       (898 DEC) UNITS DIGIT.
0383: F0          144          DFB $F0     ASCII SPACE WHEN $80 IS ADDED.
0384:             145 *
0384:             146 ************-WARNING-***************
0384:             147 *                                   *
0384:             148 * DO NOT MAKE PROGRAM TOO LONG.     *
0384:             149 * DOS 3.2 POINTERS START IN $03D0.  *
0384:             150 *                                   *
0384:             151 *************************************
0384:             152 *                                   *
0384:             153 * BSAVE AUTO-NUMBER.A$302.L$82      *
0384:             154 *                                   *
0384:             155 *************************************

    0   ERRORS IN THIS ASSEMBLY
```

DOS Dissasembly

by Lee Meador

We continue the listing of the assembly language code for the DOS 3.2. This is supposed to the original listing of RWTS. It has passed through many generations of copy. This is the version found in the WOZPAK, a collection of information put out by the Apple Puget Sound club. The last six pages will be in our next issue. Also, I expect to have the commented disassembly of the lower level routines that RWTS calls. (RDADR, SEEKABS, POSTNIBL, PRENIBL, READ and WRITE.) You can find more about RWTS in your green, spiral bound DOS 3.2 manual from Apple Computer, Inc.

```
* 302, 383

302- 48 8A 48 A2 05 BD
308- 78 03 9D 7E 03 CA 10 F7
310- 68 AA 68 48 A9 21 85 38
318- A9 03 85 39 20 EA 03 68
320- 60 48 8A C9 06 10 0A 68
328- 91 28 BD 7E 03 18 69 B0
330- 60 68 2C 00 C0 10 FB 91
338- 28 AD 00 C0 2C 10 C0 48
340- C9 98 D0 0D A9 1B 85 38
348- A9 FD 85 39 20 EA 03 68
350- 60 C9 8D F0 02 68 60 8A
358- 48 A2 03 FE 7E 03 BD 7E
360- 03 C9 0A 30 08 A9 00 9D
368- 7E 03 CA 10 EE 90 05 A9
370- 01 9D 7E 03 68 AA 68 60
378- F0 F0 01 00 00 F0 F0 F0
380- 01 00 00 F0
```

USING RWTS ROUTINE

Normally, user access to and from DISC-II is achieved through the use of APPLEDOS. However, another method of accessing the disk is directly available from assembly language. From assembly language the user may read or write a block of 256 bytes to anywhere on the disk, thereby giving the user greater flexibility in using the disk.

Every diskette used in the Disc-II drive is seperated into 35 tracks, numbered Ø to 34. These tracks may be pictured as the grooves on a phonograph record, except they are not connected with each other. Basically, the tracks are concentric circles, with the large hole in the center forming the common center of the circles. The disk drive has a 'head' (which may be pictured as the needle on a record player, except the head is magnetic and never touches the disk itself) and this head 'seeks' to different tracks. Track Ø is on the outside, and is the largest circle, whereas track 34 is the inside track, nearest the center.

Each track contains 13 sectors. Sectors are logical groupings on each track, allowing the user to work with blocks of 256 bytes at a time, rather than the full 3328 bytes that fit on a track. The sectors are numbered Ø to 12, arranged around the disk. As the disk spins, each sector will pass underneath the head, at which time it may be read from or written to. Each sector actually consists of two portions; the address field, and the data field. The address field contains information telling what track the head is on, what sector number is about to spin past the head, and the volume number of the diskette. The data field contains the actual 256 bytes that is associated with each track and sector.

The read or write a track and sector routine (referred to as the RWTS routine) allows, from assembly language, any track and sector to be read from or written to. To use this subroutine, the user must create an IOB (Input/Output control Block) table. The IOB directs the routine which track and sector to use this access, which slot and drive to use, and what volume number is expected on this diskette.

To use the RWTS routine, the user must set up the IOB, then JSR to the subroutine. When RWTS is entered, the A & Y registers must point at the IOB. (A register contains high address byte, Y register contains the low address byte) The IOB format is as follows: (17 bytes used)

BYTE#	NAME	PURPOSE
1	IBTYPE	Tells the RWTS routine what type of IOB this is. Should be a Ø1. No other type codes available currently.
2	IBSLOT	Gives the slot number of the controller card for the disk to be used this access.

BYTE#	NAME	PURPOSE
3	IBDRVN	Gives the drive number to be used this access. Must be either a Ø1 or a Ø2.
4	IBVOL	Expected volume number to be found. Note that Volume ØØ matches any volume actually found on the diskette.
5	IBTRK	Track number to use this access. Must be in the range Ø to 34.
6	IBSECT	Sector number to use. Must be in the range Ø to 12.
7&8	IBDCTP	Pointer to the device characteristics table. Must contain low, then high byte of address of device characteristics table.(see below)
9&10	IBBUFP	Pointer to where the data to be used should be located. Must contain the low, then high byte of address. (Said to be pointing at the data buffer)
11	------	Unused.
12	------	Unused.
13	IBCMD	Command code. Tells the RWTS routine what to do. Commands are: ØØ -- Null command. Does nothing but start the disk and position the head. Ø1 -- Read a track and sector. Ø2 -- Write a track and sector. Ø4 -- Format the disk. Note that once a disk has been formatted, there is no data on the diskette, and therefore a sector cannot be read until it has been written first.
14	IBSTAT	Error code. If the RWTS routine returns with the carry flag clear, no error has occured. If it returns with the carry set, this byte will indicate what type of error has occurred. $1Ø -- Diskette is write protected, and cannot be written to. $2Ø -- Volume mismatch error, ie, a different volume number was found than was expected. $4Ø -- Drive error. Something very strange is happening. (Undefined) $8Ø -- Read error. RWTS routine was unable, after reapeated attempts, to read either the address field or the data field. If the data field for this sector has never been written, then a read error will result, since there is no data to read.
15	IBSMOD	The number of the volume actually found is returned in this location. Not valid if a read error has occurred.
16	IOBPSN	Slot number used for access before this one. MAKE SURE THIS LOCATION IS SET UP!
17	IOBPDN	Drive number used for previous access.

Format of Device Characteristics Table: (4 bytes total)

Byte#	Name	Purpose
1	DEVTPC	Device type code, telling what type of device this is. Should be a zero for DISC-II

Byte#	Name	Purpose
2	PPTC	Number of phases per track. Should be a one for DISC-II.
3&4	MONTC	Motor on time count complemented, in 100 microsecond intervals. Should be $D8EF for DISC-II, stored as low, then high bytes.

The entry point for the RWTS routine = $3D9

JSR $3E3 + A+Y will point to IOB.

Internal Workings of RWTS:

The RWTS routine does more than just read or write data; it also formats diskettes, and more importantly, handles and attempts to recover from errors encountered. Following is a general outline of the workings of RWTS:

1. Determine the new slot number for the controller.
2. Is the new slot number=old slot number? Yes, step 4.
3. Wait for motor to turn off.
4. Is the motor currently on? Save result of test.
5. Turn the motor on regardless of previous state.
6. Is the previous drive=current drive? Yes, step 8.
7. Set motor on test result from step 4 to false. (Since we changed drives, wait for new drive to come up to speed)
8. Seek head to desired track.
9. Was the motor on in step 4? Yes, step 11.
10. Wait for the motor to come up to speed.
11. Were we given the Null command? Yes, step 32.
12. Were we given the format diskette command? Yes, step 36.
13. Were we given the write sector command? Yes, prenibblize the data to be written.
14. Set RETRYCNT=48.
15. Read the next address field.
16. Was an error encountered in reading an address field? No, step 23.
17. Set RETRYCNT=RETRYCNT-1.
18. Is RETRYCNT=0? No, step 15.
19. Is SEEKCNT≠0? Yes, step 33. (We have recalibrated once before, no hope for user)

20. Recalibrate out to track ØØ, then to desired track.

21. Set SEEKCNT nonzero. (indicating we have recalibrated once already)

22. Go back to step 15.

23. (We read an address field correctly) Are we on the correct track? Yes, step 26.

24. Seek to correct track.

25. Go back to step 15.

26. Is this diskette the correct volume number? No, step 33.

27. Is this the correct sector? Yes, step 31.

28. Set RETRYCNT=RETRYCNT-1.

29. Is RETRYCNT=Ø? No, step 15.

3Ø. Go to step 33.

31. Do either read or write operation, depending on command given.

32. If no error occurred in actual data operation, indicate no error and go to step 34.

33. Indicate appropriate error.

34. Turn the motor off.

35. Exit.

36. (Format the disk.) Set TRKCNT=Ø.

37. Set Auto Sync count=8Ø.

38. Seek to Track ØØ.

39. Write this track several times over with self sync nibbles.

40. Format track:
    A. Set sector count=Ø.
    B. Write (Auto Sync count) nibbles of self sync.
    C. Write address marks.
    D. Write volume number, track number, sector number, and checksum bytes.
    E. Write closing address mark.
    F. Write 429 nibbles of self sync to replace the data field until real data is written in.
    G. Set sector count=(sector count+7) MOD 13 to interleave the sectors with a factor of 4.
    H. If sector number ≠ Ø, go back to step A.

41. Read the next address field.

42. If sector Ø address field was not read, we overwrote sector Ø with self sync.

Therefore, set Auto Sync count=Auto Sync count-1, then go back to step 40.

43. If sector 0 was read, this track was formatted correctly. Move in a track and loop back to step 39, if head was not on track 34.

44. If head was on track 34, we have successfully formatted this diskette. Exit.

```
 1 ****************************
 2 *                          *
 3 *          DISK II         *
 4 * READ/WRITE TRACK-SECTOR  *
 5 *                          *
 6 *   COPYRIGHT 1978 BY      *
 7 *  APPLE COMPUTER, INC.    *
 8 *                          *
 9 *  ALL RIGHTS RESERVED     *
10 *                          *
11 *       R. WIGGINTON       *
12 *                          *
13 ****************************
```

```
74 *     READ/WRITE A       *
75 *   TRACK AND SECTOR     *
76 *                        *
77 **************************
78 *                        *
79 *   ENTER WITH A & Y     *
80 * REGISTERS POINTING TO  *
81 * THE I/O CONTROL BLOCK  *
82 * (THE 'IOB'). INSIDE    *
83 * THE IOB:               *
84 *                        *
85 * IBTYPE: IOB TYPE CODE  *
86 *        (SHOULD BE A 01)*
87 *                        *
88 * IBSLOT: CONTROLLER SLOT*
89 *        NUMBER FOR THIS *
90 *        ACCESS.         *
91 *                        *
92 * IBDRVN: DRIVE NUMBER   *
93 *        FOR THIS ACCESS *
94 *                        *
95 * IBVOL: EXPECTED VOLUME *
96 *       NUMBER. NOTE THAT*
97 *       VOLUME 00 MATCHES*
98 *       ANY VOLUME NUMBER*
99 *                        *
100 * IBTRK: TRACK TO USE    *
101 *       THIS ACCESS      *
102 *                        *
103 * IBSECT: SECTOR NUMBER  *
104 *        TO USE THIS TIME*
105 *                        *
106 * IBDCTP: POINTER TO THE *
107 *        DEVICE CHARACTER*
108 *        ISTICS TABLE.   *
109 *                        *
110 * IBBUFP: POINTER TO THE *
111 *        PLACE THE DATA IS*
112 *        OR SHOULD BE.   *
113 *                        *
114 * IBDLEN: AMOUNT OF DATA *
115 *        IN BYTES TO BE  *
116 *        PROCESSED.      *
117 *                        *
118 * IBCMD: COMMAND CODE:   *
119 *       0-> NULL COMMAND *
120 *       1-> READ SECTOR  *
121 *       2-> WRITE SECTOR *
122 *       4-> FORMAT DISK  *
123 *                        *
124 * IBSTAT: ERROR CODE:    *
125 *       0-> NO ERROR     *
126 *     $10-> WRITE PROTECT*
127 *     $20-> VOLUME ERROR *
128 *     $40-> DRIVE ERROR  *
129 *     $80-> READ ERROR   *
130 *                        *
131 * IBSMOD: LOCATION TO    *
132 *        RETURN THE VOLUME*
133 *        NUMBER ACTUALLY *
134 *        FOUND.          *
```

```
135 *                          *
136 * IOBPSN: PREVIOUS SLOT    *
137 *       NUMBER USED LAST   *
138 *       ACCESS.            *
139 *                          *
140 * IOBPDN: PREVIOUS DRIVE   *
141 *       NUMBER USED LAST   *
142 *       ACCESS.            *
143 *                          *
144 ****************************
145 *                          *
146 * DEVICE CHARACTERISTICS   *
147 * TABLE DESCRIPTION:       *
148 *                          *
149 * DEVICE TYPE CODE         *
150 *  (ZERO FOR DISK II)      *
151 *                          *
152 * NUMBER OF PHASES PER     *
153 * TRACK (TWO FOR DISK II)  *
154 *                          *
155 * MOTOR ON TIME IN 100     *
156 * MICROSECOND INTERVALS    *
157 * COMPLEMENTED. ($D8EF     *
158 * FOR DISK II)             *
159 *                          *
160 ****************************
```

This is a poor place to stop! RWTS and HIRES are on the next page, but we need to hold off till June's issue. Lee's disassembly was just beginning to get interesting, but will still be here next month. Thanks to Lee Meador of the Fort Worth Apple Users Group for this series, which continues for several more installments, all of which will be re-printed here in APPLE BARREL.

<<< WANT AND DON'T WANT ADS >>>

FOR SALE: 8, 4116 memory chips bought through HAAUG. 16K for $55.00 from Phil Krueger, work 659-3511, home 665-5054. These 200ns chips are guaranteed by the club.

<<< A WHOLE NEW WORLD! >>>

Word has been going around for several weeks about Microsoft's Z-80 SoftCard for the Apple. Expected at the end of May, this peripheral will allow Apple owners to take advantage of the immense body of Z-80 software, including the CP/M system from Digital Research, and a BASIC compiler, soon to be ready.

Advantages will be:

PRINT USING for formatted output

LONG VARIABLE NAMES up to 40 characters

RANDOM DISK I/O with variable length records

WHILE/WEND statement

16-DIGIT PRECISION

ENHANCED GRAPHICS, with LINE, PUT and GET

SUPPORTS SOUND

AUTO and RENUM

EDIT MODE SUBCOMMANDS

CALL statement for assembly language or FORTRAN

CHAIN and COMMON to pass variables to another program

The cost will be under $400, and it is possible that a local dealer might be interested in offering an introductory discount.

Pascal Problems

by Pat McGee
P.O.Box 20223
Houston, Texas 77025

This is a list of problems I have had using the Apple Pascal system. Some are outright bugs, while others are problems caused by poor documentation.

Long Integers:

I expected them to work just like regular integers, except hold bigger numbers. They don't. In some places they do, in others they cause compilation errors, and sometimes they just plain don't work.

They do work as expected in most arithmetic expressions and a parameters to functions and procedures.

Trying to have a function return a value of type long integer causes a compilation error. The Apple Hot Line said that this was a limitation that had not been documented, not a bug. Long integers are similar in internal format to strings, and strings cannot be used in this manner.

There are several bugs involving long integers.

1. Typing a 10 digit number when the system is executing
   Read(input,I) where I:Integer[9]
   usually causes the system to crash. The only way to recover is to reboot.
2. Sometimes, keying in any number when the system is trying to read a long integer will cause it to *STK OFLOW* and reinitialize itself. I haven't found exactly what things work and what don't.
3. The expression TRUNC(Adr - 32768) where Adr:Integer causes *STK OFLOW*, but TRUNC(Adr - 16384 - 16384) does not.

Mod Function:

This does not work properly. Jensen & Wirth (p13) state that
A Mod B = A-((A div B)*B).
However, in Apple Pascal, it is implemented as
A mod B = |A|-((|A| div B)*B).
This can be seen by looking at -1 mod 2. This is particularly bad when looking at the definition of modulo munbers from back in high school. I was taught that if A mod B = C then (A+B) mod B was also = C. The implementation does not match this.

Arctangent Function ATAN:

This function returns the wrong angle for tangents less than -1. Use the following code when you want to use this:

```
If Tangent < 0 then
  Angle := -Atan(-Tangent)
Else
  Angle := Atan(Tangent);
```

For Loops:

I was trying to time a for loop, so I typed in:

```
Writeln(output,'BEFORE LOOP');
For i := 0 to 32767 do {nothing};
Writeln(output,'AFTER LOOP');
```

The computer printed "BEFORE LOOP", then I waited, with cocked stopwatch. After a while, I decided an alarm clock would be a more

appropriate instrument. Even later, I was considering a calendar. Well, back to the drawing board. Changing 32767 to 32766 produced a nice quick loop, but changing it back to 32767 caused another infinite wait.

Apparently, the compiler designers blew it. The value of I should have been checked against 32767 before being incremented, or the increment should have checked for overflow.

To avoid the problem, either use 32766 or do the following:

```
Const Max = 32767
Type LoopControlState = (looping,thru);
Var  State:LoopControlState;
     I:Integer
Begin
  I := 0; State := looping;
  Repeat
    { Whatever }
    If I < Max then I := I+1 else State := thru;
  Until State = thru;
```

I use this instead of any for loop, because it is more versitile, and because it works in all cases. There are other reasons involving the use of variables that do not go outside the specified range.

Filer W(hat Command:

This command tells you the name of the workfile and whether it has been saved or not. In a single drive system, it works file. But, if you G(et a file from a different disk drive than you booted from, do something to it, then S(ave it back to the other disk, the W(hat command thinks that the workfile has not been saved, when in fact it has been.

Filer T(ransfer Command:

If you have two disks in the system at the same time and they have the same name, DON'T USE THE T COMMAND!!!!!!! You will wipe out part of at least one disk!. The filer gets very confused under these circumstances, and is apt to wipe out the disk you are transferring from, as well as the one you are transferring to. Furthermore, you sometimes don't find out until later just which files are messed up. They will look just fine in the directory, but the contents will be so much garbage.

If you must to this, first change the name of one of the disks, do the transfer, then change the name back to the original. The manual says (once, in a very obscure place which I can't find again) not to put in two disks with the same name, but doesn't say why.

Another problem I had was in using the T command to transfer several files from one disk to another. When I keyed in

```
T  AMF:T.=.TEXT,AMFBACK:$
```

I got the messge DESTROY AMFBACK:? (Y/N)
I don't know what would have happened if I had said yes because I never had the guts to try it.

System Library:

Several times I have seen the message:

```
REQUIRED INTRINSIC(S) NOT AVAIUABUE
```

when trying to R(un or E(xecute a program. I soon found out that SYSTEM.UIBRARY had to be in the system. However, this was not the complete answer as I found out when I put a disk with it in #4 and tried again. As it turns out you MUST boot from a disk that has the library on it. If you boot from a disk without it, then put in a disk with it, the system can't find it.

This is documented in the manual, but only in a discussion of making a new library file. This is a place a beginner would not look at, and I skipped it my first few times through the manual. It should be in the section on E(xecute also.

Assembler:

When doing a forward branch (not a jump), the listing does not properly reflect the contents of the code file. When the branch is processed, the listing reads, for example:

```
D3EA|F0**       BEQ    $1
```

A few lines later, when the lobel is defined, the listing reads

```
D3EA*00
```

It should read

```
D3EB*05
```

Both the address and the contents are wrong.

Editor:

When in D(elete mode and deleting off the bottom of the screen, the editor rewrites the screen starting with the next line to be deleted at the top. It then blanks out the first 3 characters of that line and positions the cursor to the first blanked out character. These three characters have not been deleted, but the editor makes it look like they have been. Until I found out that everything was OK, I used to panic and ESC out of the delete and start over. This is not necessary, as they have not been deleted.

Conclusion:

This is not all the complaints I have with the Apple Pascal system, but a all the others involve the poor documentation or things that I would have designed differently. Most of the documentation problems I expect to be cleared up when Jef Raskin and his crew write a manual. The current manual was copied mostly verbatim from the UCSD Pascal manual, and almost all of its problems stem from that source.

If you have encountered a problem not in this list, please tell me (and Apple) about it. Hopefully we can work out a way to avoid it and keep others from wasting much effort finding the same bugs over again.

<<< PASCAL MINI-COURSE >>>

The full "Introduction to Programming in Pascal" course advertised in the last issue of APPLE BARREL had too few registrants to begin. However, the instructor, David Black, has offered a briefer, less comprehensive, less expensive course for any in H.A.A.U.G. who are interested. This abbreviated class can begin any time between june 6 - 20. Richard Bluefarb and Computer City will again coordinate it. Call Richard at 821-2702 to express your interest. David and Pamela Black will shortly be moving to Austin, so those who think they'll "get around to learning to use my Pascal system someday" might want to find time now!

SORT SUBROUTINES

Sorting is the process of arranging a bunch of data (a data base) according to a certain criterion. The most common sorting problems which will be covered in this article are 1)arranging a numerical array in numerical order and 2)arranging a string array in alphabetical order. In the following program listings A(I) is an N item array of numerical data and A$(I) is an N item array of alphanumeric data.

The simplest but usually the slowest routine is called Selection Sort. A listing for Selection Sort of numerical data follows:

```
1000  FOR I = 1 TO N - 1
1010  FOR J = I + 1 TO N
1020  IF A(I) <  = A(J) THEN 1040

1030 T = A(I):A(I) = A(J):A(J) =
     T
1040  NEXT J
1050  NEXT I
1060  RETURN
```

This routine steps through the array N-1 times. During the first pass through the it selects the smallest item and moves it up to the top of the array. On succesive passes through the array it moves the next smallest item next to the top and so forth until it has made N-1 passes and all items are in order. There are a fixed number of passes and an array that is already sorted will take the same number of passes as acompletely random array.

Several faster routines are available which minimize the number of passes through the array. Heapsort is about ten times faster than Selection Sort tor a numerical array. Two versions of Heapsort are presented below, one for numerical data and another for string data:

```
1000 L =  INT (N / 2) + 1:K = N
1010  IF L = 1 THEN 1030
1020 L = L - 1:S = A(L): GOTO 105
     0
1030 S = A(K):A(K) = A(1):K = K -
     1
1040  IF K < 1 THEN A(I) = S: RETURN

1050 J = L
1060 I = J:J = J + J
1070  IF J > K THEN A(I) = S: GOTO
     1010
1080  IF J < K THEN  IF A(J) < A(
     J + 1) THEN J = J + 1
1090  IF S >  = A(J) THEN A(I) =
     S: GOTO 1010
1095 A(I) = A(J): GOTO 1060
```

```
1000 L =  INT (N / 2) + 1:K = N
1010  IF L = 1 THEN 1030
1020 L = L - 1:S$ = A$(L): GOTO 1
     050
1030 S$ = A$(K):A$(K) = A$(1):K =
     K - 1
1040  IF K < 1 THEN A$(I) = S$: RETURN
1050 J = L
1060 I = J:J = J + J
1070  IF J > K THEN A$(I) = S$: GOTO
     1010
1080  IF J < K THEN  IF A$(J) < A
     $(J + 1) THEN J = J + 1
1090  IF S$ >  = A$(J) THEN A$(I)
      = S$: GOTO 1010
1095 A$(I) = A$(J): GOTO 1060
```

Another sort for numerical data is Quick Sort which is about one third faster than Heapsort. It could also be coded for string arrays. Also listed to the right is a program that generates a random N item numerical array for testing the above routines. After generating the array if clears the screen, prints "SORTING" and calls the sort subroutine at 1000. After completing the sort it prints each item consecutively.

QUICK SORT

```
1000 TP = 1:LO(1) = 1:UP(1) = N
1010  IF TP <  = 0 THEN  RETURN
1020 LB = LO(TP):UB = UP(TP):TP =
     TP - 1
1030  IF UB <  = LB THEN 1010
1040 I = LB:J = UB:TE = A(I)
1050  IF J < 1 THEN 1080
1060  IF TE >  = A(J) THEN 1080
1070 J = J - 1: GOTO 1050
1080  IF J <  = I THEN A(I) = TE:
      GOTO 1150
1090 A(I) = A(J):I = I + 1
1100  IF I > N THEN 1130
1110 : IF A(I) >  = TE THEN 1130
1120 I = I + 1: GOTO 1100
1130  IF J > I THEN A(J) = A(I):J
      = J - 1: GOTO 1060
1140 A(J) = TE:I = J
1150 TP = TP + 1
1160  IF I - LB < UB - I THEN LO(
     TP) = I + 1:UP(TP) = UB:UB =
     I - 1: GOTO 1130
1170 LO(TP) = LB:UP(TP) = I - 1:L
     B = I + 1
1180  GOTO 1030
```

TEST PROGRAM

```
10 N = 100
15  DIM A(N),LO(N),UP(N)
20  FOR I = 1 TO N
30 A(I) =  RND (N)
40  NEXT I
50  HOME : GOSUB 1000
60  FOR I = 1 TO N
70  PRINT A(I)
80  NEXT I
100  END
```

To give you some idea of the speed of these routines, Selection Sort takes almost 2 minutes for a one hundred item array. Heap sort takes less than twenty seconds and Quick Sort takes about fifteen seconds. Heapsort sorts numerical data about as fast as alphanumeric arrays. I have not tried the others on string data. If you want to sort faster than this you might try a machine language routine. Ampersort and perhaps others are available which are probably hundreds of times faster than Applesoft based sorts.

-- by David Marchand
Chairman
Hardware Projects

April 2, 1980

Apple is proud to announce an important new addition to its family of high level languages:

NAME: Apple FORTRAN

SHIP DATE: Early third quarter 1980

VITAL STATISTICS:

The availability of this powerful, well-established language provides an excellent opportunity to sell to engineering and technical professionals as well as the education market.

FORTRAN is universally known throughout the scientific community and is taught in many colleges and universities. Large libraries of FORTRAN programs are already in existence, particularly for math and science applications.

Apple FORTRAN is ANSI Standard Subset FORTRAN 77. This latest computer industry standard provides significant additions and enhancements over the previous 66 standard (FORTRAN IV). For example, structured programming concepts are added to the traditional FORTRAN statements through expanded IF statement constructs.

Apple FORTRAN operates in the Apple Pascal Language System. Thus, the same comprehensive software development environment which we provide to our Pascal users is now available to FORTRAN programmers. The Editor, Linker, Filer, and assembler can all be used with the Apple FORTRAN compiler, which, like Pascal, produces P-code. And FORTRAN can take full advantage of Apple's high resolution graphics capabilities, sound, and control paddles through Turtlegraphics and Apple Stuff units. Under the Apple Language System FORTRAN, Pascal, and assembler routines can be created and integrated into a single program.

Apple FORTRAN will be featured at NCC in Anaheim, May 19, and will make its European debut in April at the Hanover Fair.

APPLE FORTRAN PACKAGE (A2D0032):

* Two (2) FORTRAN system diskettes (16 sector)

* Apple FORTRAN Language Reference Manual

SYSTEM REQUIREMENTS:

* 48K Apple II or Apple II Plus

* Apple Language System

* Apple II Disk Drive with controller
  (Although one drive is adequate for small programs, two drives are strongly recommended for ease of use and more serious programming development.)

* Video monitor or television

PRICE:

* Suggested retail price will be approximately $200.

MORE TO FOLLOW:

Complete technical details, dealer price, and packaging information will be mailed to you in the next few weeks.

THE BOTTOM LINE:

FORTRAN provides an outstanding sales opportunity in two markets: industrial/scientific and education. Thousands of engineers and scientists use FORTRAN in their work every day on large expensive computer systems. Now they can develop FORTRAN programs on the Apple.

In the education market, Apple now offers the Big 3 teaching languages--Pascal, BASIC, and FORTRAN--on an inexpensive microcomputer system.

<<< STRINGY FLOPPY >>>

A firm in California, Exatron, has announced the marketing of a "stringy floppy" tape drive. It comes with a complete operating system and is, in effect, a slow disk! The system uses continuous loop micro cassettes, in lengths of 10, 20 and 50 feet. The 50-foot version will store about as much as one 5 1/4" floppy disk. A worst case load might take as much as 20 seconds, as contrasted with 5 to 7 seconds for a disk. Price is anticipated at around $250. This is clearly a desirable alternative to either a full Disk II with controller card, at $595 retail, or your basic Panasonic portable with all its well-known problems.

<<< DOS 3.P.1 SECURITY SYSTEM >>>

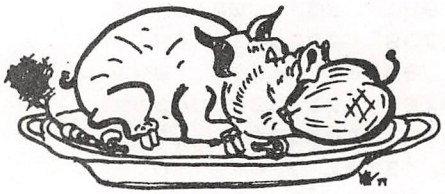
Apple Inc. is now licensing a secure DOS as factory support for OEM users. "Secure" translates roughly as "cannot copy/cannot read/randomized files/DOS altered beyond recognition/reset and control-C disabled/cannot be saved out to tape/does not support LOAD command, so cannot be examined/etc. This is the latest round in the Great Copywrite Wars, of course, and it attempts to deal constructively with software piracy, not to mention hobbyist software swipes, ah, swaps!

Whether or not everything from Bob Bishop's bouncing balls on up will now be rendered uncopiable, a la VisiCalc and Super.Text, remains to be seen, but it is no surprise that software authors have demanded and gotten second-strike armaments. You can be sure, however, that the very market forces that give rise to DOS 3.P.1 will also foster serious attempts to create a universal copy program for the Apple, and that the arcana of microprogramming and the P5 and P6 chips on the disk controller card will be assaulted without mercy. It is only a matter of time until DOS-secure disks bear the inevitable label, "Warning! I break for cloneheads!"

Houston Area Apple Users' Group
APPLE BARREL
Ed Seeger, Editor
4331 Nenana Drive
Houston, Texas 77035

(713) 723-6919

****
* BULK RATE *
* THIRD CLASS MAIL *
****



H.A.A.U.G

DeWayne Van Hoozer
4510 Avalon
Lawton, OK 73501

Postmasters:

Address correction requested