

HP-71 MISCELLANY

by Joseph K. Horn [13]

[Note added in 2015: This PDF version of the complete Titan File collection was created in August 2015. Many typos and other errors in the original Titan File articles have been corrected without comment. All newly inserted material, including links for downloads from the Internet, are in this "Core Sans M" font. The main text has been converted to Century Gothic, with HP-71 keywords converted to DejaVu Sans Mono, for maximum readability. Thanks to Bob Prospero for his great help and encouragement throughout this project. -jkh-]

The following miscellaneous items have only two things in common: they regard the HP-71, and they are more or less weird. So don't try to find the point of this article; there isn't any.

The Software IDS Volume I accidentally omitted the memory location of the CONTRAST nibble. Page 3-3 (bottom) should have this entry between TIMER1 and DD1CTL:

DCONTR (2E3FE) Display Contrast nibble.

There is a disastrous bug you should be aware of in system ROM version 1BBBB; you can mess up the file chain by CREATEing a file named ":MAIN". You should get an error, of course, but a null name followed by the :MAIN device specifier does *not* generate an error. It generates a file, apparently, but it does not show up in the catalog, and files created after it don't show up in the catalog either. The only way to repair the damage seems to be INIT :3. It's bad with a capital B. So NEVER create a file called ":MAIN".

There are four 7-nibble HP-IL memory locations, starting at 2F78D, designated "IS Assignments". The first three are documented: IS-DSP is the DISPLAY IS assignment; IS-PRT is the PRINTER IS assignment; and IS-INP is the KEYBOARD IS assignment (INP stands for INPUT, I guess). But the fourth, called IS-PLT, is undocumented. Could it be for a future command, PLOTTER IS?

LEX ID #F5 is allocated in the IDS for "Wand". Poll #40, called "pWAND", is "For additional barcode decoders". Buffer #81A, called "bWAND", is for "Wand Status/Cksum Info". No barcode reader has been advertised as in the works, but...

If you want your program to see whether or not a card reader is installed, you can PEEK\$("2C014", 1). If you get "0", then there is no card reader installed; otherwise, there is. The card reader is hard-configured in this area.

One-key Memory Lost! In FORTH, with an empty stack, the command BASICI results in a master clear. Sometimes the display goes bonkers before the dreaded Memory Lost message appears.

Command stack bug: if it's 16 high (extended to maximum possible height), then pressing down-arrow during INPUT, LINPUT, or FORTH is supposed to go to the most recent stack entry but instead it loops around to the least recent entry, and loops again if down-arrow is pressed repeatedly. If this is desired, use a stack 16 high. If not, leave it 15 high. The default stack height (5) behaves correctly.

"NEAR" rounding means in case of a tie (5) you round to the nearest even digit. Rounding 1.25 to the "near"est tenth would give 1.2 (2 is even), but rounding 1.35 to the "near"est tenth would be 1.4 (4 is even). The OPTION ROUND NEAR default setting only affects the 12th digit of results. To round "near" to other values, use the RED function. To round X to the "near"est whole number,

use $X\text{-RED}(X, 1)$. To round money to the “near”est cent, use $X\text{-RED}(X, .01)$. And so on. The MATH ROM function `IROUND` makes it even easier.

Bug!!! Merely turning off the HP-71 is not enough when you wish to change batteries or remove/insert modules! Beware! The bottom row of keys (except for `ENDLINE`) is active even when the HP-71 is turned off. Pressing any of them while the power is off puts the HP-71 into a running mode, with all that that implies. With a `DELAY` greater than 0, try pulling a module out (or pushing one in) while holding down a bottom-row key; then release the key and watch and listen to what happens! Even worse, changing the batteries with a bottom-row key down causes Memory Lost every time! Since pressing a bottom-row key is easy to do while grasping the machine to change modules or batteries, BE WARNED!!!

The clock’s adjustment factor (called “accuracy factor” in the IDS) is cleared by `RESET CLOCK`, and after that it is only set by the *second* execution of `EXACT`. (`AF` sets it too, of course). `SETTIME`, `SETDATE`, `ADJUST`, etc. do *not* affect the current adjustment factor. They “accumulate” the drift in another place in memory (2F763 through 2F77B), used by the second execution of `EXACT` to calculate the adjustment factor (stored at 2F787 thru 2F78C). So if your `AF` is to your liking, don’t worry about `SETTIME` etc. messing it up, like it did on the HP-41.

To insert `B$` in front of `A$`, use `A$[1,0]=B$`. In general, to insert a string without losing any characters starting at character `X`, use `A$[X,0]=B$`. This is not obvious in the Manual.

It has been objected that since $3*-4$ is allowed, 3^{-4} should also be allowed. This is not true. The algebraic order of operations says that *powers come first*, then negations, and finally multiplications. This allows -3^4 to give the correct answer, which is $-(3^4)$ or -81 . Thus 3^{-4} violates the order of operations and makes as much sense as 3^*4 . Since we would evaluate $3^{(-4)}$ in RPN as `3 ENTER 4 CHS Y^X`, with the negation preceding the power, we must use parentheses in the HP-71’s algebraic system to override its order of operations. Real HP-71 users will be the ones who learn its operations as thoroughly as most HP-41 users have learned the HP-41.

Bug: if the most recent command (bottom stack entry) does not end with an `ENDLINE` character (ASCII 13), as can happen when an unfinished calculation in `CALC` mode is up-arrowed, then the next command line received will merely replace that command, even if that means the creation of duplicate entries in the stack. Since duplicate stack entries are never supposed to occur, this is a bug.

To suspend a program without halting it, press `[g] [ON]` while it’s running; release to resume. This applies to machine code as well.

Three-way branch in BASIC. Rather than:

```
IF X<0 THEN 'A' ELSE IF X=0 THEN 'B' ELSE 'C'
```

use instead:

```
ON SGN(X)+2 GOTO 'A', 'B', 'C'
```

which uses fewer bytes and is faster. Speaking of which, if you type labels into a program, don’t use quotation marks. Even though `GOTO X` and `GOTO 'X'` both wind up looking like `GOTO 'X'` when listed, the former uses one less byte.

The MATH ROM functions can be used in ways that are not immediately obvious. Suppose you are a teacher. Fifty students take eight tests, each with a different weight (some tests are worth more than other tests, e.g. the final exam is worth 30% of the final grade, and the other tests are worth 10% of the final grade each). From the weights and the grades, how do you figure the final grade? Without the MATH ROM, you could do a `FOR-NEXT` loop to get the weights into an array, another

loop to get the grades into another array, and another loop to add up the products of the weights times the grades, and finally divide by the sum of the weights. With the MATH ROM you can input the Weights and Grades with a single MAT INPUT command (say, into arrays W and G respectively) and print the final grade with *one* expression: DOT(W,G)/CNORM(W).

Other MATH PAC function uses: You can use CNORM and RNORM on single-dimension arrays; RNORM(A) gives the value of the greatest element in A, and CNORM(A) gives the sum of the elements in A. BVAL can be used to make graphic displays easily.

GDISP CHARSET\$ shows the first 22 user-defined characters.

Frequency table for the BEEP command					
LOW	HIGH	LOW	HIGH	LOW	HIGH
C : 262	523	E : 330	659	G#: 415	831
C#: 277	554	F : 349	698	A : 440	880
D : 294	587	F#: 370	740	A#: 466	932
D#: 311	622	G : 415	831	B : 494	988

Outside these frequencies the '71 goes sour.

The '71 lets you calculate GOTO's not just with the ON statement. The statement GOTO "L"&STR\$(X) would create label "L153" if X=153, and would goto it. Other variations abound, like GOTO N\$, GOTO CHR\$(X), etc.

If you want your program to see if a particular module is installed, just use POS(VER\$, " MATH: ") or the like. If you want to check for a particular version, include it after the colon.

BEEP INF, EPS gives a loud tick if FLAG(-25)=1, and a quiet tick if FLAG(-25)=0.

The IDS speaks of "funny functions", "dummy arrays", and the "sticky bit". These non-standard terms give the beginner a laugh and then grief. "Funny functions" are functions that have no LEX ID# (like the FNROOT and INTEGRAL functions in the MATH ROM). They are referred to as such not because they are funny ha-ha but funny peculiar. (Volume II of the Software IDS often refers to an abnormal state as "funny"). "Dummy arrays" are array variable names followed by empty parentheses, such as A(). Dummy arrays are used in subprogram definition formal parameter lists, and may be used in subprogram call actual parameter lists, and when READING arrays from DATA or files, or PRINTing arrays to files. The "sticky bit", used by assembly programmers, is like the carry bit except on the other end; whenever a bit is shifted off the right end, it falls into the sticky bit. Zeros falling off the right end do not clear the sticky bit, however. Math routines set or clear the sticky bit to flag whether or not they got an inexact result, respectively. Whoever chose these names should be tarred and feathered.

To POKE a LEX file into memory from BASIC, you MUST do it at the end of the file chain in an IRAM. So first make sure you FREE :PORT(0) or another appropriate RAM device; then CREATE DATA XXX:PORT(0), then ADDR\$("XXX") to find where to begin poking. BUT to avoid problems, poke "00" at the very beginning, poke the rest of the LEX file into place, and only when you're done, poke the first two nibbles into place. That way if something goes wrong the '71 won't see the file there and freak out. Don't ever poke a file at the end of MAIN RAM; you'll overwrite the buffers floating above the file chain and die a slow, painful death.

If the above kinds of “Titan File” bits to chhu on (food for thought) inspire you, frighten you, delight you or cause any sort of reaction in you whatsoever, jot down your HP-71 ideas and send them in to CHHU so we can all learn from each other. Nary a day transpires without a phone call from some distant user asking me some bizarre and unanswerable but tantalizing question. Some fine day (Ho!) we might even see a regular column in the Chronicle filled with such '71 snippets. Meanwhile, BEEP ON!

Titan File #2 (CHHU Chronicle, V1 N3, Dec 1984, pages 29-31)

by Joseph K. Horn [13]

Contents:

1. ["TOKENIZE"](#)
2. ["MYSUBS"](#)
3. [Binary Morse Code](#)

1. TOKENIZE

Here's one thing that all serious programmers do: squeeze out all unnecessary bytes. It was easy on the HP-67; every program instruction was one byte. It got harder on the HP-41; some program lines took more than one byte, some didn't. But byte counting on the HP-71 is a real chore. First, it's nibble-oriented, not byte oriented. LOG10 takes two nibbles, but LGT takes seven nibbles. How is a mere mortal supposed to remember these weird things?

Here's a utility program for you that does the nibble counting automatically! You just key in a program line, and the program shows exactly what it looks like in hex code, and how long the line is. That way you can try several ways of doing something in BASIC, and see exactly which way is the shortest.

[Note added in 2015: TOKENIZE is available online on the [📁 HORN/MISC01 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy TOKENIZE directly to your HP-71.]

TOKENIZE

```
1 -- enter any BASIC line here --
10 SUB A @ N=256 @ DIM A$(N) @ A$=ADDR$(CAT$(0))
20 A$=PEEK$(DTH$(HTD(A$)+55),N)
30 A$=A$[1,POS(A$,"0F0100")-1]
40 PLIST 1,9 @ DISP A$;" (";STR$(LEN(A$));")"
```

You can save this program under any name you please, but I call it TOKENIZE because that's what it does.

To use TOKENIZE, first key in a program line as line 1. Lines 10 through 40 *must not be altered*, including their line numbers. Then CALL A. (Use the typing aid). Line 1 will be listed, then its hex tokenization is printed, followed by the number of nibbles in parentheses. If you wish, you may also use lines 2 through 9, but be sure to DELETE them afterwards, so that they don't bite you later.

Example: Which uses more memory, LOG10 or LGT? Type

```
1log10(a)
```

and then CALL A. The following will be printed:

```
1 DISP LOG10(A)
5C1439 (6)
```

which means that DISP LOG10(A) takes 6 nibbles. Then type

```
1lgt(a)
```

and then CALL A. You'll see:

```
1 DISP LGT(A)  
5C143B10231 (11)
```

which means that DISP LGT(A) takes 11 nibbles, 5 more than LOG10. So you should use LOG10 rather than LGT. (They are the same function, with different names).

Restrictions: The total number of nibbles used to represent lines 1 through 9 must not exceed 256. If you really need more than that, change the N=256 in line 10 to as big as you need.

Every BASIC program line ends with 0F (as seen with PEEK\$), the EOL token. And every line number is 4 nibbles long. To make the contents of the program line easier to see, TOKENIZE omits the first line number and the final EOL byte. If more than one line is entered, the intervening program lines and EOL tokens will be displayed and included in the count. This allows length comparison between single-line and multi-line approaches.

Besides program optimization, I've often used TOKENIZE to see the structure of complex statements like labels, GOTO's, and so on. It's quicker than finding it in the IDS, and then trying to figure out what the IDS means!

2. MYSUBS

Have you noticed that you use your HP-71 for basically the same tasks day in and day out? And those tasks are unique to yourself, no doubt. Most of them are done by BASIC programs that you keep in memory all the time, right?

Here is an idea from Paul J. Burke (CHHU #129). Rather than cluttering up memory with lots of custom programs, put all your "standard" programs into a single BASIC file, and make them all *subprograms*.

There are several benefits of keeping your "permanent" programs in subprogram form within one BASIC file. First and foremost, you can CALL them from any other program. Of course, you can CALL a regular BASIC program from another program, but you can't pass parameters that way. You can pass parameters to a subprogram.

The second benefit of a subprogram library file is that the search time for filenames is reduced. Rather than a proliferation of little BASIC files, you have just one BASIC file containing all the subprograms. So when you call up a file, the HP-71 doesn't have to wade through all your little programs.

The third good point of keeping a subprogram library is that you can copy them all onto a medium (cards, tape, disk or barcode?) in one swell foop, and back again. As you may have noticed, there is no easy way of copying a bunch of files at once from/to tape. Of course, you can write a program that does it, but keeping them in a single file eliminates that need. (Who will write the first LEX file that lets us COPY files using a wildcard filespec???)

Here's my own personal subprogram library. I call it MYSUBS, but a better name might be SUBS13 to "tag" it with my CHHU member number. How to use them and how they work are described

after the listing.

[Note added in 2015: MYSUBS is available online on the [📁 SWAP/atchco LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy MYSUBS directly to your HP-71.]

MYSUBS

```
10 SUB PROT(N$,P$)
20 FORTHX "N!",POS("SPE",UPRC$(P$)),HTD(ADDR$(N$))+20
30 SUB STK(X) @ CALL ONOFF
40 X=IP(MAX(1,MIN(X,16))
50 A=HTD(REV$(PEEK$("2F576",5)))
60 E$=REV$(DTH$(A+X*6)) @ POKE "2F580",RPT$(E$,3)
70 POKE DTH$(A),RPT$("000300",X) @ POKE "2F976",DTH$(X-1)[5]
80 CALL ONON
90 SUB GCD(A,B,G)
100 G=MOD(A,B) @ IF G THEN A=B @ B=G @ GOTO 100 ELSE G=B
110 SUB REDUCE(A,B) @ CALL GCD((A),(B),G)
120 A=A/G @ B=B/G
130 SUB D2F(F1,E,A,B)
140 F=F1 @ A0=0 @ A1=1 @ B0=1 @ B1=0
150 C=IP(F) @ A=A1*C+A0 @ B=B1*C+B0
160 IF A>1.E+12 OR B>1.E+12 THEN DISP MSG$(2) @ STOP
170 Q=A/B @ IF Q=F1 THEN STOP
180 IF -LOG10(ABS(Q-F1))>=E THEN STOP
190 IF C=F THEN STOP
200 F=1/(F-C) @ A0=A1 @ A1=A @ B0=B1 @ B1=B @ GOTO 150
210 SUB ALT @ RESTORE IO @ Q=DEVADDR("%35") @ IF Q<1 THEN
DISP MSG$(255034) @ STOP
220 SEND UNT UNL LISTEN Q MTA DDL 6 DATA 18 SDC
230 SUB BOLD @ OUTPUT :DEVADDR("%35") ;CHR$(14);
240 SUB LIGHT @ OUTPUT :DEVADDR("%35") ;CHR$(15);
250 SUB ULON @ OUTPUT :DEVADDR("%35") ;CHR$(27);"&d";
260 SUB ULOFF @ OUTPUT :DEVADDR("%35") ;CHR$(27);"&d@";
270 SUB PITCH6 @ OUTPUT :DEVADDR("%35") ;CHR$(27);"&k1S";
280 SUB PITCH12 @ OUTPUT :DEVADDR("%35") ;CHR$(27);"&k0S";
290 SUB PITCH10 @ OUTPUT :DEVADDR("%35") ;CHR$(27);"&k3S";
300 SUB SKIPPERE @ OUTPUT :DEVADDR("%35") ;CHR$(27);"&l1L";
310 SUB WRAP @ OUTPUT :DEVADDR("%35") ;CHR$(27);"&s0C";
320 SUB CHARCHOP
330 GDISP CHARSET$ @ IF KEYWAIT$="#38" THEN STOP
340 CHARSET CHARSET$[1,LEN(CHARSET$)-6] @ GOTO 330
350 SUB SAVEKEYS @ PURGE MYKEYS @ COPY KEYS TO MYKEYS:PORT
360 SUB SHRINK @ FORTHX ""
370 S=(VAL(CAT$(1)[19])-1500)*2 @ IF S<1 THEN STOP
380 FORTHX"SHRINK DROP",S
390 SUB BLDSPEC @ DIM A$[8],B$[6] @ FOR X=1 TO 6 @ INPUT A$
400 B$[X]=CHR$(BVAL(A$,2)) @ NEXT X @ CHARSET CHARSET$&B$ @ GDISP CHARSET$
410 SUB FLASH @ POKE "2E3FF","3"
420 SUB ONOFF @ POKE "2F441","1"
430 SUB ONON @ POKE "2F441","0"
```

How to use: Since these subprograms are like independent programs, you can type any one or more into memory and leave the others out, EXCEPT for REDUCE which requires GCD's presence,

and STK which uses ONOFF and ONON. A few require keywords that are not found in the built-in BASIC vocabulary, but can be added by copying the appropriate LEX file into memory.

PROT: FORTH ROM REQUIRED. Subprogram to change the protection of any file. Syntax: CALL PROT(filename string, desired protection string). Example: CALL PROT("ULAM", "S") makes the file called ULAM a Secure file, regardless of its former protection. Does not work on files with lowercase letters in their names, nor on files in ROM.

STK: Requires ONOFF and ONON subprograms, and User's Library Lex file functions REV\$ and RPT\$. Sets the command stack to be any height from 1 to 16 high. Syntax: CALL STK(height). Example: CALL STK(15) sets the command stack to be 15 entries high. Note: don't use 16; the HP-71 has a bug at that height.

GCD: Finds greatest common divisor of two numbers. Syntax: CALL GCD(first number, second number, numeric variable). Example: CALL GCD(A,B,G) finds the greatest common divisor of A and B and puts it into G. But A and B get thrashed. If you want to keep their values from getting changed, put them in parentheses like this: CALL GCD((A) , (B) , G). That'll just pass their values, and leave A and B themselves intact.

REDUCE: Requires GCD subprogram. Reduces the fraction A/B to lowest terms. Syntax: CALL REDUCE(numerator, denominator). Example: After setting A=153 and B=493, CALL REDUCE(A,B) changes A to 9 and B to 29, because 9/29 is 153/493 reduced to lowest terms.

D2F: Requires User's Library Lex file function MSG\$. Converts decimal fractions into regular fractions. Syntax: CALL D2F(decimal fraction, digit accuracy, numerator variable, denominator variable). Example: CALL D2F(PI,6,A,B) puts 355 in A and 113 in B because 355/113 is equal to PI to 6 digits.

ALT, BOLD, LIGHT, ULON, ULOFF, PITCH6, PITCH12, PITCH10, SKIPPERF, WRAP: Require HPIL ROM and HP2225B ThinkJet Printer. Syntax: CALL the desired subprogram to put the printer in the desired mode. CALL ALT places the printer in "alternate controls mode". CALL BOLD sets bold print mode. CALL LIGHT undoes bold mode, back to light printing. CALL ULON turns on underline mode. CALL ULOFF turns off underline mode. CALL PITCH6 sets expanded mode (6 chars/inch). CALL PITCH12 sets normal pitch (12 chars/inch). CALL PITCH10 sets expanded/compressed mode (about 10 chars/inch). CALL SKIPPERF makes the printer skip perforations automatically. CALL WRAP turns on wrap-around mode. The ALT subprogram exits normally if printer is not found; the other subs error out.

CHARCHOP: Requires User's Library Lex file keyword KEYWAIT\$. Chops any number of characters off the end of the current custom character set (CHARSET\$). Syntax: CALL CHARCHOP, then press any keys to chop characters off, one at a time, or press ENDLIN to exit. Example: Suppose your character set is three characters long, consisting of the three parts of the HP logo: a left bracket, the little "hp", and the right bracket. CALL CHARCHOP displays the logo. Press SPC (or any key except ENDLIN) and see the right bracket vanish. Press SPC again and "hp" vanishes. Press ENDLIN and the subprogram exits, leaving only the left bracket in the character set (and in the display).

BLDSPEC: Requires MATH ROM. Facilitates the building of custom-designed characters. Syntax: CALL BLDSPEC, then input six dot-column numbers *in binary* (least significant bits correspond to the top of the dot column), one at a time. After sixth number is input, the subprogram exits and the entire character set is displayed. (If the new character looks bad, CHARCHOP it and try again). Example: the "hp" in the HP logo can be put in this way: CALL BLDSPEC, then input 11111, 100, 11100, 11111000, 101000, and 111000 pressing ENDLIN after each. See "hp", which is now available as a special character.

FLASH: Sets display flash mode. Syntax: CALL FLASH *after* the desired display is DISPed or GDISPed (or PRINTed with no printer on the loop). Flash mode during a running program stays on until the next display update. Flash mode remains on if a program that sets it halts without updating the display, but the next keystroke turns it off. (This nifty feature is an award for those hardy enough to read this far).

ONOFF: Disables the ON key from being able to interrupt running programs or machine code routines. Syntax: CALL ONOFF. Note! Be sure to re-enable the ON key as soon as the necessity for it being disabled is past; otherwise machine lock-up is possible and escapable only by the [ON]-[/] interrupt which would exit the program and leave the ON key disabled!

ONON: Re-enables the ON key after being disabled by ONOFF. Syntax: CALL ONON. Example: CALL ONOFF @ BEEP 100,10 @ CALL ONON. After pressing ENDLINE, press ON and try to interrupt the BEEPing. As you see, holding down the ON key when it's disabled only suspends execution momentarily, but resumes as soon as it's released. After ten seconds, the beep finishes and ON is re-enabled. Suggested use: disable ON whenever the user should not suspend the program, such as between POKEs that must go together, or during the use of FORTH commands from BASIC.

Of course, your subs will differ from MYSUBS. You may wish to include some of the excellent subprograms in the HP User's Library Solutions "Utilities" Book. If you have a little program that you always keep in memory, put it in your subprogram library. You'll be glad you did!

PS: As you may have noticed, the END SUB statement is unnecessary. Leaving out "@ END SUB" saves 5½ bytes!

3. BINARY MORSE CODE

It was two-thirty in the morning. All the students here at our boarding school were fast asleep, and I was busy with some inane nightmare, no doubt. All I remember is suddenly sitting upright in bed, fully awake in an instant (a rare thing for me!), and saying "*Put a bit in front!*" Leaping from bed and running to my desk, I jotted down these immortal words, and went back to bed.

The next morning I found the note and puzzled over it. I remembered writing it, but had no idea what it meant. Something about horses, perhaps? A groggy nightthought about my receding hairline? Then it came back to me: ah, yes, Morse Code.

The day before, I'd had the brainstorm that Morse Code is made up of dots and dashes, which can be represented as zeros and ones, thus turning every letter into a binary number. But how do you tell the difference between 0 and 00 and 000? Anybody with any common sense would have come up with it right away, but I thought for at least 30 seconds and finally gave up.

The HP-41 solves the problem of leading nulls by putting a dummy byte in front, and then stripping it off later. This fact no doubt slowly percolated into my subconscious from the mud puddles of my sleep, and the monster in my nightmare took it and threw it at me right between the eyes, prompting my sudden awakening and subsequent cryptic message.

Here is a BASIC program that takes text as input, and plays it back in Morse Code. The utility of this program is unknown; if anybody still practices copying Morse Code in this post-Morse technological age, then they might enjoy it. But for me it was a challenge, and here's the outcome. I hope that the timing factors are correct; if not, change 'em.

What makes this program different from others is that there is no extensive look-up table for the character-to-code conversion. Every program I've seen had huge DATA lines or worse. Instead, I simply put all the Morsible characters into a single string. The position of each character in the string corresponds to that character's Morse code. All you have to do is convert the position into binary, and strip off the leading bit. Zeros are dots, and ones are dashes. The method is so simple that it can't be new, but I had fun re-discovering it.

INSTRUCTIONS: Run the program. Input the desired message (the default is the previously entered message) and the desired timing factor (default is the current value). The larger the timing factor, the slower the transmission; fractional inputs are allowed. Try a timing factor of 2 for starters. Only capital letters and spaces are allowed as text.

[Note added in 2015: The MATH ROM version of MORSE is available online on the [📄 HORN/MAIN01 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy MORSE directly to your HP-71.]

Morse Code, MATH ROM version (Math ROM Required!)

```
1 ! Binary Morse Code, Math ROM version
10 DIM A$[96],T$[29],B$[5]
20 T$="-ETIANMSURWDKGOHVF-L-PJBXCYZQ"
30 INPUT "MESSAGE? ",A$;A$ @ INPUT "TIME? ",STR$(T);T
40 FOR X=1 TO LEN(A$) @ IF A$[X,X]=" " THEN WAIT .21*T @ GOTO 90
50 B$=BSTR$(POS(T$,A$[X,X]),2)
60 FOR Y=2 TO LEN(B$)
70 IF VAL(B$[Y,Y]) THEN BEEP 800,.09*T ELSE BEEP 800,.03*1
80 WAIT .03*T @ NEXT Y @ WAIT .09*T
90 NEXT X @ END
```

[Note added in 2015: The HPIL ROM version of MORSE, called MORSE2, is available online on the [📄 HORN/MAIN01 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy MORSE2 directly to your HP-71.]

Morse Code, HPIL version (HPIL ROM Required!)

```
1 ! Binary Morse Code, HPIL ROM version
10 DIM A$[96],T$[29],B$[5] @ L2=LOG(2)
20 T$="-ETIANMSURWDKGOHVF-L-PJBXCYZQ"
30 INPUT "MESSAGE? ",A$;A$ @ INPUT "TIME? ",STR$(T);T
40 FOR X=1 TO LEN(A$) @ IF A$[X,X]=" " THEN WAIT .21*T @ GOTO 90
50 C=POS(T$,A$[X,X])
60 FOR Y=IP(LOG(C)/L2)-1 TO 0 STEP -1
70 IF BIT(C,Y) THEN BEEP 800,.09*T ELSE BEEP 800,.03*T
80 WAIT .03*T @ NEXT Y @ WAIT .09*T
90 NEXT X @ END
```

Titan File #3 (CHHU Chronicle, V2 N1, Jan/Feb 1985, pages 21-23)

by Joseph K. Horn [13]

HP-71 FILES, PART I of III

[So you want to use HP-71 files? Then read this article through, word for word, and work the examples each time. I personally guarantee you'll then be able to do it!]

One of the most confusing chapters of the *HP-71 Owner's Manual* is Section 14, "Storing and Retrieving Data," all about how to use "files". After reading that section, I was convinced that the book was written by somebody who never used an HP-71! The book even implies that there are two kinds of files: sequential-access files, and random-access files, which is not at all true!

Let's make it all very simple, and discuss one thing at a time. This month we'll cover SDATA files. In the future we'll cover DATA files (a different animal), and then TEXT files.

WHAT IS AN SDATA FILE?

"SDATA" stands for "Stream Data" and is a file type designed for compatibility with the HP-41. But don't let this suggest that SDATA files are only for HP-41 owners! In fact, I use SDATA files more than DATA files, yet I've *never* placed HP-41 data into them! You may even find SDATA files the most useful file type the '71 offers.

VARIABLES ARE TOO VARIABLE!

Suppose you want to create a list of 100 numbers. If the data is of a temporary nature, use an array. To create a 100-element array called A, use the command:

```
DIM A(100)
```

Trouble is, an array, like any other variable, can get blown away in a flash by any of the following statements: DIM, REAL, SHORT, INTEGER, COMPLEX, DESTROY, or STAT, and unless passed as a parameter, is unavailable to subprograms.

HOW TO CREATE AN SDATA FILE

If you want to keep your data in a less volatile location, and have it available to anybody who wants to use it, then an SDATA file is your answer. To create a 100-element SDATA file called MYDATA, use the command:

```
CREATE SDATA MYDATA,100
```

This creates 100 "records" numbered 0 through 99, and fills them with zero for you.

HOW TO ASSIGN A CHANNEL NUMBER

Rather than referring to your SDATA file by its full name, the HP-71 allows you to assign a number to it, and then refer to the file by that number. This handy "nickname" number is called the file's "channel number." Use television channel numbers as an analogy: you don't have to dial a station's full name (e.g. KNBC, WWTW, etc.), but can merely dial channel #2.

To assign a channel number to your file, use the command:

ASSIGN #1 TO MYDATA

From now on, you don't have to refer to "MYDATA" again, but can merely refer to channel #1. (Note: Other versions of the BASIC language have an "OPEN" command. In HP-71 BASIC, "ASSIGN" performs the same operation.) You can have more than one channel assigned at one time, each one to a different file.

HOW TO RECALL A RECORD

The 100 numbers in "MYDATA" (whoops; I mean in #1) are called "records", one number per record (the HP-41 calls them "registers", and the British call them "stores"). The records are themselves numbered sequentially starting at 0 (for the first record) on up. It would seem that the first record ought to be called record 1, but it isn't, and I can't tell you why. That's just life.

To read the value of any particular record, you must specify three things: the channel number of your file, the record number, and a variable that you wish to "catch" the record's value. For example,

```
READ #1,10;X
```

reads the value of record 10 (in channel #1) and stuffs it into the variable X. You can then PRINT X or do whatever you want with it. Don't worry; this does not "assign" X to record 10! Changing the value of X will *not* change the value of record 10.

HOW TO STORE A RECORD

Storing a value into a record is similar:

```
PRINT #1,10;X
```

"prints" the value of X into record 10, replacing whatever had been there. Note that this does *not* "insert" X into your file, like inserting a card into a deck. It *replaces* record 10's old value with the value of X. (We'll see an easy way of inserting/deleting records below). You don't have to use a variable; you can use a literal number instead of "X".

HOW TO FIND HOW MANY RECORDS EXIST

When PRINTing or READing records to/from an SDATA file, you may use any record number at any time, provided that it exists. This may seem obvious, but it isn't true of DATA files! DATA files can be a pain! As I said, SDATA files are very nice. If you forget how many records your SDATA file contains, an easy way to find out is to CAT it, and divide its byte length by 8. For example, if you type CAT MYDATA (sorry, CAT doesn't use channel numbers!), you'll see

```
MYDATA    SDATA    800
```

in the display. This means that MYDATA is an SDATA file, and is 800 bytes long. Divide 800 by 8, and you get 100, the number of records it contains.

Under program control, some people like to READ up a file until an error is caused, thus finding how many records exist. If you know where your file exists in the CATALOG, you can use CAT\$ to extract the file's size. I personally always reserve record 0 to contain how many records there are, and then use the other records for my actual data.

STORING/RECALLING MORE THAN ONE NUMBER AT A TIME

There are some tricks I haven't mentioned. If you want to store or recall more than one number at once, you can do it in a single command! For example,

```
READ #1,12;X,Y,Z,T
```

reads record 12 into X, record 13 into Y, record 14 into Z and record 15 into T! Likewise,

```
PRINT #1,12;X,Y,Z,T
```

stores X into record 12, Y into record 13, and so on. This is very handy when you have a handful of variables that you wish to store and recall as a group.

But even better is the ability to store and recall whole arrays in one fell swoop. Suppose you have the array of 100 numbers mentioned above, from `DIM A(100)`. You wish to recall MYDATA into the entire array. All it takes is:

```
READ #1,0;A()
```

to read all 100 values! (The parentheses indicate that A is an array, but they are optional. I never use 'em myself.) The HP-71 recognizes A as an array, and simply keeps reading from channel #1 until the array is filled. Of course, if the array is bigger than the file, you'll get an error message. By the way, you can mix array variables and regular variables in a single READ command, like `READ #1,15;J(),K,G(),P(),Q`. Just make sure you don't run off the end of the file!

PRINTing arrays is just as easy:

```
PRINT #1,0;A()
```

stores all of array A into the file, starting at record 0. Again, the parentheses are optional, but are a good idea in programs that may be read by humans. If you wish to store or recall a matrix, put a single comma between the parentheses like this `(,)` to indicate a matrix. Or you can omit the parentheses altogether, which I find less confusing.

Since an SDATA file is simply a linear list of numbers, you can store an array and recall it as individual numbers, or mix & match however you like. The fact that you stored single numbers, or arrays, or matrices, is remembered only by you. The file only contains the numbers themselves.

SEQUENTIAL ACCESS

So far, when PRINTing or READing to/from an SDATA file, we've been specifying the record number desired. This is called "random access" to the file, because we are able to access any record we randomly desire. But notice something!

When we `READ #1,12;X,Y,Z,T` we didn't tell the '71 to place record 13 into Y; it did it automatically. We specified record 12, which placed the "file pointer" there. After reading record 12 into X, the '71 automatically moved the pointer forward one record, and thus read record 13 into Y. And so on, until it read record 15 into T, *and once again moved the pointer forward one record*. So after this command is finished, the pointer is sitting on record 16. Suppose we then give the command:

```
READ #1;L
```

What do you think will happen? We didn't specify which record to use! But the pointer is on record 16, so of course record 16 gets read into L, and then the pointer is moved ahead to record 17. This is called "sequential access" because it accesses the records sequentially, one after the other. Notice that there is no such thing as a sequential file or a random-access file! Any file can be accessed randomly or sequentially.

HOW TO MOVE THE FILE POINTER

To place the file pointer wherever you want it, use the RESTORE command. To point at record 57, for example:

```
RESTORE #1,57
```

And from there you can access the file sequentially to your heart's delight. But you can't READ past the end of the file, of course! If you try, you'll get an error.

HOW TO EXPAND A FILE

BUT!! Here's the magic of sequential access. You *can* PRINT sequentially past the end of the file! Suppose we do the following:

```
READ#1,99;X
```

That reads record 99 (the last record) into X, and moves the file pointer to record 100 (which doesn't exist). Then if we

```
PRINT #1;PI
```

the '71 performs a miracle! It does not give an error! It actually expands the file to 101 records long, and places the value of PI into record 100 (and moves the pointer to record 101)! Now if you CAT MYDATA, you'll see that it is 808 bytes long. All the hassle of memory shifting etc. is automatically done for you! PRINTing more than one number (as a list or as an array) also works past the end of the file!

Notice, however, that RESTORE is a random-access type of animal, and so you cannot RESTORE the pointer to a non-existing record. The only way to set the pointer "above" the file is by READING or PRINTing the highest-numbered record. This makes sense, since this is normally done by a previous sequential PRINT.

HOW TO INSERT/DELETE A RECORD

When you combine the ability to expand a file with the ability to PRINT a whole array at once, you get the ability to *insert* a new record into the file. Suppose our file is 100 records long, and we wish to insert a new record, X, between records 52 and 53 (thus becoming the new record 53, and raising the old record 53 up to record 54 and so on). All it takes is:

```
DIM T(100-53+1) [we need to move up records 53 through 100]
READ #1,53;T [read records 53 through 100 into array T]
PRINT #1,54;T [write them back into records 54 through 101]
PRINT #1,53;X [insert the new data into record 53]
```

which takes about 0.7 seconds to execute! No need for a special INSERT command! READING an array and PRINTing it somewhere else allows not only insertion, but also deletion, rotation, and more!

HOW TO DE-ASSIGN A CHANNEL NUMBER

To “close” a file and de-assign its channel number, you may use the `ASSIGN` command like this:

```
ASSIGN #1 TO ""
```

This assigns channel #1 to “nothing” and makes MYDATA unavailable to `READ` and `PRINT` commands until assigned again. Notice that the `END` command also closes all open files. Of course, `PURGE`ing an open file not only closes it (for good!) but also de-assigns its channel number.

Note well! Whenever HP-71 memory is “reconfigured,” all open channels get closed! This is not obvious in the Owner’s Manual. Memory reconfiguration occurs in three instances: when `FREE PORT` is executed, when `CLAIM PORT` is executed, and when the contents of any plug-in ports are changed (except plugging or unplugging the card reader, which is not a soft-addressed device). So don’t do any of these when files are open, or they’ll get closed, possibly resulting in a suspended program failing to be able to `CONTINUE`.

SDATA TIDBITS

There are dire warnings in the Owner’s Manual about “end-of-file markers” and “loss of data” and other bizarre things. Pay no attention to it. None of it applies to SDATA files, just DATA files. As I said, SDATA files are very nice!

Now that you know how to create and access SDATA files, don’t forget that they are files, with all that implies. They can be copied under other names, copied to IRAM for safekeeping, or to an external medium for even safer keeping. They can be `SECURED` to allow them to be `READ` but not `PRINTed` to. (Note: The protection of a file when it is `ASSIGNED` is the protection you’ll get. So if you `ASSIGN` a file and then `SECURE` it, you can still `PRINT` to it!) You can’t `PRIVATE` an SDATA file, because that would be dorky.

One thing was left unsaid about SDATA files: You can only store numbers in them, not strings (through “normal” operations). If you have an HP-41 and its HP-IL module, you can have fun dumping numbers AND alpha strings into SDATA files on the HP-71!

Next month: DATA files, which can contain numbers and/or strings.

Titan File #4 (CHHU Chronicle, V2 N2, Mar 1985, pages 30-31)

by Joseph K. Horn [13]

HP-71 FILES, PART II of III

[Last month we covered SDATA files, the most easy to use HP-71 file type and perhaps the most useful. The following article will presuppose that you read the previous one. If you are a new member, you may wish to order back-issue reprints!]

WHAT IS A DATA FILE?

As you recall, SDATA files can only store numbers, not words, names, street addresses, and so on. If you need to store a *mixture of numbers and text* in a file, then a DATA file is what you need! (TEXT files, to be covered next month, are for applications that just need to store text and maybe a number or two).

You also remember that an SDATA file can only hold *one* number in each record. Guess what! DATA files can hold as many items (numbers or words) as you wish. This allows you to keep logically related items all together in one record.

HOW TO CREATE A DATA FILE

Unlike SDATA files, whose records are always 8 bytes long, DATA files' records can be as short or as long as you like. Unfortunately, this means that you have to figure out how long you want them to be! It's not that difficult; you just figure how many numbers and words you wish to store in each record. Here's an example.

Let's say we want to write a Telephone Number program that stores the names of friends and their telephone numbers in a DATA file. Since DATA files allow multiple items in each record, let's put each person's name and telephone number in one record. So we would like to store "Richard J Nelson" and his telephone number 7145490581 all in one record of a DATA file.

We must first answer the vital question: *What is the maximum length of the text we wish to store in each record?* If you can't answer that because you have no idea how long your text might be, then you shouldn't use a DATA file; use a TEXT file, which doesn't care how long your text is.

In our case of a telephone number program, let's agree to limit the names to 21 letters long. The rule for record length in a DATA file is:

Total Maximum String Length	PLUS
3 for each string	PLUS
8 for each number.	

Since we have one string of 21 characters, and one number, our telephone DATA file needs $21 + 3 + 8$ bytes per record, for a grand total of 32 bytes! (Your own custom DATA files will of course have different record lengths).

Let's name the phone data file "PDATA". But before you create a file you have to decide how many records you want! Let's reserve room for 50 names and phone numbers. RECAP: we want a DATA file called PDATA with 50 records of 32 bytes each. Here's how it's done:

```
CREATE DATA PDATA,50,32
```


Just remember to put the number of records first, then the record size. Since the number of records is logically more important, it is easy to remember that it comes first. Also, remember that when you create SDATA files, you *only* specify the number of records, because record length is automatically set to 8 bytes (enough for one number).

HOW TO ASSIGN A CHANNEL NUMBER

You assign a channel number to DATA files in exactly the same way as SDATA files:

ASSIGN #1 TO PDATA

From now on, you refer not to "PDATA" but to #1.

RANDOM ACCESS STORAGE

To store a record, write its entire contents to the file at once. For example,

PRINT #1,5;"Phineas McLanagan",1234567

writes Finny's name and phone number into record #5. Notice that the first item is a string, and the second is a number. You must keep track of which items are of what type, so that recalling them into the right type of variable is easy.

SEQUENTIAL ACCESS STORAGE

It can be done, but it makes a mess out of the file, with strings spanning records and other nastiness. Unless you know precisely what you're doing, don't try it.

RANDOM ACCESS RECALL

To pull out the contents of a particular record:

READ #1,5;N\$,P

reads record 5's contents into two variables, N\$ and P. If the record does not contain a string and then a number, you'll get an error when you try this! If you did the above random access storage example, then N\$ now contains Finny's name, and P is his phone number.

Unlike SDATA files, attempting to recall a DATA file record that hasn't yet been written to yields an error. To avoid this, some programmers like to fill new files with blank data right away. I prefer to keep track of which records have contents and which don't.

SEQUENTIAL ACCESS RECALL

As with SDATA files, failure to specify which item you wish to recall results in the recalling of the next item. Unlike SDATA files, however, DATA file records can contain more than one item! For example,

READ #1;N\$,P

would now read record #6, since we just read record 5 above. On the other hand,

READ #1;N\$

would read only the name in record 6. Note well! After doing this, the file pointer is left in limbo,

floating between two items in a record. Then we could

```
READ #1;P
```

to get that person's phone number. Some programmers prefer to READ items this way since it sometimes saves a nanosecond of execution time. I avoid it, preferring to recall entire records into variables at once since it saves me hours of debugging time.

STORING/RECALLING MORE THAN ONE RECORD AT A TIME

Sorry! It can't be done. Remember how we could READ a whole bunch of records at once from SDATA files? You can't do that with DATA files. If you try, you'll get an error when it hits the end of the record.

Well, I lied. You can sort of do it if your data is all strings or all numbers. Then you can read an array or matrix as described for SDATA files. But remember that *storing* an array or matrix into a file is a form of sequential access, and that is guaranteed to mess up a DATA file! I had an application once that used the first half of a file randomly and the second half sequentially, but when I tried to figure out how it worked after several months of disuse I almost had a mental hernia.

MOVING THE FILE POINTER

RESTORE works on DATA files exactly the same way as it does on SDATA files. It places the pointer at the *beginning* of the specified record. For example,

```
RESTORE #1,5
```

places the pointer at the beginning of record 5.

Unfortunately, there is no way to RESTORE the pointer *within* a record. To place the pointer within a record, you must recall some items from it without recalling the entire record, as described above. This leaves the pointer sitting at the end of the last item recalled. Although it is possible, leaving pointers floating around within records is of dubious utility and is poor programming practice because you'll never understand what's happening even if you wrote it.

DATA TIDBITS

The warning against sequential access storage is because the HP-71 has a nasty feature called the "end-of-file marker." After every sequential access PRINT to the file, the HP-71 stores an end-of-file marker in the record just written. This prevents sequential recalling beyond that point. Luckily, it does not affect random access operations. The Owner's Manual overstates the dangers of the end of file marker. It does *not* result in the loss of all data beyond it! You can read sequentially up to, and after, an end-of-file marker! It just prevents a sequential READ to go *through* that point. And it is ignored by random access READs.

If you try to store too much in a record, you'll get an error. For example, if we tried to store a record into our PDATA file with a name longer than 21 letters long, the HP-71 wouldn't have enough room to do it. Programs should test for this eventuality before the HP-71 tries to store the record. If you keep getting this sort of error, re-CREATE the file with longer records. Unfortunately, there is no way of lengthening records directly. If you wish to do this, CREATE another file with the new record length, then loop through the old file, storing its contents into the new file. Then the old file can be PURGED.

Examples of the above ideas about DATA files can be found in a program called "PHONE". See

elsewhere for the listing, instructions, and comments.

Before you forget: `PURGE PDATA` to free the memory it's gobbling up!

Next month: `TEXT` files (also known as `LIF1` files).

Titan File #5 (CHHU Chronicle, V2 N3, Apr-Jun 1985, pages 36-40)

by Joseph K. Horn [13]

Contents:

1. [Ulam in Assembly](#)
2. [Titan File Q&A](#)
3. [HP-71 Phone Program](#)

1. ULAM IN ASSEMBLY!

[This is *not* part III of the HP-71 file type series. We have covered SDATA files in [Part I \(V2 N1 P21\)](#), and DATA files in [Part II \(V2 N2 P30\)](#) with an example program called "[PHONE](#)" ([V2 N3 P38](#)). Part III will cover TEXT files. But it will have to wait till next month because something irresistibly wonderful is pre-empting it. -jkh]

[Editors note. The Phone program example of using data files was supposed to have been in our last issue with part II of Joseph's file type series. Space did not permit its inclusion. It is included with this issue and [follows this month's column.](#)]

Stanislaw Ulam didn't invent it. In fact, nobody knows who did! But it is my favorite unsolved Number Theory conjecture. It is so simple to understand and play with that it is the programmer's perfect exercise.

Here 'tis: Pick any number (not too big). *Now do this.* If it's an odd number, multiply it by 3, then add 1; but if it's an even number, then simply divide it by two. This gives you a new number. Take this new number and go back to where I said "Now do this." Repeat this process UNTIL YOU REACH 1.

Let's try it with 3:

- 1) 3 is ODD, so I multiply it by 3 and add 1, which gives 10.
- 2) 10 is EVEN, so I divide it by 2, which gives 5.
- 3) 5 is ODD, so I multiply it by 3 and add 1, which gives 16.
- 4) 16 is EVEN, so I divide it by 2, which gives 8.
- 5) 8 is EVEN, so I divide it by 2, which gives 4.
- 6) 4 is EVEN, so I divide it by 2, which gives 2.
- 7) 2 is EVEN, so I divide it by 2, which gives 1; stop!

Notice that it took seven steps. We started at 3 and got all the way up to 16 before we got down to 1! Some numbers go very high, and take a long time to get down to 1. (Try 27 some time).

Matter of fact, there is no reason at all (that I can think of) that there might not be some starting number that never reaches 1 at all! It might just keep getting bigger and bigger, with a few dramatic backslidings here and there just to keep our hopes up. Or if it's really nasty, it might get caught in an endless loop, going through the same numbers over and over!

Ulam's Conjecture (its popular name) says that *all* numbers will eventually reach 1. But it's just a conjecture, because nobody's proven it yet. If you can prove it, you will earn eternal fame in the history of Number Theory!

Ulam's Conjecture is the perfect programming exercise. Whenever I am learning a new machine, I

program Ulam. My first HP-25, Microsoft BASIC, HP-67, HP-41 and HP-71 BASIC programs were all Ulam's Conjecture! And now that John Baker is kind enough to explain '71 Assembly Language to us in his column *Exploring the 71 IDS*, it is time to attack Ulam in Machine Language!

What we want, of course, is to program the Ulam process described above, which the Germans call the "Syracuse Algorithm". Let's call $S(x)$ the Syracuse function. Using this notation, the above example boils down to this:

$S(3)=10$; $S(10)=5$; $S(5)=16$; $S(16)=8$; $S(8)=4$; $S(4)=2$; $S(2)=1$.

Notice that it takes seven steps for 3 to get to 1, using the Syracuse algorithm. Programming a $S(x)$ function is easy in BASIC, but what we really need is a program that calculates $S(x)$ over and over until it hits 1, and keeps count! I want an $ULAM(x)$ function that tells me how many steps it takes x to get to 1! Using our above example, it would boil down to this:

$ULAM(3)=7$

Doing it in BASIC is easy:

```
10 DEF FNU(X) @ C=0
20 IF X<=1 THEN 40
30 IF RMD(X,2) THEN X=X+X DIV 2+1 @ C=C+2 @ GOTO 30 ELSE
   X=X DIV 2 @ C=C+1 @ GOTO 20
40 FNU=C @ END DEF
```

Enter this program, then type $FNU(3)$ and see 7, because 3 takes 7 steps to get to 1.

I am "cheating" in the algorithm here, taking advantage of the fact that $3x+1$ must be an even number if x is odd, so I just divide by 2 right away without testing (line 30). This is to optimize for speed. The algorithm is: $(3x+1)/2=x+x/2+1$ (x odd).

But try $FNU(27)$. You'll see the right answer, 111, but it takes almost 3 seconds to find it (not counting the .12 seconds of FNU overhead). After all, 27 gets *above 9000* before it finally gets down to 1! Try $FNU(537099606)$. It takes almost half a minute to get the answer of 965. That's faster than by hand, but let's do it in Assembly!

The following Assembly Language program creates a LEX file that adds a new BASIC keyword called $ULAM$. $ULAM$ is a function that takes a number and tells how many steps it takes it to get to 1 by repeated application of the Syracuse algorithm. This lets you type, for example, $ULAM(3)$ and see 7. What is startling, of course, is the speed. $ULAM(27)$ takes about 0.02 seconds (not counting the .04 seconds of $ULAM$ overhead), which is over 100 times faster than BASIC! And $ULAM(537099606)$ returns the answer in 0.12 seconds, which is over 200 times faster than BASIC!

[Note added in 2015: $ULAMLEX$ is available online on the [📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy $ULAMLEX$ directly to your HP-71.]

Saturn Source Code for $ULAMLEX$

```
LEX 'ULAMLEX'      Create a LEX file called ULAMLEX.
ID #99             Use LEX ID number 153 (hex 99).
MSG 0              No table of special error messages.
POLL 0             No special poll handlers.
                   Begin system entry point labels:
```

POP1N EQU #0BD1C	Pop argument from stack into CPU regs
RJUST EQU #12AE2	Convert floating-point to integer.
DCHXW EQU #0ECD4	Convert decimal integer to hex.
HXDCW EQU #0ECB4	Convert hex back to decimal integer.
FLOAT EQU #16322	Convert integer back to floating-point
FNRTN4 EQU #0F238	Put answer on stack & return to BASIC.
	Begin function text table:
ENTRY ULAM	Runtime code starts at label ULAM.
CHAR #F	It's a function (not a statement).
KEY 'ULAM'	Call it ULAM(x).
TOKEN 1	Make it ULAMLEX's first keyword.
ENDTXT	End of function text table.
	Begin ULAM's actual assembly code.
NIBHEX 811	First & only argument is numeric.
ULAM GOSUBVL POP1N	Fetch argument from math stack into CPU register A.
GOSBVL RJUST	Convert argument in floating-point form into decimal integer.
C=A W	Convert decimal integer into a hex integer and leave in hex mode.
GOSBVL DCHXW	
SB=0	Clear Sticky Bit for even/odd test.
B=0 W	Clear the loop counter.
D=0 W	To see if we've reached 1 yet, we need
D=D+1 B	a 1; clear D and add 1 to set D=1.
TEST C=A W	C holds our growing & shrinking #.
?C<=D W	Is C<=1 yet?
GOYES DONE	If so, we're done; otherwise:
LOOP B=B+1 W	Add 1 to the counter.
ASRB	A=A\2. This is the even/odd test;
?SB=0	if no bit falls off the right end,
GOYES TEST	it was even. Since we're supposed to divide it by 2, but already did by the ASRB, just set C=A & repeat.
	If odd, we can multiply by 3, add 1, and divide by two by C=C+A+1, since (3C+1)/2=C+C\2+1, and A=C\2.
C=C+A W	
C=C+1 W	
A=C W	Get A ready for next even/odd test.
B=B+1 W	Since we "skipped" a step by dividing by 2 right away, we must increment the counter for the skipped step.
SB=0	Clear Sticky Bit for next test.
GOTO LOOP	Repeat Syracuse algorithm.
DONE C=B W	We're done. Fetch hex counter into C.
GOSBVL HXDCW	Convert it to a decimal integer.
A=C W	
GOSBVL FLOAT	Convert it to floating-point form.
C=A W	
GOVLNG FNRTN4	Return it to BASIC.
END	by Joseph K. Horn [13] 05/13/1985

ULAM is useable in CALC mode and in programs. It will accept any input (except NaN), but it will return valid results only for positive integers less than 1E12. It carries its internal calculations out to 20 digits (=16 hex digits), so there is no danger of internal overflow as your number gets bigger and smaller in its journey to 1 (which is better than BASIC!).

To use, either type the mnemonics as shown above (with 2 leading spaces where indicated), leaving out the comments, and then assemble; or run MAKELEX (see V2 N1 P20) with the following data:

ULAMLEX	ID#99	80 bytes				
		0123	4567	89AB	CDEF	ck
000:		55C4	14D4	C454	8502	A9
001:		802E	0013	0231	5058	4E
002:		0A00	0991	0100	0000	29
003:		F710	0000	0000	0000	A7
004:		0710	00F7	55C4	14D4	74
005:		101F	F811	8FC1	DB08	0A
006:		F2EA	21AF	68FC	DCE0	F9
007:		822A	F1AF	3B67	AF69	D1
008:		FB02	B758	1C83	2FEA	85
009:		72B7	6AFA	B758	2265	E2
00A:		EFAF	98F4	BCE0	AFA8	3A
00B:		F223	B1AF	68D8	32F0	82

NOTE: Don't enter the spaces; they are only visual aids.

One reason that Ulam's Conjecture is so hard to prove is that everybody's been looking at the ULAM function. But $ULAM(x)$ is a *useless* function, mathematically! What $ULAM(x)$ tells us nothing about x . It is useless information, as useless as the answer to the question "What is the n th prime number?"

But if you modify the Syracuse algorithm just a tad, you get a useful function! Just two changes, and we have a function that gives useful information (and may be the key to proving Ulam's Conjecture!) First change: don't stop when you reach 1; stop as soon as you fall below the number you started with. Second change: Count as one "step" either the division by two (if even), or multiplying by 3, adding one, *and dividing by 2* (if odd). Since this is a Modified Syracuse Algorithm, let's call the function $MSA(x)$. $MSA(x)$ is the number of steps it takes x to fall below itself by repeatedly performing $x=x \text{ DIV } 2$ (if x is even), and $x=x+(x \text{ DIV } 2)+1$ (if x is odd).

Here's a BASIC routine that does it:

```
50 DEF FNM(X) @ C=0 @ M=X
60 C=C+1 @ IF RMD(X,2) THEN X=X+X DIV 2+1 @ GOTO 60 ELSE
   X=X DIV 2 @ IF X>M THEN 60
70 FNM=C @ END DEF
```

The $MSA(x)$ function, calculated by typing $FNM(x)$, is "useful" because it tells us something interesting about x . For example, the MSA of any even number is 1, because in 1 step all even numbers fall below themselves (they immediately get divided by 2). $MSA(5)$ is 2, because it gets to 4 in two steps, and 4 is less than the number we started with (5). Notice that this is true for every 4th number starting with 5; i.e. all numbers of the form $5+n*2^{MSA(5)}$.

$MSA(3)=4$. This means that every 16th number starting with 3 takes exactly 4 steps to fall below itself; i.e. all numbers of the form $3+n*2^{MSA(3)}$.

In general, any number x will fall below itself in $MSA(x)$ steps, and besides that x , every number of the form $x+n*2^{MSA(x)}$ will fall below itself in the same number of steps too! If we can prove that all numbers fall below themselves, it follows that all numbers will reach 1, and the Conjecture is

proved! But how does one generalize MSA for *all* numbers?

The Assembly code for ULAM need be changed in only a few places to change it to MSA. Can you do it? Here are a few values of both functions so that you can test your code:

x	ULAM(x)	MSA(x)
27	111	59
703	170	81
1537	153	2
3711	237	37
34239	310	92
35655	323	135
626331	508	176
63728127	949	376
268549803	964	5
99999999999	296	21

Thanks to John Kennedy for first bringing this problem to my attention in the *PPC Journal*, V6 N1 P9, in which he gave it its immortal name, "Ulam's Conjecture." Douglas Hofstadter spoke of it at length in his book *Gödel, Escher, Bach* (p. 401), in which he called it the "Wondrous Property" of numbers. A. K. Dewdney described it in his column "Computer Recreations" in *Scientific American* as "Hailstone Numbers", and Martin Gardner preceded him in his "Mathematical Games" column, same magazine (June 1972 p. 115), calling it a "transcendental problem easy to state but not at all easy to solve." Richard Guy describes it in his book *Unsolved Problems in Number Theory* as the "Collatz Sequence". Heppner, Möller, and Steiner all called it the "Syracuse Algorithm". Riho Terras called it "A stopping time problem on the positive integers," an aspect of the problem that Hofstadter focused on. It has been mentioned in Memo 239 from the AI Lab at MIT (1972); in the proceedings of the 7th Conference on Numerical Computation in Manitoba (1977); and in numerous magazine articles in Italy and Germany. It seems the only one who never wrote about it is Stanislaw Ulam!

My thanks to Clifford Stern of PPC whose success in programming the Conjecture in M-Code on the HP-41 challenged me to write this article. Although it is impossible to keep up with Cliff, I'll be happier when I get a prime-factor finder written in HP-71 Assembly which runs faster than Cliff's HP-41 M-Code factorizer! Can anybody out there help me? I'd appreciate it!

2. TITAN FILE QUESTIONS AND ANSWERS

(1) DOT2DOT. The DOT2DOT program (V1 N2 P18) has caused so many people pain that it has been given a name by the AMA. If you experienced any of the usual symptoms (e.g. never got it to work, or if it worked in fits and starts, or seemed to keep ejecting the paper needlessly, or performing Memory Lost at random times), then you may have "DOT2DOTitis". The disease is marked by rapidly deteriorating success at producing anything, combined with a fanatical fixation on changing the program "just a little" in vain. The prognosis is bad; besides losing a lot of vital thinkjet paper, DOT2DOTitis shortens one's life because the victim never remembers the time thus wasted.

If you want to avoid the disease, and use DOT2DOT as it was intended (trouble-free), then you must do the following:

[A] Use the program *exactly* as published. There are copies floating around that were re-written for the HP82166A converter that won't work for the HP-IL ThinkJet. Compare your listing!

[B] The ThinkJet must be the first printer-type device on the Loop. If it is not, the program may not work. If you don't want to re-arrange your Loop, then make sure you perform a PRINTER IS %35 and DISPLAY IS * before running the program! It may be a good idea to do these two anyway, no matter what.

[C] If you still have problems, send me two listings: your program, and the devices you have on the loop (and their order). I'll do my best to figure out why you have so many devices on the loop, and then I'll write back to suggest that you take them all off the loop except for the ThinkJet. To my knowledge, the disc drive, video interface, multimeter, and HP-IL Toaster-Oven have no respect for the DOT2DOT program's output.

(2) STRINGS IN SDATA FILES. In V2 N1 P21, I implied that SDATA files in the HP-71 can receive HP-41 registers over the HP-IL, even if they contain strings. I worded it this way because that's what HP said on page 99 of the HP-71 Owner's Manual: "SDATA files are data files that can be sent to and received from an HP-41 Handheld Computer."

After being asked how to do it, I realized to my chagrin that it can't. Not directly and simply, anyway. It *can* be done, but it's an involved process that a future article will discuss.

If you have an HP-41 and an HP-71, it is easy to transfer data back and forth via HP-IL using SDATA files IF AND ONLY IF you have an HP-IL mass memory storage device, such as the digital cassette, or a disk drive. If so, then you simply dump the data onto tape or disk from one machine, and read it back into the other machine according to your mass storage device's instructions. The data files created by the HP-41 have the same "file type" as HP-71 SDATA files.

If you don't have a tape or disk drive, then wait for the forthcoming article. To whet your appetite, here's a routine that stores strings (up to six characters long) into SDATA files:

```
10 SUB ASTO(F$,N,A$) @ F$=ADDR$(F$)
20 L$=PEEK$("2F7B2",16)
* 30 LOCK A$[6,6]&A$[5,5]&A$[4,4]&A$[3,3]&A$[2,2]&A$[1,1]
40 P$=PEEK$("2F7B2",16) @ POKE "2F7B2",L$
50 L$=DTH$(37+16*N+HTD(F$))
60 POKE L$,P$[1,2]&"1"&P$[3,3]&"00"&P$[4]&"0"
70 END SUB
```

Syntax: CALL ASTO("filename", record number, "string"). The specified string will get stored into the specified record of the specified file. For example, CALL ASTO("BINGO",12,"B-12") stores "B-12" into record 12 of the SDATA file named BINGO. The file need not be assigned a channel number for ASTO to work, since it uses brute-force POKEing, not "normal" file methods. CAUTION: The routine does no error checking, and if typed or used incorrectly, it can corrupt memory or cause a Memory Lost.

* Type REV\$("ABC"). If you get an error, skip this paragraph. If you see CBA, then you have the REV\$ function available, and line 30 above can be simplified to:

```
30 LOCK REV$(A$[1,6])
```

Programmers will notice that I am using the LOCK function here. Don't worry; it is only used as a temporary scratchpad. The user's password (or lack of one) is not affected.

3. HP-71 PHONE

by Joseph Horn [13]

PHONE is an HP-71 telephone number directory program. Although it is loaded with features and is quite fool-proof, it has one severe limitation: It only stores names and phone numbers, not addresses, comments, recipes, gossip, and other necessities. This was done on purpose. PHONE was written as an example of the use of DATA files, with a line-by-line analysis to help the student. Although it is a useful program (I use it all the time!), it is not intended to be a generalized data base program!

PHONE has every feature that a reasonable person could ask for. After creating a DATA file of whatever size you want, it allows you to:

- Add new names and telephone numbers
- Browse forward and backwards through the entire file
- Search for name substrings in the entire file
- Update existing records
- Delete records
- Purge the entire file and start again
- Check how many active records there are
- Display a dehydrated Command Menu

All functions are single-key menu-driven to save you time.

INSTRUCTIONS

RUN the program. If PHDAT (the telephone number DATA file) does not exist, you will be asked "How many records?" Input the maximum number of entries you will wish to keep in memory. Don't make it too small, because you cannot expand the file once it's created. On the other hand, don't make it too big or you'll run out of memory. The file will be created, and "Ready" will be displayed. This means you may press any command key.

Press **M** to see the Command Menu. The keys shown stand for commands, and pressing any of these keys activates its respective command as explained below.

Pressing **A** puts you in Add mode. Input a name up to 21 letters long. 21 was used because a glance at the left end of the display is all it takes to see if you typed too much; if any letters of the name ran off the screen then it's too long. Then input the phone number *as a number* (don't include parentheses or dashes or spaces!). Bad inputs are ignored. The record will be added and "Ready" will be displayed. If you press **A** when there is no room left for more names, "No room" will be displayed.

Pressing **F** moves you one record Forward through the file and displays the name stored there. Pressing **B** moves one record Backward. Browsing past either end of the file wraps around to the other end of the file. Browsing in an empty file yields the "No such record" message. Note: the records are not kept in any special order.

Pressing **P** at any time displays the Phone number of the current record. Pressing **N** displays the Name of the person with that phone number. If there is no current record (for example, after a deletion or when the file is empty), these functions display "No such record".

Pressing **S** puts you in Search mode. Input a string. The program then searches for the first name in the file which contains your input (upper or lower case doesn't matter). If no match is found, "No such record" is displayed. If a match is found, the name will be displayed followed by a question

mark. It is asking you if you want that person's phone number or not. Press **Y** if **Yes**; the phone number is displayed and search mode is exited. Or press **N** if **Not**, and the program will continue searching. If there are no names on file, **Search** only displays "No such record". Hint: include special "keys" in related records so that **Search** can find them together. Example: include a "+" in the names of family members; a "!" in CHHU members' names, etc. to **Search** for them easily.

Pressing **D** instantaneously **Deletes** the current record, and the message "Deleted" is then displayed. If there is no current record then attempting to **Delete** will result in "No such record" being displayed. CAUTION! The program does not distinguish between intentional and non-intentional pressing of the **D** key! Pressing **D** immediately after **Adding** a new name (when "Ready" is displayed) will result in its deletion.

Pressing **Z** performs a **Zap** of the entire file. Unlike **Delete**, **Zap** asks you if you really meant it. When asked if you wish to delete all the names on file, press **Y** if **Yes** (at which time you will be asked for a new file size as described above), or press **N** if you do **Not** wish to delete the file (at which time "Ready" will be displayed with no change to the file). **Zap** can also be used to check how many active records there are; just press **Z**, then **N**.

Pressing **U** puts you in **Update mode**. This is the same as **Add mode**, except that the current record's name and telephone number are displayed as default inputs, and the updated record replaces the current record in the file. Use **Update** whenever somebody changes their phone number, or when you wish to change the "key"s in a name.

Pressing **Q** is the proper way to **Quit** the program. The current number of active records is stored at that time and then the program ends. Exiting the program any other way will result in a corrupt and unreliable data file.

Recap of commands:

- A**dd new names & phone numbers.
- B**ackward browse.
- D**elete current record.
- F**orward browse.
- M**enu display.
- N**ame of current record.
- P**hone number of current record.
- Q**uit program.
- S**earch for name.
- U**ppdate current record.
- Z**ap entire file and start over.

LINE-BY-LINE ANALYSIS:

10) **N\$** is Name\$; **S\$** is Search\$ and Scratch\$; **P** is Phone number; **N** is Number of active records; **R** is current Record number; and **M** is Maximum records in the file.

20) If the **UNSECURE** in line 30 generates an error, that means it must not exist, so jump to the routine that creates and initializes it.

30) **PHDAT** is the **DATA** file that contains the names and phone numbers. **UNSECURE**ing it is necessary because the user may have **SECURED** it from the keyboard, and you can't update **SECURE** data files. (It also is a tricky way of testing to see if it exists). Assign channel #1 to it. Record 0 does not contain a name or phone number, but housekeeping information: the number of active records and the size of the file itself. These are read into their variables. Sequential access logic is used

because after an `ASSIGN` the pointer is at the beginning of the file anyway.

40-50) The “main loop”. If no key is pressed or if a key is pressed that does not correspond to a program label, the error trap in line 40 branches back to line 50. But as soon as a key is pressed that corresponds to a label, control passes to that label. `PHONE` spends most of its time on line 50.

60) “Quit”. Update the housekeeping data in record 0 (number of active records and file size), tell the user that we’re “Done”, and `END`, which closes the file.

70) “Phone number”. If there is no current record, tell the user so.

80) Otherwise, display the phone number in United States format, with three-digit area code in parentheses, a space, three-digit prefix, a dash, and final four digits. If the area code or any other leading digits are missing, display an asterisk in their place. Then go back to main loop.

90) The “No such record” routine used by almost every command. Jumps to main loop.

100-110) “Name”. If no current record, say so. Otherwise display the name and go back to main loop.

120) “Delete”. If no current record, say so.

130) If current record is last active record, it can be deleted merely by subtracting one from the number of active records. This is done by skipping line 140.

140) Records are deleted by taking the last active record and moving it to the record being deleted. This insures that the file has no memory-wasting “ghosts” in it, i.e. inactive records among the active ones.

150) After deletions, there is no active record, so the record pointer is set to zero. Since the last active record is now unneeded, we can delete it by subtracting one from the number of active records. A message that deletion has been accomplished is displayed, and we return to the main loop.

160-170) “Add”. If the file is full (number of actives equals number of possibles) then “No room” is displayed and we return to the main loop without allowing any new names. Otherwise the input defaults are cleared to allow a fresh name and phone number. The number of active records is incremented and the current record pointer is set to the new record.

180-190) Input a name. If the name is longer than 21 characters, the error trap returns us to the input to try again. If nothing is typed, it lets us try again too. These two input routines are used by `Add` and `Update`.

200-210) Input a phone number. If something other than a number is used, the error trap sends us back to the input to try again. `Scratch$` is used rather than the more prosaic `Phone number` variable because we can thus have fancier defaults. In this case, `Add` has a blank default, and `Update` has a numeric default.

220) Put the new (or updated) record into the file. We jump to the beginning of the main loop to display “Ready” and reset the error trap.

230) “Search”. If there are no records, don’t even try! (Tell the user “No such record”). Otherwise input the string to be searched for and convert it to uppercase.

240) Loop through all the active records.

250) If the search string is not found in the uppercase version of the record's name, then go on to the next record. If it is found, then ask the user if this is the right name.

260) If the user presses nothing or something other than Y or N, keep waiting for a key to be pressed. If "Yes" is pressed, then jump to the "Phone number" routine. If "No" is pressed, continue searching.

270) Go ahead and check the next record. If all the records have been checked, clear the current record pointer and say "No such record".

280) "Zap". Ask the user if a total wipeout is desired.

290) If the user presses nothing or something other than Y or N, keep waiting for a key to be pressed. If "No" is pressed, then say "Ready" and enter main loop without doing anything to the file. If "Yes" is pressed, then:

300) Purge entire data file and jump to the routine that creates a new one.

310) "Forward". If empty file, say so. Otherwise, move record pointer forward. If that falls off the end of the list, then wrap it back to record 1.

320) Read the current record and jump to the Name routine. This line is used by Backwards too.

330-340) "Backwards". If empty file, say so. Otherwise, move the record pointer backwards one. If this backs us up into record 0, then wrap it around to the last record. Then jump to read and display that record.

350) "Update". If there is no current record, say so. Otherwise set up the default phone number string and jump to the input routines.

360) "Menu". Display the first letters of the eleven menu commands, and jump back to the main loop.

370) The file creation routine. Ask the user how many records are desired.

380) Create the PHone DATa file with that many records plus one (for housekeeping record 0) and record length 32 (21 byte name, 3 byte string overhead, and 8 byte phone number). Assign channel #1 to it.

390) Store the initial housekeeping info in record 0 (zero active records, and the size of the file) and jump to the main loop. Sequential access logic is used because after an ASSIGN the pointer is at the beginning of the file anyway.

PROGRAM MODIFICATIONS

If you have the function KEYWAIT\$ available due to the presence of it in a LEX file in a ROM or in RAM, your batteries will last much longer if you replace the KEY\$ in lines 50, 260 and 290 with KEYWAIT\$. The program will operate as usual, but when waiting for a menu command or a Yes/No keystroke, the HP-71 is held at "idle" power drain (low) rather than at "running" power drain (high).

Also, if you wish to add functions that store/load the PHDAT file to/from mass storage, be my guest. All I have is a card reader, so I didn't add such functions.

Another nice feature you may wish to add would be a function that lists the file on a printer. Keeping the records in alphabetical order was a task that my midnight brain was not up to; Search makes it unnecessary anyhow.

[Note added in 2015: PHONE is available online on the [📁 HORN/MAIN01 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy PHONE directly to your HP-71.]

```
PHONE      BASIC      1080 bytes
-----
10 DIM N$(21),S$(21),P,N,R,M
20 ON ERROR GOTO 370
30 UNSECURE PHDAT @ ASSIGN #1 TO PHDAT @ READ #1;N,M
40 DISP "Ready" @ ON ERROR GOTO 50
50 GOTO KEY$
60 'Q': PRINT #1,0;N,M @ DISP "Done" @ END
70 'P': IF R=0 THEN 90
80 DISP USING "'('3*') '3*'-'4*';P @ GOTO 50
90 DISP "No such record" @ GOTO 50
100 'N': IF R=0 THEN 90
110 DISP N$ @ GOTO 50
120 'D': IF R=0 THEN 90
130 IF R=N THEN 150
140 READ #1,N;N$,P @ PRINT #1,R;N$,P
150 R=0 @ N=N-1 @ DISP "Deleted" @ GOTO 50
160 'A': IF N=M THEN DISP "No room" @ GOTO 50
170 N$="" @ S$="" @ N=N+1 @ R=N
180 ON ERROR GOTO 190
190 LINPUT "Name: ",N$;N$ @ IF N$="" THEN 190
200 ON ERROR GOTO 210
210 INPUT "Phone #: ",S$;S$ @ P=VAL(S$)
220 PRINT #1,R;N$,P @ GOTO 40
230 'S': IF N=0 THEN 90 ELSE LINPUT "Search: ";S$ @ S$=UPRC$(S$)
240 FOR R=1 TO N @ READ #1,R;N$,P
250 IF POS(UPRC$(N$),S$)=0 THEN 270 ELSE DISP N$;"?"
260 ON POS("YN",KEY$)+1 GOTO 260,80,270
270 NEXT R @ R=0 @ GOTO 90
280 'Z': DISP "Zap ALL";N;"names?"
290 ON POS("YN",KEY$)+1 GOTO 290,300,40
300 PURGE PHDAT @ GOTO 370
310 'F': IF N=0 THEN 90 ELSE R=R+1 @ IF R>N THEN R=1
320 READ #1,R;N$,P @ GOTO 110
330 'B': IF N=0 THEN 90 ELSE R=R-1 @ IF R<1 THEN R=N
340 GOTO 320
350 'U': IF R=0 THEN 90 ELSE S$=STR$(P) @ GOTO 180
360 'M': DISP "A,B,D,F,M,N,P,Q,S,U,Z?" @ GOTO 50
370 OFF ERROR @ INPUT "How many records? ";M @ N=0
380 CREATE DATA PHDAT,M+1,32 @ ASSIGN #1 TO PHDAT
390 PRINT #1;N,M @ GOTO 40
```

Titan File #6 (CHHU Chronicle, V2 N4, Jul/Aug 1985, pages 21-22)

by Joseph K. Horn [13]

HP-71 FILES, PART III of III

[In [V2 N1 P21](#), we discussed SDATA files. In [V2 N2 P30](#), we covered DATA files. We were supposed to hit TEXT files last issue (V2 N3) but I had a mathematical spasm, and wrote about Ulam's Conjecture instead. Beginners, re-awake!]

WHAT IS A TEXT FILE?

Remember SDATA files? They only store numbers. And their records are a fixed length, eight bytes long. You can only store one number per record. That's why they're so easy to use.

Remember DATA files? They are more powerful than SDATA files, but they're not as easy to use. They also have fixed-length records, but you have to specify the length yourself. You can store more than one number in each record, and you can even store text in a DATA file. All these features were covered in previous issues of the Chronicle (see above).

The TEXT file (also called LIF1 file) is another file type, and is very different from the SDATA and DATA file. The main difference is that TEXT files (as the name implies) are designed to hold text, not numbers (although they can). The second difference is that records in TEXT files are not fixed-length. Records in TEXT files are of "variable length". That means that in the same TEXT file you can find records of different lengths. This makes the TEXT file a very different animal.

If you have an HP-75, you already know that TEXT files on the HP-71 correspond to LIF1 files on the HP-75, and can be used to transfer files between the two machines. Since you already know that, I won't go into it here.

HOW TO CREATE A TEXT FILE

This is easier than for SDATA or DATA files! All you have to do is pick a name. No numbers to worry about! If we want to create a TEXT file called NOTEPAD, all it takes is

```
CREATE TEXT NOTEPAD
```

The file will be found in the CATalog, but its length will be 0. That's because we haven't put anything into it yet. It automatically grows as we add text to it.

HOW TO ASSIGN A CHANNEL NUMBER

Just as with any other file, you must, assign a channel number to a TEXT file before you can store to it or recall from it. To assign channel #1 to NOTEPAD, just

```
ASSIGN #1 TO NOTEPAD
```

From now until channel #1 is closed, we do not refer to our file as "NOTEPAD" but as #1.

HOW TO STORE TEXT

To add a new line to the end of the TEXT file, just PRINT it. YOU CAN'T STORE RANDOMLY. Only sequential storage is possible with TEXT files, because of the variable-length record structure.

So be sure you build the whole file at the time you need it and in the order you need it! There is no way (without fancy LEXfile functions) to perform updates to the middle of a TEXT file.

Say we want to put the following three “notes” into our NOTEPAD file:

- (1) Dinner with Paul Delman September 18: Lawry’s, L.A.
- (2) Stuffing Party at CHHU meeting Sep 20: Emulex, S.A.
- (3) Titan File #6 due, Sep 22.

You could just PRINT #1, "Dinner with..." etc., but let’s

```
DIM A$[96] @ FOR X=1 TO 3 @ LINPUT A$ @ PRINT #1;A$ @ NEXT X
```

instead. That asks you for three lines of text input, and stores them into NOTEPAD. Change the 3 to INF, and you can type text to your heart’s content, and when you’re done, press ATTN. The file will nicely contain all your lines of text.

SEQUENTIAL ACCESS RECALL

Since TEXT files have variable length records, they are really designed for sequential access recall. To recall the record at the current file pointer’s position, just

```
READ #1;A$
```

Just be sure that A\$ is dimensioned big enough to “catch” the entire record. There is no limit to the length of a TEXT file’s record; it is as long as you made it when you PRINTed it.

Of course, this not only reads the current record into A\$ but also moves the pointer forward to the next record, ready for the next READ.

RANDOM ACCESS RECALL

HP was nicer than nice with this one. Most computers don’t allow it, but the HP-71 does allow random access recall (only) of TEXT files. Say we want record 2:

```
READ #1,2;A$
```

BUT WAIT A MOMENT! Think. Random access recalling of SDATA or DATA files is easy for the HP-71 to do, because it just multiplies the record length times the number of the record you want, and blammo, it knows where the record begins.

But how on earth is random access recall of a TEXT file supposed to work? The HP-71 has no idea where your desired record begins. The only way it can possibly do it is by the tedious method of starting at the first record, checking its length, jumping to the end of that record, checking the next record’s length, jumping over it to the next one, and so on, leap-frogging over record after record until at long last the desired record is found.

This is in fact the way the HP-71 “pretends” to allow random access recall of TEXT files. If you only have a dozen records or so, it will almost seem like it really was random access! But if you get a TEXT file with several hundred records, you’ll notice a pause as the file pointer is leap-frogging to your high-numbered records.

A small price to pay for this tremendous convenience!

If this slows your application program down, then keep in mind that sequential access recalling of TEXT files does not have this drawback. If you can do it sequentially, do it. Only use random access when absolutely necessary, or when speed doesn't matter.

MOVING THE FILE POINTER

Same as with SDATA files. Just

```
RESTORE #1,1234
```

to set the pointer to record 1234. Don't forget that record 0 is the first record! (Why they do it that way is a great mystery. The first line of a BASIC program isn't 0. The first character of A\$ isn't A\$[0,0]. The first TIMER isn't TIMER #0. ON X GOTO doesn't start with X=0. Why should the first record be 0???)

Unlike DATA files, the file pointer is never left in the middle of a record, because TEXT files only have one item per record: a single string, which can be of any length.

STORING/RECALLING MORE THAN ONE STRING AT A TIME

This is fun! It's just like SDATA files, and is easy as pie, and very useful. You can store three strings at a time by separating them by commas:

```
PRINT #1;"HELLO", "THERE", "FRED!"
```

This prints three records. You could use variables too, of course.

You can store a string array in one command too. If you have set up an array of 100 strings by

```
DIM M$(100)
```

then you can store the entire set of 100 strings by

```
PRINT #1;M$()
```

(you may omit the parentheses if you like; they're optional). Each of the 100 strings gets put in its own record.

Recalling is the same; you can

```
READ #1,5;A$,B$,C$
```

to read record 5 into A\$, record 6 into B\$, and record 7 into C\$.

To read 100 records into the string array M\$, starting from record 945, just

```
READ #1,945;M$()
```

Again, the parentheses are optional. (Good programmers use 'em to make the program more readable by humans. I never use them myself, because I believe that programs are made to be read by machines, not humans. Those of you who've tried to read my programs have seen the results of this philosophy!)

TEXT FILE TIDBITS

There is a LEX file called "TEXTUTIL" available from the User's Library (#03182-71-0) that makes life with TEXT files much nicer. (This LEX file also exists in modified form in the Text Editor ROM, in the 41 Translator ROM, and in the FORTH/Assembler ROM as "EDLEX". See their respective manuals for details). With this LEX file in place, you can LIST or PLIST a TEXT file directly by name, as if it were a BASIC file! This makes printing of TEXT files a one-command operation without any need to write a program. You can also insert and delete records anywhere in a TEXT file. There is a SEARCH function that quickly returns the location of any specified string in a TEXT file. There are other goodies in there too. Try it!

I mentioned above that you can store numbers in TEXT files. If you

```
PRINT #1;1234,56789
```

it looks like you've stored 1234 in one record and 56789 in the next, but you really stored "1234" and "56789", which is different (takes more memory). Happily, you can

```
READ #1;X,Y
```

if the current record and the next contain numbers or numeric expressions, and the appropriate values will get read into X and Y. So if you've been using DATA files to store lots of text with a few numbers, you might do better to use TEXT files.

Although I mostly store numbers, and therefore love SDATA files dearly, I've used TEXT files to store: CHARSET\$ after designing a good set of special characters; quick notes when writing was out of question (several times when riding in a car, bus, and plane, and once while driving, but that was stupid); FORTH and ASSEMBLY program source code; TRANSFORMed BASIC programs; and lots of ThinkJet graphics data (such as the Space Shuttle picture). But how can one get excited about TEXT files when they don't help solve Ulam's Conjecture?

PUBLIC NOTICE: The Titan File is not and never has been a subset of any book or books. If you noticed a similarity between 📖 [HP-71 BASIC Made Easy](#) and this series on HP-71 files, please don't think that CHHU is using worn snippets from books as articles! In every case, I wrote the CHHU Chronicle articles first, and then revised and percolated them into book form. (See how the articles are proofread, and the book wasn't!) I believe that members of CHHU are far above the average HP handheld user in expertise and brains. My book was designed for the average HP handheld user, NOT the membership of CHHU. If you bought it, hey, great, I need the money. But you paid a lot for your CHHU membership, and I promised Richard and y'all to write a regular column, and that's what's happening. Let *hoi polloi* read my book. Save your money and read The Titan File! I'd be honored. -jkh-

Titan File #7 (CHHU Chronicle, V2 N6, Oct 1985, pages 23-26)

by Joseph K. Horn [13]

Yes, the editor of the Titan File does answer all letters personally. Yes, he appreciates it when he doesn't have to pay for the stamp. No, he doesn't promise that your letter will become immortalized in the CHHU Chronicle. But it might! Here are just a few of the questions received, and the answers given.

Q1. Robert Walker of Racine, Wisconsin, asks how to get the RPT\$ ("repeat string") function, as used in the MAKELEX program in the excellent book [📖 HP-71 BASIC Made Easy](#), page 120.

A1. You have several choices. You can buy the Software Developers' Handbook for the HP-71 from the Users' Library (P/N 00071-90097), which comes with cards, tape or disc containing lots of files, one of which contains RPT\$.

OR you can latch onto somebody else's copy of the Lex File called STRINGLX, not available separately from HP; it is this file that contains RPT\$ and ten other terrific string manipulation functions. [[📖 download its plain-text documentation here](#)]

OR you can use your ASSEMBLER ROM to assemble a new Lex File called RPTLEX, the assembly source listing of which you can find elsewhere in this issue of the Chronicle.

OR you can run the MAKELEX program on the RPTLEX machine code listing, which also appears elsewhere in this issue.

[Note added in 2015: STRINGLX is available online on the [📖 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy STRINGLX directly to your HP-71.]

Q2. J. B. Gero of Manchester-by-the-Sea, Massachusetts, knows how to input an array in BASIC (using a FOR/NEXT loop), but wants to know how to input any user-specified array. For example, suppose I want to input array A, or array B...

A2. If you must do it in BASIC, you have a choice. You can do it the long, easy way: have all the possibilities literally spelled out in the program, and accessed by labels:

```
10 INPUT "Which array? ";A$ @ GOTO A$
20 'A': FOR X=1 TO 10 @ INPUT A(X) @ NEXT X @ END
30 'B': FOR X=1 TO 25 @ INPUT B(X) @ NEXT X @ END
40 'C': FOR X=1 TO 15 @ INPUT C(X) @ NEXT X @ END
50 etc...
```

Or you can do it the short, difficult way, using the MODIFY statement found in the Lex File "SPLEX" that comes with the "Spread-71" program (program #V7524 in the HP Software Distribution Center; \$50.00). The MODIFY statement allows you to add, delete, or change lines of BASIC code in any program (including the currently running program). This allows self-modifying programs:

```
10 INPUT "Which array, last item #? ";a$,b$
20 MODIFY CAT$(0),"30FORX=1TO"&B$&"@INPUT"&A$&"(X)@NEXTX"
30 REM - LOOP WILL GET PLACED HERE AND EXECUTED
40 END
```

If you don't want the whole HP-71 Spreadsheet program, or if the `MODIFY` statement isn't worth \$50 to you, you might persuade a friend who has it to share SPLEX with you. It is not available separately from HP. [[📄 download its plain-text documentation here](#)]

[Note added in 2015: SPLEX is available online on the [📄 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy SPLEX directly to your HP-71.]

But if you don't insist on doing it in BASIC, there is a simple way of doing it with the MATH ROM! You just type from the keyboard `MAT INPUT A` to input the whole A array, or type `MAT INPUT B`, etc. You don't even need a program!

Q3. J. M. Andres of Palos Verdes Estates, California, asks if the ability to have a remote keyboard is unique to the FORTH/ASSEMBLER ROM, or if there is some other way to get the `KEYBOARD IS` statement.

A3. `KEYBOARD IS` and other related keywords are found in the Lex File called "KEYBOARD", contained in the FORTH/ASSEMBLER ROM. Since it isn't "private", it can be copied from a friend's ROM to an external magnetic medium, and thence to your machine, or directly into your HP-71 if you can borrow his ROM! Or you can buy it separately from the Users' Library for \$50 as program #03194.

[Note added in 2015: KEYBOARD is available online on the [📄 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy KEYBOARD directly to your HP-71. A different version of the lexfile on the same disk is called KEYBRD. Try both.]

Q4. Brian Dollar of Columbus, Ohio, asks how to turn on BOLD MODE on the ThinkJet.

A4. Just `PRINT CHR$(14)` where you wish bold printing to begin, and `PRINT CHR$(15)` where you wish to go back to light (normal) printing. You can do this really easily if you remember that the CTRL key allows you to type most of the first 32 ASCII characters; just hit [CTRL] [N] to get CHR\$(14), which displays as a Greek "tau", and hit [CTRL] [O] (that's the letter "O", not the digit "0") for CHR\$(15), which displays as a Greek "phi". But they aren't printed at all; they are interpreted by the ThinkJet as control characters (hence the name of the CTRL key). Unfortunately, the `PRINT` command does count them as printed characters when calculating the length of the line and comparing it with your `PWIDTH` setting. To avoid this problem, use `OUTPUT` instead of `PRINT` whenever you wish to send characters that the ThinkJet interprets as control codes.

Q5. David M. Hahn of Lansing, Michigan, asks how to emulate the HP-41 "RND" (round) function.

A5. The RND function, also found on the HP-67 and other HP handhelds, changes the value of X to actually be what it looks like in the current display mode (FIX, SCI, ENG, 2 digits, 4 digits, whatever). This is easily emulated: `X=VAL(STR$(X))`.

Q6. Ty Waltemyer of York, Pennsylvania, asks if it is possible to type more than 96 characters into an `INPUT` or `LINPUT`.

A6. No. (But see the next question!)

Q7. Nobody, anywhere, asked how to type more than 96 characters into a string.

A7. It's possible, but it's sloppy. You can dimension your target string to the maximum reasonable length, and then "strobe" it in a chunk at a time:

```
DIM A$[1000],B$[96]
LINPUT A$
LINPUT B$
A$=A$&B$
LINPUT B$
A$=A$&B$
etc...
```

To do it automatically, just put this idea in a BASIC routine like the following. You just type a chunk, hit ENDLIN, type another chunk, and so on; to exit, press ENDLIN with no input:


```
10 DIM A$[1000],B$[96]
20 LINPUT B$
30 IF LEN(B$) THEN A$=A$&B$ @ GOTO 20 ELSE END
```


Trouble is, you might not know what the maximum reasonable length is. Memory might be unavailable for huge "maximum reasonable length" strings. Here's the way I do it:

```
10 DIM M$,N$,A$[96] @ M$=""
20 LINPUT A$ @ IF NOT LEN(A$) THEN END
30 DIM N$[LEN(A$)+LEN(M$)] @ N$=M$&A$
40 DIM M$[LEN(N$)] @ M$=N$ @ GOTO 20
```

This keeps redimensioning the target string (M\$) using a scratch string (N\$). This allows entering strings of length proportional to the amount of available memory.

Q8. David Mabijs of Bryan, Texas, and Emmett Payne of Altadena, California, ask how to "UNPRIVATE" a PRIVATE file.

A8. The fact that a file is PRIVATE is stored in its header, and the only way to "UNPRIVATE" a file is to change that part of the header that says it's PRIVATE. If the file is in ROM, where changes are obviously impossible, then you can't "UNPRIVATE" it. But you can copy it into RAM (if you have enough memory) and it is easy to unPRIVATE files in RAM. It can be done from FORTH in a single statement. It can be done from BASIC if you have the UNPRIV Lex File. It can even be done from plain vanilla BASIC! How? Send a self-addressed, stamped envelope and \$1, and you'll find out. [ [download that plain-text document here](#)] Enclose a magnetic medium *if* you want the UNPRIV Lex File and BASIC routine recorded on card, tape or disc.

[Note added in 2015: UNPRIV is available online on the  [SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy UNPRIV directly to your HP-71.]

Q9. James Coffey of Fort Worth, Texas, asks for the MAKELEX and source code listings of the Lex File that adds the MSA(x) function.

A9. MSA (Modified Syracuse Algorithm) was featured in the [Titan File #5 about Ulam's Conjecture \(V2 N3 P37d\)](#). Modifying the ULAM(x) function into the MSA(x) function was left as an exercise for the reader. Compare your results with mine, which appear [elsewhere in this issue](#).

[Note added in 2015: I called my MSA lexfile SYRACUSE. It is available online on the [📁SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy SYRACUSE directly to your HP-71. [📁Here is its plain-text documentation.](#)]

Q10. Paul D. Miller of Convent Station, New Jersey, asks what is the best way to get started in Assembly Language programming on the HP-71.

A10. Buy the FORTH/ASSEMBLER ROM, buy Volumes I and II of the IDS (Volume III is not necessary), join CHHU, read John Baker's excellent column "Exploring the 71 IDS", and do nothing else for a few months. That's what I've done, and after about a year at it, I'm finally beginning to understand it!

Q11. Ralph C. Koeller of Platteville, Wisconsin, asks how to add the "Heavyside step function" to the HP-71. It is defined as $H(t) = (1 + \text{SGN}(t)) / 2$.

A11. Easy! If you want it from the keyboard or CALC mode, just press EDIT, then key into the workfile:

```
10 DEF FNH(T)=(1+SGN(T))/2
```

(You may use any line number, but it must be before any SUBs in the workfile). Then you merely type FNH(x) to get H(x).

If you want it in a program, just include the above definition in your program. You may then use FNH(x) in that program. And, as long as that program is the current file (the one displayed when you press CAT), then FNH is also available from the keyboard and CALC mode.

Notice that many functions that we want Lex Files written for are instantly creatable as user defined functions, like FNH above. No need for lex wizardry!

Q12. Richard de Queiroz of Palos Verdes Estates, California, asks how to review and optionally edit variables like this routine does on the HP-41:

```
LBL A, RCL 00, "X=", ARCL 00, PROMPT, STO 00, RTN
LBL B, RCL 01, "X=", ARCL 01, PROMPT, STO 01, RTN
LBL C, RCL 02, "X=", ARCL 02, PROMPT, STO 02, RTN
etc...
```

A12. It's built in! The INPUT statement has a "default" feature. For example, try this: INPUT "X=", STR\$(X);X. Notice that the old value of X is displayed, and pressing ENDLINE or CONT leaves it the same, but you can also type a new value. This is one time that BASIC is a lot easier than the HP-41!

Q13. John Weale of Stamford, Connecticut asks how to get a catalog of just the SDATA files in memory. His program's data is stored in several SDATA files, and he wants the program to do a CAT ALL but only show the SDATA files.

A13. Use CAT\$:

```

10 DELAY 9,9 @ FOR X=1 TO INF @ IF NOT LEN(CAT$(X)) THEN 50
20 IF CAT$(X)[12,16]#"SDATA" THEN 40
30 DISP CAT$(X)
40 NEXT X
50 DELAY 0,0 @ END

```

Unlike CAT ALL, this only lists the SDATA files, and only lets you move down through the catalog, not up.

Q14. Justin Gray of Dana Point, California, asks some good questions about FORTH. (A) How can you list FORTH dictionary words? (B) How can you selectively delete FORTH words? (C) How can you avoid the “dictionary full” error? (D) How can you purge FORTH and keep it from magically reappearing?

A14. (A) You cannot list FORTH words, per se. They do not exist in the machine as lists of instructions, but only as a bunch of numbers (similar to an indecipherable sequence of GOSUB's in BASIC). If you get the Software Developers' Handbook from HP (see Answer #1 above), it comes with an amazing FORTH UTILITIES file that adds, among other wondrous things, the “UN:” FORTH word which “decompiles” any specified FORTH word into more or less its original listing. A similar decompiler can also be found in a new HP-71 package by William Wickes (author of the Translator ROM) called “HP-71 Translator Pac Programmer's Tool Kit” available for \$60 from the Users' Library as program #V7540.

(B) You cannot “selectively” delete FORTH words. When you use FORGET to delete a word, it also deletes all words defined after the one being deleted, because of the way FORTH words are “chained” (threaded?) together. FORGET is therefore very much like PCLPS in the HP-41; it not only clears the named program, but wipes out all programs from there up to the most recent one in memory!

(C) If you have the FORTH/ASSEMBLER ROM, then 2000 GROW *expands* the dictionary by 1000 bytes. If you have the 41 Translator ROM, in FORTH mode you can use GROW as above, or use 2000 DSIZE to make *exactly* 1000 bytes available in the dictionary; in HP41 mode, use 2000 XSIZE to do the same. If you need more dictionary space than that, use a number bigger than 2000.

(D) To purge FORTH, first get into BASIC. If you have the FORTH/ASSEMBLER ROM, just PURGE FORTHRAM; if you have the 41 Translator ROM, then PURGE FTH41RAM. BUT!!! The very moment that you re-enter FORTH, or HP41 mode, or use FORTHX, then a new Forth Ram file will be created. To avoid that, don't execute FORTH, HP41, or FORTHX. Safer still, yank the module!

Q15. Carl Zander of Eureka, California, asks how to put the 82162A HP-IL Thermal Printer/Plotter into double-wide mode, graphics mode, etc.

A15. All it takes is an output of the proper “escape code”. Character number 27, obtained by CHR\$(27), is called the “escape” character. Most printers, including the HP-IL Thermal printer, interpret the escape character as, “Look out, here comes a special command,” and interpret the very next characters received as an encoded command. For example, the printer's manual says that $\text{e}_c\&k15$ is the escape code for double-wide mode. That means that all you have to do is PRINT CHR\$(27)&"&k15"; to turn on double-wide mode. Notice that “ e_c ” means CHR\$(27), and the “&” in the escape code is not the same as the “&” used to concatenate CHR\$(27) with the rest of the code. If your PWIDTH is not INF, you should use OUTPUT instead of PRINT to avoid early

carriage returns.

Graphics mode's escape code is $\text{e}_c * \text{bnnnGx}_1 \text{x}_2 \dots \text{x}_{\text{nnn}}$ where nnn is the number of dot-column characters at the end. So to graphically print a "degrees" symbol, composed of dot-column characters 6, 9, 9, 6, you just `PRINT CHR$(27)&"*b4G"&CHR$(6)&CHR$(9)&CHR$(9)&CHR$(6)`. Or use the CTRL key (see answer #4 above) with CTRL F for CHR\$(6) and CTRL I for CHR\$(9).

Q16. Douglas Osheroff of Watchung, New Jersey, asks if the HP-71 BEEP statement can be used to dial a phone.

A16. No. The Touch Tone® tones are actually a combination of two tones simultaneously, one tone representing the pressed key's row, the other tone representing the column. Since the BEEP statement on the HP-71 only allows a single frequency (not chords), Touch Tones are not possible. But creating a lexfile function "TONE" ought to be possible which, say, is given a string argument like `TONE "17146332041"` and then dials those numbers. It could include special characters like # and *, and something else for a pause, making long-distance services easier to access. I'm dreaming. Such a lexfile does not exist. Yet.

Q17. Fred Buhler of Yonkers, New York, asks how to get the STARTUP string out once it's been put in.

A17. There is no simple way in BASIC. If you have the STARTUP Lex File, then you have the `STARTUP$` function which returns exactly what you want. Without that Lex File, you can use `FORTHX` from BASIC:

```
FORTHX "HEX 808 FINDBF DUP 3 - C@ 2/ 1- TYPE CR"
```

Q18. John Millard of Perth Ontario, Canada, asks how to get a paper trace of CALC mode's operations, like the HP-41's TRACE mode.

A18. You can't! The best I can suggest is to do your calculations in BASIC mode with `DISPLAY IS PRINTER` set.

Q19. André Côté of Ste-Foy Quebec, Canada, asks how to insert special CHARSET characters into ThinkJet printed text, such as "é" in his own name. He says creating the CHARSET character was easy, and assigning it to a key is easy, but it doesn't print the way it looks.

A19. You cannot directly print CHARSET characters on the ThinkJet; its character set is unchangeably burned into ROM. But *the characters you want are in that set!* Check out the "Roman 8 Extension" set that your ThinkJet uses for `CHR$(161)` through `CHR$(255)`. Merely assign the desired `CHR$` to a key, and press that key wherever you wish the ThinkJet to print that character. Of course, it'll *display* funny, but it's the internal ASCII code that matters, not the way it shows up in the LCD.

The Users' Library sells a program (#03187, \$12 w/ cards) designed specifically for this purpose; it adds a new keyword to BASIC ("ROMAN8"), which in one stroke defines the entire HP-71's Alternate Character Set to be the same as the ThinkJet's Roman 8 Extended Character Set.

[Note added in 2015: That library is called THINKJET. It is available online on the

[📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy THINKJET directly to your HP-71.]

Q20. Too many folks from as many places have written bemoaning the way PWIDTH and ENDLINE foul things up. They make DOT2DOT fail. They create unwanted carriage returns. They cause loss of data when things “fall off the edge” of the page unprinted. They wreak havoc when you want a carriage return and get something else...

A20. Set them yourself! The normal ENDLINE setting is obtained by merely typing ENDLINE. Do *not* type ENDLINE "", because then the PRINT command will *never* perform a carriage return / line feed, and your output will get printed all glommed together. (I just looked up that word, and it doesn't exist, so don't write me to say it's spelled wrong!)

The default PWIDTH setting is 96, which is stupid, unless you use PRINT when you should be using DISP. You should either set it to your printer's width (24 for the little thermal printer, or 80 for the ThinkJet or impact printers), or to infinity: PWIDTH 24, PWIDTH 80, and PWIDTH INF. I always keep mine at PWIDTH INF to avoid problems, and the default ENDLINE setting. Although I like surprises in life, I don't like them coming from my printer.

Q21. There is a blind mouse on one corner of a wire-frame n-dimensional hypercube (2=square, 3=regular cube, 4=tesseract, etc.), and a cheese in the diametrically opposite corner. He walks randomly from corner to corner (without ever turning 180° to retrace his steps) until he finds the cheese. How would one program the HP-71 to generate such a random path, labeling every corner and every wire travelled, from beginning of the walk until the cheese? The user begins the program by inputting the desired dimension of the hypercube up to, say, 26.

A21. Next month!

[Note added in 2015: SYRACUSE is available online on the [📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy SYRACUSE directly to your HP-71.]

Saturn Source Code for SYRACUSE LEX 'SYRACUSE' ID #99 MSG 0 POLL 0 POP1N EQU #0BD1C RJUST EQU #12AE2 DCHXW EQU #0ECDC HXDCW EQU #0ECB4 FLOAT EQU #1B322 FNRTN4 EQU #0F238 ENTRY MSA CHAR #F	KEY 'MSA' TOKEN 2 ENDTXT NIBHEX 811 MSA GOSBUL =POP1N GOSBVL =RJUST C=A W GOSBVL =DCHXW SB=0 B=0 W D=C W LOOP B=B+1 W ASRB ?SB=0 GOYES TEST	C=C+A W C=C+1 W A=C W SB=0 GOTO LOOP TEST C=A W ?C>D W GOYES LOOP C=B W GOSBVL =HXDCW A=C W GOSBVL =FLOAT C=A W GOVLNG =FNRTN4 END
---	---	--

SYRACUSE ID#99 76 bytes

```
-----  
      0123 4567 89AB CDEF ck  
000: 3595 2514 3455 3554 B9  
001: 802E 0091 3210 1000 70  
002: 8900 0992 0200 0000 33  
003: F710 0000 0000 0000 A7  
004: 0510 00F5 D435 1420 EB  
005: 1FF8 118F C1DB 08F2 45  
006: EA21 AF68 FCDC E082 F1  
007: 2AF1 AF7B 7581 C832 0D  
008: 21A7 2B76 AFA8 2268 4A  
009: EFAF 69F7 FDAF 98F4 32  
00A: BCE0 AFA8 F223 B1AF F2  
00B: 68D8 32F0          E7
```

[Note added in 2015: RPTLEX is available online on the [📄 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy RPTLEX directly to your HP-71.]

RPTLEX ID#52 145 bytes

```
-----  
      0123 4567 89AB CDEF ck  
000: 2505 45C4 5485 0202 9C  
001: 802E 0054 6150 1000 A9  
002: 2210 025B 0B00 0000 FC  
003: F710 0000 0000 0000 A7  
004: 0810 00F7 2505 4542 BB  
005: B01F F842 28FD 8DB0 89  
006: 4F01 7F8F 322B 14D0 D1  
007: 31B0 8D39 3901 01AF A6  
008: 08F8 3DB0 8AC9 01CF B1  
009: 6AA0 1021 3713 5C21 FF  
00A: 0810 BD71 198F 943B 5B  
00B: 1DBE 2423 1321 BBB8 38  
00C: F214 0112 D811 0131 30  
00D: 1B49 5F21 4213 016F B7  
00E: 1328 BA90 8DD4 4901 8A  
00F: 11CC 1014 2111 B133 BC  
010: 8FEE 0B15 7E11 3137 DC  
011: 135E AD81 BBB8 F214 FC  
012: 2130 AF0D 4BF0 BF02 0E  
013: 0A0C 1CF1 5178 DC32 46  
014: F0          A6
```

Saturn Source Code
for RPTLEX

```

.....
LEX      'RPTLEX'
ID       #52
MSG      0
POLL     0
ENTRY   RPT
CHAR     #F
KEY      'RPT$'
TOKEN    11
ENDTXT
NIBHEX  8422
RPT GOSBVL #0BD8D
GOC      R1
D1=D1+  16
GOSBVL  #1B223
GOC      R2
R1 LCHEX  0B
GOVLNG  #09393
R2 R1=A
A=0      W
GOSBVL  #0BD38
?A#0    A
GOYES   R3
D1=D1-  16
GOTO    E4
R3 R2=A
CD1EX
D1=C
C=C+A   A
R0=C
R3=C
D=C     A
C=R1
GOSBVL  #1B349
C=D     A
C=C-A   A
GOC     R5
AD0EX
D0=(5) #2F8BB
DAT0=A  A
A=R2
B=A     A
A=R0
D1=A
D0=(5) #2F594
A=DAT0  A
D0=A
D0=D0+  16
AD0EX
?C>=A  A
GOYES   R6
R5 GOVLNG #0944D

```

```

R6 A=R1
A=A-1 A
R1=A
GOC   R7
C=R3
AD1EX
GOSBVL #1B0EE
GONC   R6
R7 A=R3
CD1EX
D1=C
A=A-C  A
B=A    A
D0=(5) #2F8BB
A=DAT0 A
D0=A
A=0    W
A=B    A
ASL    W
ASL    W
P=     0
A=A-1  P
D1=D1- 16
DAT1=A  W
E4 GOVLNG #0F23C
END

```

Titan File #8 (CHHU Chronicle, V3 N1, Jan/Feb 1986, pages 23-26)

by Joseph K. Horn [13]

Contents:

1. [Buckaroo Bonzai the Mouse](#)
2. [Using USING](#)
3. [HP-71 Stereo Music](#)

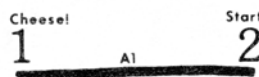
1. BUCKAROO BONZAI THE MOUSE

Finds Cheese on an 8-Dimensional Hypercube!

Buckaroo Bonzai, as everybody knows, is a mathematical mouse. The only thing he loves more than a fresh math puzzler is fresh cheese! But, as everybody also knows, the mice of song and legend are always blind, so our poor Buckaroo must rely on his sense of smell, which is unfortunately not very directional. If he is in the vicinity of cheese, he can smell its presence, but he cannot tell where it is! So he is condemned to wander along a random path until he finally finds the cheese.

One fine day, Buckaroo The Mouse finds himself at one end of a line called "A1". (See illustrious illustration). He smells cheese nearby, which happens to be at the beginning of the line, logically labeled "1". Since he is at the end of the line, his position is labeled "2". His challenge is to figure out how to leave point "2", traverse line "A1", and arrive at point "1" to get the cheese. Fortunately, this is not difficult, as there are no choices to be made!

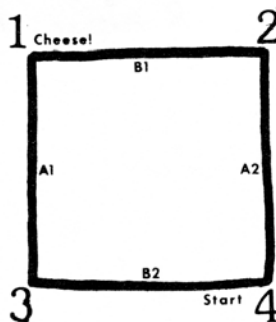
1 DIMENSION (LINE)



This is an example of random traversing in 1-dimensional "space". If you run the "BUCKAROO" program (listed elsewhere) with an input for 1 dimension, you'll see the mouse's path listed in the display as "Start-2 A1-1 Cheese!" which means "Start at point 2, travel along path A1 to point 1, and find the cheese!"

The next day, Buckaroo finds himself at one corner of a square, at point "4". At the opposite corner, labeled point "1", is a cheese. (See illustration). This time, Buckaroo has a choice: he can travel along path A2 to point 2, and then along path B1 to the cheese, or he can follow path B2 to point 3, and thence along path A1 to the cheese.

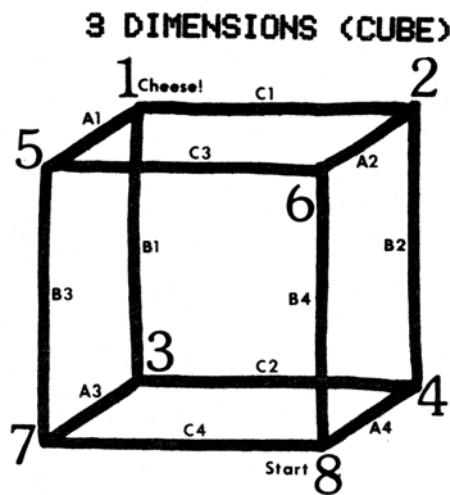
2 DIMENSIONS (SQUARE)



This is an example of random traversing in 2-dimensional space. Running the BUCKAROO program with an input for 2 dimensions results in either of two displays: "Start-4 A2-2 B1-1 Cheese!" or "Start-4 B2-3 A1-1 Cheese!", the two possible ways Buckaroo can get to the cheese.

Guess what happened the next day! Sure enough, Buckaroo finds himself on one vertex of a cube! And, of course, he smells a cheese (which he is too dumb to induce is on the opposite vertex of the cube). Our heroic mouse begins his random walk (see illustration). There are now *six* (3 factorial) possible ways to get to the cheese in three moves each. But that's not all! Since he's blind, and moves randomly, he can miss the cheese altogether and just wander about aimlessly on the rest of the cube! He never turns 180° to retrace his steps. He's not that dumb! So finding the cheese was inevitable in 1 and 2 dimensions. But now it's complicated enough for Buckaroo to get lost!

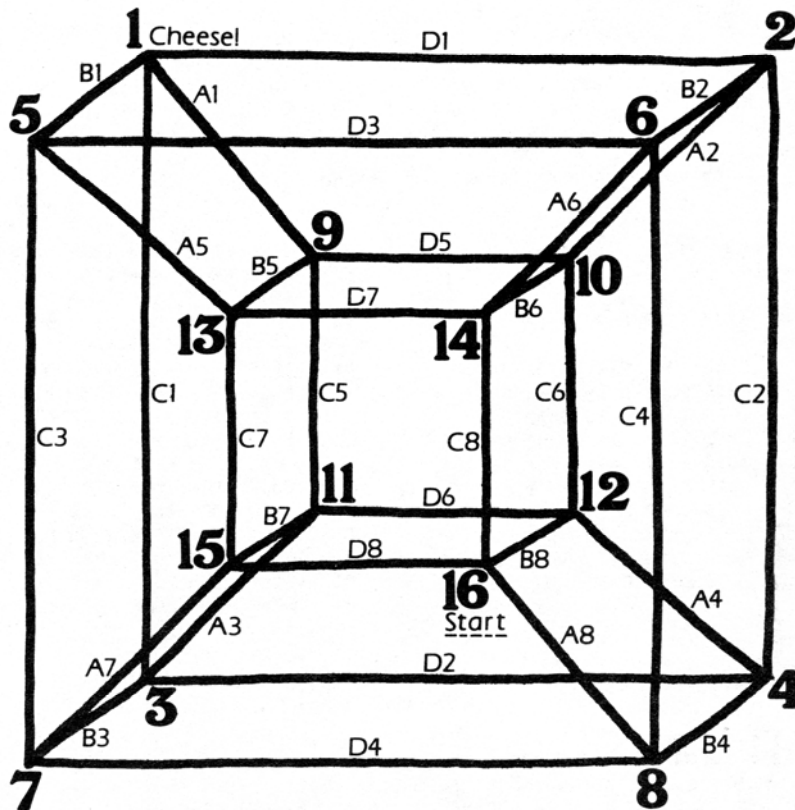
Run BUCKAROO with an input for 3 dimensions. As before, his entire itinerary will be displayed: every point, and every path between them. Follow along on the illustration.



But probability says that given enough time, a blind mouse will find his way from point 8 to point 1 on a cube. So Buckaroo does find the cheese, and the day ends happily.

You know what happens the next day! A 4-dimensional cube is called a "tesseract". The Mouse finds himself on corner 16 and smells the cheese on corner 1. (The illustration is perforce unsatisfactory: it is a two-dimensional drawing of a four-dimensional object. Try drawing a cube on a line!) But he doesn't know where the cheese is, and begins his random walk from point to point along the tesseract's edges. The shortest way to the cheese is to take four paths, and there are 24 (4 factorial) different ways of doing that! Not to mention that there are now lots of ways of getting lost and meandering through the tesseract for a long time before finding the cheese! But find it he will, and spends a happy evening enjoying his meal.

4 DIMENSIONS (TESSERACT)



The next day? Ah, I'll let you draw it. But the program handles a 5-dimensional hypercube just fine! Matter of fact, you can use up to 26 dimensions without any problem, but don't expect to live to see Buckaroo find the cheese!

The subtitle problem, finding cheese on an 8 dimensional hypercube, is realized by running the BUCKAROO program with an input for 8 dimensions. If you want to be sure that the mouse does find the cheese within your lifetime, you can seed the random number generator with the right value. For example, if you RANDOMIZE PI before pressing the RUN key, Buckaroo will get to point 1 in only 16 moves! Can you find a RANDOMIZE seed that results in a shorter run?

IMPORTANT NOTE: The BUCKAROO program uses an external function called BVAL (Base VALue). If you have the MATH ROM, then you can key in and run BUCKAROO right away. If you don't have the MATH ROM, then you must add the BVAL function to your HP-71. You can do this easily enough by running the Fange/Tobiasson MAKELEX program (cf. CHHU Chronicle V2 N1 P20; or [HP-71 BASIC Made Easy](#), p. 120) on the BVALLEX listing below. Or, if you have the FORTH/ASSEMBLER ROM, you can type the source code listing of BVALLEX (which appears below) into a TEXT file and then assemble it. The source code was obtained with a modified version of Thompson's "DIS" disassembler (V2 N1 P6). Sorry, but the source code is uncommented because the disassembler is smart enough to extract BVAL from the MATH ROM but not smart enough to add comments!

[Note added in 2015: BVALLEX is available online on the [SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy BVALLEX directly to your HP-71.]

The BVAL function's syntax, in case you'd like to play with it, is: BVAL(string, base number). The string is interpreted to be a string of digits in the specified base, and BVAL returns its value in base 10. The only legal bases are 2, 8, and 16. The string must evaluate to no more than 999999999999 (1E12-1). EXAMPLES: BVAL("10011001",2) returns 153, because 10011001 in base 2 is equal to 153 in base 10. BVAL("FEDCO",16) gives 1043904 because FEDCO in hex is 1043904 in decimal.

BUCKAR00 BASIC 277 bytes

```

10 STD @ DIM D,X,N,T @ INPUT "Dims? ";D @ IF D<1 OR D>26 THEN 10
20 DIM A$(D) @ A$="" @ FOR X=1 TO D @ A$=A$&"1" @ NEXT X
30 PRINT "Start-"&STR$(2^D)&" ";
40 N=CEIL(RND*D) @ IF X=N THEN 40 ELSE X=N @ T=T+1
50 A$(X,X)=STR$(NOT VAL(A$(X,X)))
60 N=BVAL(A$[1,X-1]&A$[X+1],2)+1
70 PRINT CHR$(64+X)&STR$(N)&"-STR$(BVAL(A$,2)&" ";
80 IF N>1 THEN 40
90 BEEP @ PRINT "Cheese! ("&STR$(T)&" moves)"

```

[Note added in 2015: BVALLEX is available online on the [📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy BVALLEX directly to your HP-71.]

BVALLEX ID#02 191 bytes

```

      0123 4567 89AB CDEF ck
000: 2465 14C4 C454 8502 7A
001: 802E 0005 9120 1000 88
002: D710 0205 0500 0000 0B
003: F710 0000 0000 0000 A7
004: 0810 00F7 2465 14C4 BF
005: 501F F842 278E 0AF0 8E
006: 8F83 DB0A F181 C100 69
007: D3CF 6A90 AF02 08F7 A5
008: A251 1713 3031 3879 CE
009: 3133 0373 87A8 0330 F5
00A: 3938 FC70 B15E 187B B4
00B: 606D A033 1464 8FC7 36
00C: 0B14 EEA6 C310 3B6A A6
00D: 2B96 8A3D B562 05A1 9F
00E: 440D 8799 1A14 45CA 4A
00F: 144F B87A 80A1 444B 07
010: 04CE 57D0 5A18 45A0 9F
011: 4118 E78A 3606 C5F2 42
012: 0D23 141D A058 F684 D8
013: C0AC 2AD9 AB61 CF8D D4
014: 832F 08FC 1DB0 5908 24
015: D51D B08F 322B 1490 1C
016: 8D91 FB01 7F84 984A 85
017: 84BD 2302 8A67 0859 15
018: 0130 88A6 7085 A013 79
019: 1018 A6AC 85B0 1 09

```

BVALLEX Source Code									
	LEX	'BVALLEX'		GOYES	L2	GOYES	L7	GOVLNG	#0BD15
	ID	2		LCHEX	3930	A=A+A	WP	L11	GOSBVL #1B223
	MSG	0	L2	GOSBVL	#1B07C	GOC	L3		GOC L13
	POLL	0		GONC	L5	L7	SETHex	L12	GOVLNG #0BF19
	ENTRY	BVAL		?ST=1	11	C=C-1	A	L13	D1=D1+ 16
	CHAR	#F		GOYES	L4	GONC	L6		ST=0 9
	KEY	'BVAL'	L3	GOTO	L12	SETDEC			ST=0 10
	TOKEN	5	L4	LCHEX	4641	B=A+B	WP		ST=0 11
	ENDTXT			GOSBVL	#1B07C	GOC	L3		C=0 A
BVAL	NIBHEX	8422		GOC	L3	L8	SETHex		LCHEX 2
	GOSUB	L10		A=A-1	B	C=R0			?A#C A
	A=0	W	L5	LCHEX	30	D=D+1	A		GOYES L14
	GOSBVL	#0BD38		A=A-C	B	?D=C	A		ST=1 9
	B=0	W		P=	11	GOYES	L9		RTN
	ASRB			?A=0	B	GOTO	L1	L14	LCHEX 8
	R0=A			GOYES	L8	P=	0		?A#C A
	D=0	A		C=D	A	C=0	A		GOYES L15
	D=D-1	A		GONC	L7	LCHEX	14		ST=1 10
	GOTO	L8	L6	SETDEC		A=C	A		RTN
L1	A=0	W		A=A+A	WP	SETDEC		L15	LCHEX 10
	P=	0		GOC	L3	GOSBVL	#0C486		?A#C A
	GOSBVL	#152A7		?ST=1	9	C=0	S		GOYES L12
	D1=D1+	2		GOYES	L7	C=B	M		ST=1 11
	LCHEX	3130		A=A+A	WP	C=A	X		RTN
	?ST=1	9		GOC	L3	D1=D1-	16		END
	GOYES	L2		A=A+A	WP	GOVLNG	#0F238		
	LCHEX	3730		GOC	L3	L10	GOSBVL #0BD1C		
	?ST=1	10		?ST=1	10	GONC	L11		

2. Using USING

You want your outputs in some special, customized format, but don't know how to do it? Here goes!

No chatter, just lots of examples! TRY THEM ALL. You will find USING to be one of the most powerful and friendly keywords your HP-71 has! (Besides the fact that you paid dearly for it; a huge chunk of the operating system is the USING code!)

Rather than typing DISP USING, just type USING. The HP-71 will interpret this as an "implied DISP". Saves time. If the output is to be to the printer, use PRINT USING. If to some other HP-IL device, or if you want to bypass PWIDTH character counting, then use OUTPUT USING.

NOTE WELL: The HP-71 Owner's Handbook and Reference Guide are not too useful regarding USING, but you don't have to memorize all the USING symbols. Use the Quick Reference Guide (the pocket guide that comes with the HP-71). Its table of USING symbols on pages 22 and 23 is very useful!

<u>USING format string; parameter list</u>	<u>output result</u>
USING "3dc3dc3d.4d";12345678.22229 -----> 3 digits, comma, 3 digits, comma, 3 digits, point, 4 digits (USA)	12,345,678.2223
USING "3dp3dp3dr4d",12345678.22229 -----> 3 digits, period, 3 digits, period, 3 digits, radix, 4 digits (Europe)	12.345.678,2223

<u>USING format string; parameter list</u>	<u>output result</u>
USING "4d";1,2,153,12,8	1 2 153 12 8
USING "4d";134,18,997,11,1.7	134 18 997 11 2
USING "4d";-7,0,11/7,INF,NAN	-7 0 2 Inf Nan
-----> Notice how "4d" repeats	
USING "#, '\$'3dc3d.2d/";1234.8,1e6-1,1.95	\$1,234.80 \$999,999.00 \$1.95
-----> The "#" suppresses carriage return, the "/" forces a carriage return. "\$" floats in front of digits.	
USING "23'-'"	-----
USING "8*";153	*****153
USING "8z";153	00000153
USING "3zx";1,22,333,4444	001 022 333 ***
USING "'\$'3*c3*.2dx'exactly'";5995.95	**\$5,995.95 exactly
USING "23'-'"	-----
USING "' ('3d') 'x3d'-'4d";7146332041	(714) 633-2041
USING "3d'-'2d'-'4d";344382447	344-38-2447
USING "'\$'xdxdxdxdx.xdxdx'!";1537.95	\$ 1 5 3 7 . 9 5 !
USING "' ('a') '5a";"Copy","List","Purge"	(C)opy (L)ist (P)urge
USING "m3d";1,-2,3,-4	1 -2 3 -4
USING "s3d";1,-2,3,-4	+1 -2 +3 -4
USING "d.s2d";pi	3.+14
USING "d.2ds";pi	3.14+
USING "2(4dr)4d.";1,22,333	1, 22, 333.
USING "kx,^";1,2,3,4,5,6,7	1 3 5 7

There are, of course, an infinite number of possibilities. Just about any imaginable output format can be obtained with USING and the right format string. (Yes, Harry, there are a few fancy things that BASIC can do in one statement!)

3. HP-71 STEREO MUSIC!

At a recent meeting of the Orange County Chapter of CHHU, we got both voices of Bach's Musette from the Anna Magdalena Notebook running on *five* HP-71's simultaneously... and they were all perfectly in tune and time! The effect was amazing. It was better than stereo... it was quintaphonic!

Fancy HP-71 music is made possible by a lexfile called NOISE written by Steve Williams. Like the HP-75's I/O ROM, the NOISE lexfile adds a new keyword to BASIC, called NOISE. This statement takes encoded notes and durations and drives the built-in sound generator (HP-75's speaker, or HP-71's beeper or "bender") to play music. Since you can encode practically any set of frequencies and durations, such as video game laser blasts or Pac-Man death throes, the name "NOISE" is appropriate!

Since NOISE requires encoded information, a second file called MUSIC takes normal music notation (sorta) and performs the encoding for you. So you can take a sheet of music and type it into the HP-71, press a button, and hear it play! You can also save your composition into a text file, so that you don't have to type it in ever again, and MUSIC doesn't have to re-encode it either.

But every HP-71 has a slightly different clock speed. Since people with perfect pitch are offended by sour notes, and two HP-71's making MUSIC together would sound terrible, Jim Horn wrote a program called TUNEUP. It figures out what your clock speed is, and "tunes" the NOISE lex file for your HP-71. Once run, you won't need TUNEUP again unless your clockspeed changes (e.g. you switch from near-dead batteries to AC) or you lose your personalized NOISE lex file.

To run MUSIC:

(1) Make sure NOISE is in memory.

(2) RUN TUNEUP if necessary to fine-tune NOISE (it takes 12 seconds to finish).

(3) RUN MUSIC

(4) To play a piece of music stored in a text file:

Input an EQUAL SIGN (=) followed by the file name, e.g. =WTELL or =MUSSETTE. If you wish to play a file that's on a tape or disc, include the device specifier in the name, e.g. =WTELL:TAPE or =MUSSETTE.TUNES

(5) To compose a new piece of music, input the notes, rests, and timings according to the following rules (6 through 9 below):

(6) Default time is $\frac{1}{4}$ second per note & rest.

Default octave is from middle C up to middle B.

(7) Type natural notes in uppercase (C, D, E, F, G, A, B).

Type sharps in lowercase (c, d, f, g, a).

There are no flats. Use equivalent sharps. (Bb = A# = a)

Rests are input as an R.

(8) Change timing by typing it in parentheses. E.g. if you type (1/8) then the following notes & rests will be one eighth of a second long. Any number is allowed.

(9) To go to the next higher octave, type a +.

To go to the next lower octave, type a -.

Maximum range is 3 octaves above & below middle D.

(10) Type any combination of the above notations into the input line, then press ENDLINE. Then continue typing input, pressing ENDLINE after each line. Use the command stack to make repetitive entries easy.

(11) Press ENDLINE without input to listen to your composition.

You may have to wait a while for translation to finish.

(12) If you wish to save your composition in a file, input a file name. You may include a device specifier if you wish to save it on tape or disc. You may then play it as described above (4).

(13) If the music translator finds an input error, it will tell you what error was found, and where.

(14) For stereo, use two HP-71's!

EXAMPLES:

Input `EDCDEEERDDDREGGREDCEEEEDDED` to play "Mary Had a Little Lamb" with default tempo and octave.

Input `=WTELL` (if you have the `WTELL` file in memory) to hear about 2 minutes of the William Tell Overture (the "Lone Ranger" theme).

Input `ABCDEFGH` and see "`H @ 8 ???`" displayed, to indicate that `H` is not a proper input, and was found at position 8 in the input.

Input the following for the Happy Birthday song:

```
(2/8)C(1/8)C(3/8)DCF(5/8)E(1/8)R
(2/8)C(1/8)C(3/8)DCG(5/8)F(1/8)R
(2/8)C(1/8)C(3/8)+C-AFED
(2/8)a(1/8)a(3/8)AFG(5/8)F(1/8)R
```

Notice the similarities between lines 1, 2, and 4; wise use of the command stack makes music entry much easier. Try inputting the whole thing twice in a row, using the command stack. Try it again, preceding the first line with a `+`. The command stack makes a great music editor!

Notes for the technically-minded: `NOISE` is a BASIC statement that uses a single string parameter: `NOISE A$`. The string is made up of pairs of bytes, the first byte of each pair being an encoded frequency, and the second byte of each pair being an encoded duration:

```
NOISE FREQUENCY$ & DURATION$ & FREQUENCY$ & DURATION$ & . . .
```

The "encoding" is simple. `FREQUENCY$ = CHR$(16*OCTAVE+NOTE)` where `OCTAVE=8` for the middle octave, and can go up to 11 and down to 5; and `NOTE=0` for A, 1 for A#, 2 for B, and so on up to 11 for G# (although it accepts up to 15 for C in the next octave). `DURATION$ = CHR$((LOG(SECONDS)/LOG(2)+12)*16)`.

Thus `NOISE CHR$(16*8+3) & CHR$((LOG(.5)/LOG(2)+12)*16)`
which simplifies to `NOISE CHR$(131)&CHR$(176)`

plays a middle C for half a second. The maximum frequency number is 181; anything above that is forced silent by `NOISE`. I used a frequency of 255 for rests in the `MUSIC` program. (No, it's NOT ultrasonic; it's silent!)

`NOISE` does not use the operating system's routines for driving the beeper, but drives it directly with better algorithms for making music. It is therefore better than `BEEP` for hitting the notes square on, and for exact tempo, essential when playing an HP-71 duet or trio. The current version does not take the clock speed into account, however, which makes a program such as Jim Horn's `TUNEUP` necessary. Steve Williams said he might revise his `NOISE` lexfile to tune itself automatically.

`NOISE` pays no attention to flag settings. It always works, even if `BEEP OFF` (flag -2) is set, and is always loud, even if flag -25 (the `LOUD` flag) is clear. If (for some obscure reason) you want it to be quiet, you can modify the `NOISE` lexfile to play everything quietly:

```
POKE DTH$(174+HTD(ADDR$("NOISE"))), "8"
```

Poke a "4" to change it back to the loud setting. Better yet, use the HP-71 earphone module for private listening.

As listed, NOISE is LEX ID #5E, and token #CC. Use John Baker's program "RESOURCE" (V2 N7 P15b) to change the ID and token # if desired. NOISE has proliferated on magnetic media under the LEX ID #5F, token #01, which conflicts with the lexfile keywords COLOR, DT, and DAY, all of which are also 5F 01.

Available music files: WTELL (by somebody at HP); MUSETTEA & MUSETTEB, INV8A & INV8B (four Bach pieces, entered by James Horn for HP-71 stereo duets); MARYLAMB and ROWBOAT (entered by Joe Horn), WHYKNOT (a piano ditty by Greg Jones, entered by Joe Horn), and PHASER (produces a startling sound effect). Get them all at your next CHHU Chapter meeting! Astound your friends, humiliate your HP-41, and drive your fiancé up a wall!

[Note added in 2015: MUSIC and all the related files mentioned here are available online on the [HORN/GOODY1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy MUSIC and the other files directly to your HP-71.]

MUSIC BASIC 698 bytes

```

1 ! MUSIC by JOSEPH K. HORN
2 ! REQUIRES NOISE LEXFILE by STEVE WILLIAMS
3 ! TO TUNEUP, RUN TUNEUP by JAMES L. HORN
10 DESTROY ALL @ N$="CcDdEeFfGgAaB" @ T$="" @ DIM A$[96]
20 INPUT A$ @ IF NOT LEN(A$) THEN 50
30 IF A$[1,1]="=" THEN 190
40 DIM M$[LEN(A$)+LEN(T$)] @ M$=T$&A$ @ DIM T$[LEN(M$)] @ T$=M$ @ GOTO 20
50 DIM M$[LEN(T$)*2] @ D=130 @ D$=CHR$(160)
60 FOR X=1 TO LEN(T$)
70 N=POS(N$,T$[X,X])
80 IF N THEN M$=M$&CHR$(D+N)&D$ @ GOTO 130
90 IF T$[X,X]="R" THEN M$=M$&CHR$(255)&D$ @ GOTO 130
100 IF T$[X,X]="(" THEN
    D$=CHR$(LQG(VAL(T$[X+1]))/LOG(2)*16+192) @
    X=POS(T$,")",X+1) @ GOTO 130
110 IF POS("+-",T$[X,X])=0 THEN BEEP @ DISP T$[X,X];" @";X;" ???" @ END
120 IF T$[X,X]="+" THEN D=D+16 ELSE D=D-16
130 NEXT X
140 NOISE M$&CHR$(255)&CHR$(0)
150 INPUT "Save to file: ";A$
160 IF NOT LEN(A$) THEN END
170 SFLAG -1 @ PURGE A$
180 CREATE TEXT A$,LEN(M$)+2 @ ASSIGN #1 TO A$ @ PRINT #1;M$ @ END
190 UNSECURE A$[2] @ ASSIGN #1 TO A$[2]
200 DESTROY M$ @ DIM M$[MEM/2-100] @ READ #1;M$ @ GOTO 140

```

NOISE ID#5E 301 bytes

```

0123 4567 89AB CDEF ck

000: E4F4 9435 5402 0202 1A
001: 802E 0082 1020 1000 56
002: A520 0E5C CCC0 0000 14
... continued on next page ...

```

```

003: F710 0000 0000 0000 A7
004: 0530 00D9 E4F4 9435 98
005: 54CC 1FF8 FAB6 308D D2
006: 2713 08D3 9450 9FFF 76
007: F6EF FF8F 681F 08F1 76
008: 3DB0 7A61 1098 4C80 1E
009: 8FAF 2D68 1EBF 2BF2 54
00A: BF2A D767 B032 F14A 4B
00B: B731 0075 B0B8 C318 7E
00C: 3237 7D03 3F30 0AF4 EF
00D: F027 B90B 90B9 023A B4
00E: 94B1 A205 8132 F10A 1B
00F: B7AF 1318 CAE5 6ACF BA
010: 8141 00A5 F4E5 3104 BF
011: 7B50 31E3 2670 8029 2B
012: B91B 91B9 120B DD32 46
013: F101 10AC 680D FAB8 C0
014: 91B2 0A3D 5CFA AF80 16
015: 187C 41A5 853E 20A5 AB
016: F460 654F 8D84 A80D DD
017: 015B 0170 C4C4 A6A1 24
018: 19C2 134D 215E 3AF1 65
019: D5AE 015B 0170 01A6 A2
01A: A80C F203 1300 E628 A9
01B: 0DF6 600A 15A6 E59F 24
01C: AA08 1C81 C203 1018 09
01D: 0DFB 6A51 1B95 B645 07
01E: 9F4B 0B91 A6C5 9F20 65
01F: 0170 801A 5005 5030 25
020: 50BB 4077 4073 40BF B7
021: 301C 30C8 3095 3092 2E
022: 30BF 200D 208A 2028 A9
023: 20E5 2096 2048 201A 25
024: 20EB 20DD 20EF 2002 7A
025: 3034 3086 30F8 307B 30
026: 301E 30D0 40B3 40B6 25
027: 40D9 4007 01      0A

```

TUNEUP BASIC 351 bytes

```

1 ! TUNEUP by JAMES L. HORN
2 ! REQUIRES (AND MODIFIES) THE NOISE LEXFILE
10 L=502+HTD(ADDR$("NOISE")) @ S=0 @ S$=PEEK$("2F977",4)
20 FOR J=4 TO 1 STEP -1 @ S=16*S+HTD(S$[J,J]) @ NEXT J
30 K1=S/27.4 @ K2=S/64
40 FOR I=0 TO 15 @ F$=DTH$(K1/2^(I/12))
50 FOR J=0 TO 3 @ POKE DTH$(L+4*I+J),F$[5-J,5-J] @ NEXT J @ NEXT I
60 L=L+64 @ FOR I=0 TO 15 @ F$=DTH$(K2*2^(I/16))
70 FOR J=0 TO 3 @ POKE DTH$(L+4*I+J),F$[5-J,5-J] @ NEXT J @ NEXT I
80 DISP "Tuned!" @ END

```

Titan File #9 ... never existed. It somehow got skipped during the transition from CHHU Chronicle to HPX Exchange.

Titan File #10 (HPX Exchange, V1 N1, Jan/Feb 1987, pages 24-27)

by Joseph K. Horn <53>

Welcome to "The Titan File"! This column is by and for HP-71 ("Titan") users. Anything and everything you or I can dream of that makes the HP-71 earn its keep belongs here! And if you remember the Titan File from the CHHU Chronicle, then welcome back! We'll just continue here with HPX (thanx, Brian!) where we left off in CHHU. That makes this Titan File #10. If you don't have any of the earlier columns, contact me for photocopies. This one will continue what [column #7](#) began: Members' questions and the HP-71 Answer Man's answers. Since that column had 21 questions & answers in it, here we go with Q22:

Q22. Paul Silberg of Salem, Oregon, and Lee Kenderdine of Antioch, Tennessee, want to know why `COPY A$:TAPE` doesn't work, even if `A$` contains the name of a file that is on tape or disc.

A22. Syntax error! BASIC, like all languages, has strict rules of syntax. You can:
`COPY A$&" :TAPE"`

or

`COPY :TAPE TO A$`

but you can't

`COPY A$:TAPE`

because that's mixing variables (`A$`) and unquoted string literals (`:TAPE`) in an illegal way. The first option, `COPY A$&" :TAPE"` uses the string concatenator (`&`) to join `A$` with a quoted string literal (`" :TAPE"`) which is legal. The second option, which I prefer by far, uses no quotes, no concatenator, no monkey business; it's great in programs.

Q23. Fred Farr, of Minneapolis, Minnesota, and practically everybody else, asks if there is an up-to-date lexfile list.

A23. No. As soon as I update mine, rumors drift in about a whole disk full of hot new lexfiles from some desert island. But if you're interested in what I have at the moment, send a self-addressed stamped envelope (NO MONEY!!!) and ask for my current lexfile list (that's ALL lexfiles known to me, not just the ones I wrote).

Q24. André Côté of Ste-Foy, Quebec (Canada) asks why they didn't put `MAT READ` in the HP-71 Math ROM. After all, they did put it in the HP-75 Math ROM; why did we get short-changed?

A24. Because the bare-bones HP-71 has it! BUT it isn't called `MAT READ`, it's just plain old `READ`. If you use a variable that has been `DIM`'ed, then `READ` will automatically fill the array or matrix with `DATA`. For example,

```
10 DIM M(5)
20 READ M
30 DATA 123,234,345,456,567
```

will nicely fill all 5 elements of array `M` with the `DATA` values. No need for `MAT READ`; just use `READ`. It's the HP-75 owners who got short-changed!

Q25. André also asks whether keywords in lexfiles can be isolated into their own, smaller lexfiles, and whether or not linked lexfiles can be unlinked into their original lexfiles.

A25. Yes, and yes! Although there is no easy way to surgically remove (lextract?) a single keyword (or specific set of keywords) from a lexfile, it can be done IF you have a disassembler, an understanding of the HP-71 IDS that's good enough to cope with your disassembler's deficiencies, enough time to wade through the disassembly listing to find what you're looking for and to retype it as a new source code file, and an assembler. It's not as mind-boggling as writing original code, but it's time consuming. I've gotten pretty good at hacking up lexfiles and making new ones with the keywords I want. If you need any help, let me know. Regarding unlinking lexfiles, there is a beautiful program called LEXSPLIT that asks for the name of the lexfile you're going to split up, tells you how many lexfiles it originally was before they were linked, asks for names for each of them, and makes NEW unlinked lexfiles, leaving the original linked lexfile intact! I wish my copy of LEXSPLIT had an author's name in it, because I've plumb forgotten who wrote it. Send SASE for a copy.

[Note added in 2015: LEXSPLIT is available online on the [📁 HORN/MAIN01 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy LEXSPLIT directly to your HP-71.]

Q26. Bill Collins of Boulder, Colorado, asks what lexfiles exist that would help with HP-71 ThinkJet graphics.

A26. If the GRAPH71 package (available from EduCalc) doesn't do what you need, then these LEX files might help:

LEXFILE – KEYWORD(S)

- BITBYTLX – BITBYTE\$(,\$,#) turns bits into bytes. [[📁 SWAP/lexfl1](#)]
- D2DLEX – DOT2DOT\$(,\$,#,#) “widens” bits any specified width. [[📁 SWAP/lexfl1](#)]; [[📁 DOC](#)]
- FLIPLEX – FLIP\$(,\$) flips the order of the bits in each byte. [[📁 SWAP/lexfl1](#)]
- HIGHLEX – HIGH\$(,\$) sets the high bit of every byte. [[📁 SWAP/lexfl1](#)]
- MAPLEXB – MAP\$(,\$,\$,\$) replaces any byte(s) with any other. [[📁 SWAP/lexfl1](#)]; [[📁 DOC](#)]
 MAP file,\$,\$ does the same on an entire TEXT file.
- PATTERN – PATTERN\$(,\$) converts text to a GDISP pattern. [[📁 SWAP/lexfl1](#)]; [[📁 DOC](#)]
- PRNLEX – BEGINGR\$, ENDGR\$, HIGHGR\$, LOWGR\$, RASTER\$(#), and tons of non-graphics ThinkJet keywords. [[📁 SWAP/lexfl1](#)]
- RGCMSD – CPRSRG\$(,\$) compresses graphics into fewer bytes. [[📁 SWAP/lexfl1](#)]; [[📁 DOC](#)]
 EXPDRG\$(,\$) expands compressed data back to normal.
- ROWCOL – ROWCOL\$(,\$) transposes row-graphics to column. [[📁 SWAP/lexfl1](#)]
- SBITLEX – SBIT\$(,\$,#,?#,#) sets or clears any bit. [[📁 SWAP/lexfl1](#)]; [[📁 DOC](#)]
 SBIT\$(,\$,#,#) tests any bit in a string.
- SHRINKLX – SHRINK file; shrinks space at end of TEXT files. [[📁 SWAP/lexfl1](#)]
- SPCKLX71 – SPACK\$(,\$) packs text strings into fewer bytes. [[📁 HORN/LEX01](#)]
 SUNPACK\$(,\$) unpacks packed strings back to normal.
- STRB00L – AND\$(,\$,\$) bitwise boolean AND on two strings. [[📁 SWAP/lexfl1](#)]
 OR\$(,\$,\$) bitwise boolean OR on two strings.
 EXOR\$(,\$,\$) exclusive-OR on two strings.
 COMP\$(,\$) bitwise boolean NOT on a string.
 REVBIT\$(,\$) flips the order of the bytes' bits.

Also available in many lexfiles are these four useful functions:

- REV\$(,\$) reverses the order of the bytes in a string. [REVLEX: [📁 SWAP/lexfl1](#)]
- RPT\$(,\$,#) repeats a string any number of times. [RPTLEX: [📁 SWAP/lexfl1](#)]; [[📁 DOC](#)]
- STRSUM\$(,\$) adds the ASCII values of all the bytes. [STRSUMLX: [📁 SWAP/lexfl1](#)]; [[📁 DOC](#)]
- CNTBITS\$(,\$) counts bitwise how many 1's are in the string. [FATLEX: [📁 SWAP/lexfl1](#)]

[Note added in 2015: The lexfiles listed above are available online in various LIF disk images. If you have a PIL-Box, you can download the disk image by clicking on the download link given above, and then copy the lexfile directly to your HP-71. Some lexfiles also have documentation available online; click to download their plain-text DOC if shown above.]

Q27. Eric Larkin of Union City, California wants to know how to recall the dimensions of a variable after dimensioning it.

A27. If you have the Math ROM, use `UBND` and `LBND`. For example, `DIM M(3,4)`. Then `UBND(M,1)` gives 3, and `UBND(M,2)` gives 4. `LBND(M,1)` gives either 0 or 1, depending on whether you had `OPTION BASE 0` or `OPTION BASE 1` active when you DIMensioned M. If you don't have a Math ROM, you should go buy one. If you can't, then you should weep and gnash your teeth. If that doesn't help, you can always `PEEK` at a variable's dimensions, as in the program `LISTVARS` on the French swapdisk. A full discussion of that is beyond the scope of this Q&A column.

[Note added in 2015: The `UBND` and `LBND` commands can also be added to the HP-71 by installing the `UBOUNDLX` lexfile, which contains only those two commands. It is available online on the [📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy `UBOUNDLX` directly to your HP-71.]

Q28. Keith Gordon of Munsonville, New Hampshire, asks how to get Greek letters on the ThinkJet.

A28. You can't. Of course, you can create them using ThinkJet graphics, and if you really need them, go ahead, but it'll take a bit of doing to design them all, dot by dot. The ThinkJet does not have a Greek font, or Greek mode, or Greek character set, or anything like that. Sorry.

Q29. Keith also asks how to list a `STAT` array's contents and edit the individual entries in it.

A29. Sorry again! If you mean that you want to list the individual data that you `ADDED` into the `STAT` array, forget it! It cannot be done, because they don't exist. When you `ADD` data, the `STAT` array is modified to reflect that addition, but the data itself is thrown away and is not stored anywhere. In effect, just the running average and a number related to the standard deviation are stored in the `STAT` array, not the individual data. But if you mean that you want to access the values that `ARE` in the `STAT` array (even though they are `NOT` the data you `ADDED`), that is very easy, and is done as for any array, e.g. `DISP A(3)`, or `A(3)=153`, etc.

Q30. Dr. Martin Koenig of Cincinnati, Ohio, points out that the "TIMEFMT" subprogram in the HP-71 Utilities Book gives results like "05:14:04. AM" and wonders how to get rid of that extraneous dot.

A30. Excellent question! `BASIC` is supposed to be easy, but look how messy getting rid of one lousy dot can be!

Solution 1: Change HP's subprogram like this. Change line 290 to:

```
290 S=IP(FP(T*100)*100)
```

and delete line 340. That seems to work.

Solution 2: Don't just CALL TIMEFMT(T,T\$,12). Instead, use this in your calling program:
STD @ CALL TIMEFMT(T-RED(T,1E-4),T\$,12)
That seems to work too.

Solution 3: Leave the subprogram alone, and call it normally, but then don't use T\$ as is; use T\$[1,8]&T\$[10]. That leaves out the dot! (This is the most trivial solution).

Solution 4: Don't even use that subprogram. There are lexfiles that do it faster, with no dots. For example, the TCNV\$(#) function in the TCONV lexfile takes a decimal number of seconds and converts to HH:MM:SS format. The HMS\$(#) function in the TIMELEX file takes a decimal number of hours and does the same.

[Note added in 2015: TCONV is available online on the [📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy TCONV directly to your HP-71.]

Q31. James Cave of Princeton, Texas, wants a complete bug list for the HP-71.

A31. So do I! The one I have is huge, but it's still growing! If you'd like a photocopy of what I've got, just send an SASE and a request for the HP-71 buglist.

Q32. James also asks for the shortest and fastest way to toggle a flag.

A32. If the flag number is F and X is a scratch variable, then a short & very fast way is:
X=FLAG(F,NOT FLAG(F))

Notice that this suggests a short & fast way to SWAP two flags' values. If the flags are F and G, and X is scratch, then X=FLAG(F,FLAG(G,FLAG(F))) swaps 'em quick as a wink!

Q33. Dr. August Nechi of Stone Mountain, Georgia, notices that nothing happens when he tries to turn on "blink mode" by POKE "2E3FF", "3". What's wrong?

A33. Nothing's wrong. Blink mode is disabled by ANY display change, and pressing ENDLINE after a command (such as POKE "2E3FF", "3") clears the display, thereby terminating blink mode (which was enabled for one zillionth of a second). To see blink mode in action, POKE "2E3FF", "3" @ WAIT 5. The display blinks for 5 seconds (5 blinks) and then clears. Or try this: "Blink Mode!" @ POKE "2E3FF", "3". Since there is something to display, blink mode won't stop until you interrupt it. Try it again after setting DELAY 3. The POKE doesn't take effect until the 3-second DELAY is past.

Q34. Richard de Queiroz of Palos Verdes Estates, California, asks if there is a way to change the beep-margin from 96 to 80.

A34. Yes and no. Yes, you can get any of various lexfiles that add the MARGIN command, and use it to set MARGIN 80 (as the HP-75 does). No, you can't move the "hard margin" over; you'll always be able to type 96 characters, even if the "beep margin" is at the 80th column. A method of programming limited-length inputs using non-readable display characters is described in the HP-71 Software Developers' Handbook, page 9-6.

[Note added in 2015: MARGELX is available online on the [📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy MARGELX directly to your HP-71. Its

plain-text documentation is available here: [📄 DOC](#). Another lexfile on the same disk is called MARGELEX; try both and see which you prefer.]

Q35. Howard Snapp of Waukesha, Wisconsin, notices that the GEDIT program (in the HP-71 Software Developers' Handbook) doesn't print correctly, and asks if it can be fixed.

A35. You're right; it doesn't work! Whoever wrote it did NOT have the HP-71 or the ThinkJet in mind. Yes, it can be fixed; you need a copy of D2DLEX in memory though. Just add these lines to the program:

```
280 PRINT CHR$(27);"*rA"; @ FOR J=0 TO 7 @ FOR K=1 TO 4
282 PRINT CHR$(@&);"*b66W";DOT2DOT$(A$,J,4) @ NEXT K
284 NEXT J @ PRINT CHR$(27);"*rB"; @ GOTO 50
```

Try values other than 4 in lines 280 and 282 for different sized printouts. In case your version of the Software Developers' Handbook doesn't have the other bugs out, also be sure to make these changes to GEDIT too:

Line 20 should end with: @ PWIDTH INF

Delete line 60 and replace with: 60 SFLAG 5

Delete line 90 and replace with: 90 GOSUB 290

Line 100 should be only: 100 K\$=KEY\$

Line 200 should have: @ GDISP A\$ @ A\$=GDISP\$ inserted between the READ and GOTO statements.

Lines 210,220 and 230 should end with: @ GOTO 50

Line 260 should be: 260 INPUT "X,Y:",STR\$(X)&,",\$STR\$(Y+1);X,Y

Q36. Ed Winfield of Hollis, New York asks how to interrupt a program that has a lot of inputs in it, take a break for lunch, then come back and resume where you left off without starting all the inputs all over again.

A36. Simple: just press the [OFF] key at an input prompt, and the HP-71 will immediately turn off. Enjoy lunch. When you return, press [ON], and you will see the "SUSP" annunciator telling you that the program was suspended. Press the [CONT] key, and you will be whisked immediately back into action, precisely at the same input that was running when you pressed [OFF]! There is only one exception that I can think of. If you press [OFF] during a MAT INPUT (a Math ROM specialty), you will have to begin all over again with array element 1 when you CONTInue. Since the Math ROM displays the element number expected, and since the partially typed array is still available in the command stack, this should not be a problem at all. If it is, plug in your HP-71 to AC, set flag -3 (the continuous ON flag), and just leave the '71 turned on, right through lunch!

Q37. Jim Bell of Honolulu, Hawaii, asks for the memory address of the display.

A37. Unfortunately, the display is NOT mapped into a single chunk of memory, but three chunks, and they're not even the same size. To satisfy your curiosity, here are the addresses. The leftmost 46 display dot columns (numbered 0 thru 45) reside at hex 2E104 thru 2E15F. The middle 48 columns (numbered 46 thru 93) reside at hex 2E200 thru 2E25F. The rightmost 38 columns (numbered 94 thru 131) reside at hex 2E300 thru 2E34B. The even-numbered addresses are the top four dots of each column, and the odd-numbered addresses are the bottom four dots. So try this: "FA" @ POKE "2E10C", "F7". See what happened? The "FA" was displayed, then the POKE

drew a line on the right side of the "F" making it look like a squarish "A".

Q38. Donald Humbert of New York City wants to know my definition of prime numbers.

A38. MY definition? As opposed to THE definition? Okay; my definition: A Prime Number is a natural number with exactly two natural number factors. ("Natural number" = integer greater than 0; also called the "counting numbers".) Notice that this definition doesn't fit 1, which has only one natural number factor (itself). 2 is the first prime number, its factors being 1 and 2. This agrees with THE definition, but I like mine better because it's easier to understand. "Do two and only two natural numbers go into it with no remainder? Then it's prime."

Q39. Bill Figueroa of Richardson, Texas, has the Variable Cross Reference program from the HP-71 Utilities Book, but can't use it because it bombs out if there are more than 63 variables for it to handle. Is there a better one?

A39. Yes. Just send an SASE and request "VARXREF", an improved version of the Variable Cross Reference program. This one has NO limits. If you want it on card or disc, send 2 cards (it's 1933 bytes long) or a disc, and enough postage on the SASE.

[Note added in 2015: VARXREF is available online on the [📁 HORN/MAIN01 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy VARXREF directly to your HP-71.]

Q40. Hans Decarli of Pine Grove, California, doesn't have a printer, so he programs his HP-71 to stop at every display so that he can write down the output, and then makes the program resume. The problem he notices with this approach is that the HP-71 is doing nothing when he's writing down each output, and he's doing nothing when the HP-71 is running; what a waste of time! He wonders if there is some way to have the HP-71 running while he's writing down the answer.

A40. YES!!! This will sound odd, but think about it. All you have to do is pause BEFORE each DISP, instead of after. You can't use DELAY for this. A simple keystroke-seeking line is:

```
30 IF NOT LEN(KEY$) THEN 30
```

If you have the KEYWAIT\$ or KEYSLP\$ function or one like them, then you can simply say A\$=KEYWAIT\$, with A\$ being any scratch variable. Notice what happens when you do this? When you press a key, the next output is immediately displayed (because we paused right before the DISP), and then the program goes running full steam in search of the next output, *while you are writing down the last output*. No time wasted; you are working, and the HP-71 is working. No solution's perfect, however. Notice that this method prevents us from ever getting the very FIRST output; the program will pause itself right before the first output! My solution to this is silly, but it works: I start the program with PUT "X". (You can PUT any character you like.) This makes the first pause get skipped, because it thinks that a key was pressed!

```
10 PUT "X"  
20 GOSUB 'SEARCH' @ BEEP 1400, .07  
30 IF NOT LEN(KEY$) THEN 30  
40 DISP whatever the output is  
50 GOTO 20  
60 'SEARCH' @ etc.
```

The BEEP in line 20 is a bonus: the HP-71 chirps when it finds the next output, so that you know, but it doesn't change the display, in case you're still writing down the previous one.

[Note added in 2015: The lexfiles KEYWAIT (KEYWAIT\$) and KEYSLEEP (KEYSLP\$) are both available online on the [📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy one or both lexfiles directly to your HP-71.]

Q41. Harold F. Byrd of Chula Vista, California, asks if there is any way that the "LOCKOFF" lexfile can be used to unlock somebody else's HP-71.

A41. Harold, really now. Many people rely on the LOCK feature of their HP-71's to prevent unauthorized access to their machines. Why? Perhaps they have data in there that should not fall into the wrong hands. Perhaps they are developing brilliant new software and they don't want their ideas stolen. Perhaps they simply have stuff in there that they don't want messed up by some knucklehead with hyperactive fingers. They have a right to believe that LOCK is what HP said it is: a security device. So how could I in good conscience publish the method (quite simple, really) of using "LOCKOFF" or "NOSTRTUP" to unlock somebody else's HP-71? No, no, I can't...

[Note added in 2015: LOCKOFF is available online on the [📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy LOCKOFF directly to your HP-71.]

Q42. Evan Magnuson of Hammond, Louisiana, asks if there is any way to print out the command stack.

A42. It's easy if you have the PEEKUTIL lexfile (also called UTILLEX); all it takes is this little routine:

```
10 A=PEEK("2F576",5)
20 FOR X=PEEK("2F976")+1 TO 1 STEP -1
30 L=PEEK(A,3)
40 IF NOT L THEN 60
50 PRINT STR$(X)&": "&TEXT$(A+3,L/2-1)
60 A=A+L+6
70 NEXT X
```

If you don't have PEEKUTIL, it would take less effort to send me an SASE (with mag card or disk if you want it recorded) than it would take to rewrite this to just use PEEK\$.

[Note added in 2015: PEEKUTIL is available online on the [📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy PEEKUTIL directly to your HP-71.]

Q43. Evan also asks if the alarm annunciator is ever used in the HP-71.

A43. Not really. The Debugger that HP sells uses it as a "Debugger Active" annunciator, but the Debugger manual tells how to change the software so that any other annunciator can be used instead. SUPERLEX, sold by Titan Software, contains keywords for turning the alarm annunciator on and off. Nothing uses it seriously. However, I just got a letter in the mail today (dated 24 June 1987) from a young lexfile expert who says he's thinking about writing "an alarm lexfile which is similar in operation to the HP-41 XYZALM function." That's good news! And who knows; he might give the alarm annunciator some real use, after all these years!

Q44. Vern C. Hunt of Orange, California, is looking for a better way of printing a line of dashes than the obvious and explicit `PRINT "-----"`.

A44. Use `USING! PRINT USING "60'-' "` will nicely print 60 dashes. If you are a total speed demon, `PRINT RPT$("-", 60)` is a tad faster, but you need a lexfile that contains `RPT$`.

Q45. Lotsa folks from all over have asked if there is a short routine that converts from decimal degrees or hours (HR) to hours-minutes-seconds format (HMS), like the HP-41 has.

A45. Just add these two UDF's (user-defined functions) to your program that needs the conversions:

```
10 DEF FND(X)=X+FP(X)/1.5+FP(X*100)/90 ! FND(x)=HR(x)
20 DEF FNT(X)=X-.4*FP(X)-.004*FP(X*60) ! FNT(x)=HMS(x)
```

Then, whenever you need to convert decimal hours (or degrees) to H.MMSS format, just use `FNT(x)` as if it were a built-in function. (The "T" in `FNT` stands for "Time format"). To convert H.MMSS back to decimal hours, use `FND(x)`. (The "D" stands for "Decimal format"). The comments at the end of each line are just in case you forget which is which. The line numbers can be anything at all, but the `DEF`'s **MUST** reside within the main program or subprogram that uses them. You needn't have both if you only need to convert one way. You'll love how **FAST** these are!

Q46. Tim Eide of Ventura, California, bought the Debugger but was upset to find that it contains the software on disc, not on cards. Since he owns only a card reader, he asks if the Debugger is available on cards.

A46. Send me 11 mag cards, an SASE with enough postage on it, and a request for `DBGLEX1A`, `DBGLEX2A`, `DBGLEX3A` and `DBGMAINA`.

[Note added in 2015: All four lexfiles are available online on the [📁 HORN/LXSORT LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy the lexfiles directly to your HP-71. These are dangerous, low-level utilities; be sure to follow the directions to avoid a Memory Lost!]

Q47. Horst Debushman of Frankfurt, West Germany, wants to know if there is a lexfile that can put the HP-71 into gradian mode.

A47. No, and for good reason. To convert from degrees to gradians, all you have to do is divide by .9; no lexfile required for that! Multiply by .9 to go from gradians back to degrees. Anyway, what's a German using Japanese angles for? It reminds me of the time a fellow walked into the HP booth at the Consumer Electronics Show, and hassled a poor HP employee about a calculator on display because it didn't have hyperbolics. The HP rep in all sincerity asked him, "But what are hyperbolics used for?" It deflated the guy completely, because he had no idea what hyperbolics are used for! He probably saw it on other calculators, and missed it on the HP, and thought it "should" be there. Gradian mode? Gimme a break.

Q48. Michael Scano of Palo Alto, California, likes the way `PRINT USING` can format things, but he wonders if there is some way of getting it to output to a string variable instead of just to a printer or display.

A48. Yes! There is a lexfile called FMTLEX that adds two keywords to BASIC, one of which is PRINT T0. This new version of PRINT uses USING like normal, but dumps the output into a string variable! For example,
PRINT USING "'Total:', '\$'3dc3d.2d" T0 A\$;12345.67;
places the string "Total: \$12,345.67" into A\$. Of course, now that it's in a string, you can put it in a file or whatever you want with it. For a copy of FMTLEX, send SASE, 1 mag card if you want it recorded, and a request for FMTLEX. It also contains IMAGE\$, a way of pre-processing IMAGEs before USING them. [[📄 download its plain-text documentation here](#)]

[Note added in 2015: FMTLEX is available online on the [📄 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy FMTLEX directly to your HP-71.]

Q49. Michael Markov of Lockwood, New York, was somewhat put out by the way I seemed to give my blessings to software piracy in [Titan File #7, Questions 2 and 3](#). Isn't this wrong?

A49. Software piracy is illegal, bad, and stupid. Illegal because enough citizens demanded it. Bad because benefitting from the work of another without giving them just remuneration for their labor is immoral. And stupid because it ultimately has two horrendous effects: software is sold with awful copy protection schemes that make everybody (especially hard-disk users) very upset, and the authors of the really good software (the stuff that gets pirated the most) quickly get discouraged and stop writing code, and our sources of elegant programming disappear. Mike, I couldn't agree with you more about software piracy: it is indeed A BAD THING. In Titan File #7, I did not recommend or advocate software piracy. I suggested that if a user wants a MINOR PART OF A WHOLE, when that part is only available by purchasing the whole, then go ahead and copy it from a friend or dealer or whatever. It's like reading a friend's Calvin and Hobbes comic book, and finding one strip in it that really makes you howl, so you borrow the book and photocopy that one page. Is that breach of copyright? Of course not. Photocopying half the book is another story entirely! If somebody wants the HP-71 Software Developers' Handbook, I tell them to go buy it. But if they want just the STRINGLX lexfile, which amazingly is not available through the Users' Library, then I just give them a copy. This is not software piracy. Of course, this is my opinion. Not everybody agrees with me. Not everybody is right.

Titan File #11 (HPX Exchange, V1 N2, Mar/Apr 1987, pages 14-16)

by Joseph K. Horn <53>

Contents:

1. [Symbolic Math \("SYMBOLIX"\)](#)
2. [Computer-Generated Mazes](#)
3. [Spaghetti Code](#)

1. "SYMBOLIX" IS HERE!

The HP-28's ability to perform symbolic math impresses everyone who sees it (except Harry Bertuccelli?), AS LONG AS it's one of the examples from the book, or something equally simple. Don't try to impress anybody with a "real" problem, because you will cause the HP-28 either to go comatose, or to run out of memory, neither of which is impressive. But some of us don't want to impress anybody; we just want the job done.

If that's your reason for shelving your HP-71 and forking out a laughable cut of your salary for an HP-28, forget it! The HP-71 can do symbolic math too! AND BETTER THAN THE HP-28! (You may want an HP-28 for its other features, however, like its excellent stack logic & display.)

A "freeware" program called "SYMBOLIX" gives your HP-71 the ability not only to perform full symbolic algebra and derivation, like the HP-28, but also FULL symbolic integration (not only on polynomials)! The problem that Harry said couldn't be done on the HP-28 "in less than 10 minutes" (a tremendous exaggeration) is performed by SYMBOLIX in just over one minute, with exactly the answer that he wanted! (Cf. HPX V1 N1 P12 first line) That's not bad, considering that it's written ENTIRELY in HP-71 BASIC (no lexfiles or ROMs required).

The listing is NOT published here, because it is 20K long, and takes forever to type in, not to mention the time it takes to find bugs caused by bad typing. Most of the 644 program lines are full length, and many are too long to type in (they were entered by Monkey Business techniques). It's freeware, but is copyrighted with all rights reserved, so don't get ideas of making a ROM out of it. It's an HP-71 rewrite of a program written by The Soft Warehouse for the Atari computer in 1982, and therefore I cannot charge for it NOR can I allow others to profit (\$\$) from it. It is PRIVATE, and even if you bust that, you still won't be able to LIST it (more Monkey Business). If you like it, RUN it; if you don't like it, PURGE it. Don't even try to modify it; ANY modification at all will decompile the GOTO/GOSUB offsets, and it won't run. (There are no line numbers. They are removed by ... some Monkey Business!).

A full set of instructions will be included with your copy of SYMBOLIX. To whet your appetite, here are its features:

ALGEBRAIC EXPANSION & SIMPLIFICATION. Example: $(2X^{2Y}+3Z)^3$ returns $8X^6Y^3+36X^4Y^2Z+54X^2YZ^2+27Z^3$ (that's Harry's example!). This is what the HP-28 SHOULD do when you press EXPAND.

SYMBOLIC AND CONSTANT VARIABLE ASSIGNMENT. Example: $E=EXP(1)$ sets the variable E equal to the constant "e". $N=B/C$ sets the variable N equal to "B/C", and then, whenever B & C change, N changes accordingly! Self-referential and circular assignments are possible. This is the same as the HP-28's STO function.

ALGEBRAIC EXPANSION OPTION. Turn on/off expansion of product logs into log sums, and trig

functions into sines and cosines. Example: after performing the $E=EXP(1)$ assignment mentioned above, then $LOG(EA^2)$ returns $2LOGA+1$. The HP-28 does not offer this option; it either always does it, or never does, depending on the function.

HARMONIC EXPANSION OPTION. Turn on/off expansion of sine & cosine products and powers into angle sum or multiple angle identities. Example: $SINA^3$ returns $.25COS(3A)+.75COSA$. The HP-28 does not offer this option either.

PARTIAL DERIVATIVES. Example: $AX^3+SINX \%X$ (“%X” means “derive with respect to X”) returns $3AX^2+COSX$. The HP-28 does this quite well. But I never got it to do $SINX/X \%X$ (it always ran out of memory after thinking for a long time), but SYMBOLIX gets $-SINX/(X^2)+COSX/X$ right away.

ANTIDERIVATIVES (INTEGRALS). Example: $4X^3-2X \$X$ (“\$X” means “integrate with respect to X”) returns X^4-X^2 . The HP-28 performs this on polynomials quite well. But it can't do it to non-polynomials, which SYMBOLIX handles just fine. Example: $COS(LOGX)/X \$X$ returns $SINLOGX$.

The re-write from Atari to HP-71 BASIC involved a few improvements. The trig functions work correctly according to **DEGREE** or **RADIAN** mode (in the Atari they were always radians). Atari BASIC does not have **ASIN** or **ACOS**; they had to be calculated. The fast HP-71 **RMD** function replaced a lot of number crunching in the Atari version. Whereas variables are faster than numeric literals in Atari BASIC, the opposite is true in the HP-71, so the “constant” variables were replaced by literals, for speed. It runs faster on the HP-71 than the Atari version runs on the Apple][e!

Sound good? For your **FREE** copy of SYMBOLIX, send either 16 **GOOD** mag cards or one **GOOD** 3.5" disc (no cassette tapes!), a self-addressed envelope with enough postage on it, a request for “SYMBOLIX”, and **NO MONEY**. (Address below). The instructions will be printed on ThinkJet paper kindly donated by Jeremy Smith.

[Note added in 2015: SYMBOLIX is available online on the [📁 SWAP/swap11 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy SYMBOLIX directly to your HP-71. Its documentation is here: [📄 DOC](#)]

2. COMPUTER GENERATED MAZES

“MAZE” is a BASIC program that generates and prints random mazes. They are the simplest type: enter at the top, follow the path, and exit the bottom. Nothing fancy. But it works!

To use, just input the desired size (vertical and horizontal dimensions; try **10, 10** for starters). The top line will be printed immediately, then the HP-71 will silently calculate the rest of the maze. This may take a lot of time; be patient (10 by 10 takes about 30 seconds). The maze will be printed when all the paths are calculated. Due to the random nature of the maze generation process, it is possible for the calculated maze to be non-solvable (no way out), but the program detects this, and in this case re-runs itself to try all over again.

The listing here assumes that you are using a ThinkJet printer. Line 30 assigns to **E\$** the escape code for a half-line feed. If you are not using the ThinkJet, change line 30 accordingly. Also, the maze is made of ThinkJet character 252, the little square block (■). If you don't have a ThinkJet, use some other appropriate character.

The two mazes printed here are both 5.5 inches on a side, but differ in character width and line height. The maze with fatter paths used an input of 32, 32 in ThinkJet default modes. The maze with skinny paths used an input of 44, 58 in compressed mode at 8 lines per inch. Experiment with your printer's modes and characters! (How about graphics mode?...) [Note added in 2015: The mazes mentioned here were accidentally not published in the CHHU Chronicle.]

[Note added in 2015: MAZE is available online on the [📁SWAP/swap10 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy MAZE directly to your HP-71.]

MAZE BASIC 1641 bytes

```
10 ! MAZE ! Joseph K. Horn
20 INPUT "V,H? ";V,H @ T=V*H+1 @ OPTION BASE 1
30 E$=CHR$(27)&"="&CHR$(13) @ RANDOMIZE @ CFLAG 1 @ PWIDTH INF @ ENDLINE
40 X=CEIL(RND*H) @ FOR I=1 TO H @ IF I=X THEN PRINT "■ "; ELSE PRINT "■■";
50 NEXT I @ PRINT "■";E$;
60 DESTROY W,L @ INTEGER W(H,V),L(H,V)
70 W(X,1)=1 @ C=2 @ S=1 @ R=X @ GOTO 100
80 R=RMD(R,H)+1 @ IF R#1 THEN S=RMD(S,V)+1
90 IF NOT W(R,S) THEN 80
100 CFLAG 0 @ IF R=1 THEN 290 ELSE IF W(R-1,S) THEN 290
110 IF S=1 THEN 190 ELSE IF W(R,S-1) THEN 190
120 IF R=H THEN 140 ELSE IF W(R+1,S) THEN 140
130 ON CEIL(RND*3) GOTO 470,490,510
140 IF S#V THEN 160
150 IF FLAG(1) THEN 180 ELSE SFLAG 0 @ GOTO 170
160 IF W(R,S+1) THEN 180
170 ON CEIL(RND*3) GOTO 470,490,540
180 ON CEIL(RND*2) GOTO 470,490
190 IF R=H THEN 250 ELSE IF W(R+1,S) THEN 250
200 IF S#V THEN 220
210 IF FLAG(1) THEN 240 ELSE SFLAG 0 @ GOTO 230
220 IF W(R,S+1) THEN 240
230 ON CEIL(RND*3) GOTO 470,510,540
240 ON CEIL(RND*2) GOTO 470,510
250 IF S#V THEN 270
260 IF FLAG(1) THEN 470 ELSE SFLAG 0 @ GOTO 280
270 IF W(R,S+1) THEN 470
280 ON CEIL(RND*2) GOTO 470,540
290 IF S=1 THEN 400 ELSE IF W(R,S-1) THEN 400
300 IF R=H THEN 360 ELSE IF W(R+1,S) THEN 360
310 IF S#V THEN 330
320 IF FLAG(1) THEN 350 ELSE SFLAG 0 @ GOTO 340
330 IF W(R,S+1) THEN 350
340 ON CEIL(RND*3) GOTO 490,510,540
350 ON CEIL(RND*2) GOTO 490,510
360 IF S#V THEN 380
370 IF FLAG(1) THEN 490 ELSE SFLAG 0 @ GOTO 390
380 IF W(R,S+1) THEN 490
390 ON CEIL(RND*2) GOTO 490,540
400 IF R=H THEN 440 ELSE IF W(R+1,S) THEN 440
410 IF S#V THEN 430
... continued on next page ...
```

```

420 IF FLAG(1) THEN 510 ELSE SFLAG 0 @ GOTO 490
430 IF W(R,S+1) THEN 510
440 IF S#V THEN 460
450 IF FLAG(1) THEN 80 ELSE SFLAG 0 @ GOTO 540
460 IF W(R,S+1) THEN 80 ELSE 540
470 R=R-1 @ L(R,S)=2
480 W(R,S)=C @ C=C+1 @ IF C=T THEN 590 ELSE 100
490 S=S-1 @ L(R,S)=1 @ GOTO 480
510 W(R+1,S)=C @ C=C+1 @ L(R,S)=(L(R,S)#0)+2
530 R=R+1 @ IF C=T THEN 590 ELSE 290
540 IF FLAG(0,0) THEN 570 ELSE W(R,S+1)=C @ C=C+1
550 L(R,S)=2*(L(R,S)#0)+1
560 S=S+1 @ IF C=T THEN 590 ELSE 100
570 SFLAG 1 @ IF L(R,S) THEN L(R,S)=3 @ GOTO 80
580 L(R,S)=1 @ R=1 @ S=1 @ GOTO 90
590 IF NOT FLAG(1,0) THEN 60
600 FOR J=1 TO V @ PRINT "■";
610 FOR I=1 TO H @ IF L(I,J) DIV 2 THEN PRINT " "; ELSE PRINT " ■";
620 NEXT I @ PRINT E$; @ FOR I=1 TO H
630 IF RMD(L(I,J),2) THEN PRINT "■ "; ELSE PRINT "■■";
640 NEXT I @ PRINT "■";E$; @ NEXT J @ PRINT @ CFLAG 0,1 @ END

```

3. SPAGHETTI CODE RULES!

Before you ASSIGN "Titan File" TO *, please ponder one thing. "Structured programming" is all the rage these days. "Spaghetti code" has become an epithet. Are the days forever gone when programmers wrote for computers, not teachers? Is there anyone out there who still cares more about the program being as short and fast as possible, than about it being readable by the Computer Science 101 lab assistant? Yes, we're still here, the die-hards who refuse to sacrifice elegance on the altar of the false god of portability. If it's good enough to want on another system, it's worth rewriting for that system. Ergo SYMBOLIX, the worst case of spaghetti code I've ever seen. It's so ugly it's beautiful. It hasn't got a single module in it; the program pointer jumps around so much it's a wonder that it doesn't fall out. MAZE too is a mess; just read that listing and tell me line-by-line what's happening! Glorious! I'd like to see it in a structured language, like FORTH! Of course, we've seen SYMBOLIX in FORTH; it's called the HP-28. Need I argue further? Ponder it.

Titan File #12 (HPX Exchange, V1 N3, May/June 1987, pages 17-19)

by Joseph K. Horn <53>

Contents:

1. [Apology to Chris Bunsen](#)
2. [Theology ROM](#)
3. [Hard Disk in a Module](#)
4. [Uses of PEEKUTIL](#)
5. [TOKENIZE in one keystroke](#)
6. [LEXCAT lexfile cataloger](#)

1. APOLOGY TO CHRIS BUNSEN

This Titan File must start with an apology to Chris Bunsen, an HP employee who has written some wonderful software for the HP-71. (I mentioned his GRAPH71 in [HPX Exchange V1 N1 P25, Q26](#)). It seems that I obtained a copy of SPLEX from an imprudent source, and believed it to be “freeware”. With this in mind, I ran the LEXSPLIT program on it, to remove the LIST\$ and MODIFY keywords, and thus created the lexfile known as BASICLEX. This has been proliferating among the HP-71 community for almost two years now, and only last Saturday did I learn from Brian that it is in fact “pirated” software.

So besides publicly apologizing to Chris Bunsen, who wrote SPLEX for his HP-71 spreadsheet program, I must also ask all of you kind folks who have a copy of BASICLEX to immediately PURGE it from memory, disk, tape and cards! It does not have the support of the author, to put it mildly. I just finished chatting with him on the phone, and he was relieved to hear that I had not copied it from a purchased copy of his spreadsheet, but still wants the record set straight.

PURGE BASICLEX!!! It's not on the official swapdiscs [actually it is... on SWAP09, SWAP11, JPC02, and LEXFL1], but it may be on one of your copies, so please check.

One reason for killing BASICLEX is that it reportedly contains bugs that SPLEX does not contain. I have not verified this, but no sense tempting fate, so use SPLEX instead of BASICLEX. This is possible because Chris kindly donated SPLEX to the Markov Swapdisc program. You can find it on SWAP07, along with [source code listing disassembled and commented by Mike Markov](#).

[download the plain-text documentation for SPLEX here](#)

[Note added in 2015: SPLEX is available online on the [SWAP/swap07 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy SPLEX directly to your HP-71.]

Final note: until this morning's chat with Chris, he was not aware of the LIST\$ bug mentioned in the CHHU Chronicle and in the [BUFSTO10 documentation on SWAP07](#). The bug works like this. Type 10 DIM A(MEM/16). Then type A\$=LIST\$(10). You will immediately see Memory Lost! The problem is the MEM in the DIMension. In general, any lexfile function (including LEX 01, the built-in lexfile) used in a DIM statement of any kind (like INTEGER, COMPLEX, etc.) when pulled out by the LIST\$ function during an assignment to a string variable, causes Memory Lost. So anything like A\$=LIST\$(10,F\$) when F\$'s line 10 is INTEGER M(MAXRC) will always wipe out. Perhaps Chris will track this bug down and give us a debugged version of SPLEX. If so, it'll be on the next swapdisc!

2. THEOLOGY ROM

As the astute noticed, the Quick Reference Guide in [📖 HP-71 BASIC Made Easy](#) (pg. 135) refers to the CREATE ALL statement in the THEOLOGY ROM. This is just one of the many keywords to be found in this huge lexfile. It is based on a higher source code, which unfortunately this magazine is too small to contain. But I just have to tell you about CREATE ALL; you are sure to get a big bang out of it!

CREATE ALL, as the name implies, creates everything. Since this is such an immense task, it takes about six DAYS (!) to finish, after which the configuration code works for another 24 hours making sure that what was CREATED is indeed good, during which time the HP-71 will appear to have gone into deep sleep. Of course, CREATE ALL only works if ALL does not already exist. If ALL already exists, CREATE ALL gives the "File already exists" error. If this occurs, it isn't enough to PURGE ALL or even DESTROY ALL. The THEOLOGY ROM has an expanded version of END ALL that does the trick. Proper use of END ALL is vital, since further action by users who have ENDED is impossible. The following is suggested: "END ALL @ CREATE ALL" on one line!

If interested in CREATE ALL and END ALL, please look for a subset lexfile called GENESIS on a future swapdisc! But PLEASE use it carefully!

3. HARD DISK IN A MODULE

Looking for the perfect Christmas Stocking-stuffer gift for a loved one who already has an HP-71? Here it is!

Diebstahl Electro-Acoustical Products, Inc (DEAP) has designed the most amazing HP-71 peripheral ever. Thanks to a new, esoteric and mendacious technology, DEAP has managed to fit a complete hard-disk Winchester drive and controller board INSIDE one (1) HP-71 module package!

DEAP, formed in the early 70's by several Triangle Fraternity EE's at Illinois Institute of Technology (IIT), has often been showered with adumbration, but has remained unknown to most OEM's. Now, their new product may obviate facing the problems related to sudden fortune and fame.

Although not yet officially announced, an interview between your Titan File editor and DEAP's hot-line modem and ALPO bulletin board system revealed the following incredible details regarding the new disk module:

The device plugs directly into any HP-71 port and addresses the data in file format just like the 9114 disk drive from HP. Approximately 3142 megabytes (that's over 3KKK) are on-line. Special commands allow storing/recalling files, buffers, alternate operating systems, and digitized images. Rumors that disk access is so fast that files can be run without copying them back into RAM are tantalizing.

The drive was actually a design mistake. DEAP had programmed the IIT computer to generate any conceivable device when given the electronic and packaging parameters. One day they tried to design a large-memory, low-power 3.5-inch Winchester drive, but somebody typed .5 by mistake. To their amazement, the program found a solution and designed the new HP-71 disk drive.

The module is designed with a piggy-back bus. If one is not enough, you can just plug another one into the one already there. Due to the daisy-chain logic, however, access time rapidly degenerates after more than one terabyte is on-line. Also, the battery drain becomes considerable.

One design oddity is that data is not erasable once it has been written to disk. There are two heads: a write head, and a read head. Since they cannot move over each other, the write head is always ahead of the read head, and writes sequentially only. Updating files is done by copying it with the desired revisions to a whole new file. This is not a design mistake; DEAP points out that with over a gigabyte of storage, you could fill 100K a day and still not fill the disk in 83 years. Besides, you can specify which version of a file you wish to access, in case your updating takes a wrong turn.

The storage is so mind-bogglingly capacious that DEAP includes as a free demo file on disk of all the U.S. census data from 1890 to date, and a digitized image of Warnock & Geschke printing out a digitized image of themselves on an HP LaserJet from an HP Vectra, via Postscript. Although it is not mentioned in the manual, prototypes have also been found to contain every HP-65 program in the Users' Library, plus an HP-67 program that translates them into HP-67 code so that HP-71 Translator Pac owners can run them. The technical-minded reader may be interested to know that the pack is formatted into 10,606 cylinders of 296,209 bytes each.

The secret to the bit density and miniature package is the low flying height of the disk head, which is measured in thousandths of an angstrom. Since this would cause problems with large air molecules (carbon monoxide is okay, but carbon dioxide is too big and can cause misreads), the module is permanently sealed in an almost perfect vacuum. This design, called Micro-Angstrom-Gap Go-One-Time technology, or MAGGOT for short, is smaller and more expandable than WORM drives.

Functions that allow the 9114 disk drive to interface directly with the MAGGOT have been provided for making backups. Although a full MAGGOT would fill more 3.5-inch floppies (sometimes called "stiffies") than HP could ship in a century, this says less about the MAGGOT's or 9114's capacity than it does about HP's famous shipping schedule.

DEAP is currently designing an add-on that allows disk dumps at 18,000 baud to videocassette systems. Under consideration is an interface to a print-image microfilmer, for those who would like their HP-71 to run one of these \$75,000 devices.

Only one problem prevents the MAGGOT from being released to the market. Relativity Theory states that rotation and acceleration are non-uniform motions. DEAP sidestepped this problem by uniformly accelerating the rotation. Eventually the near-light-speed velocity of the disk pack creates a strong gyroscopic effect, so much so that any motion of the HP-71 with any angular differential to the plane of rotation (such as lifting the HP-71 from the desk, or bumping the desk, or even pressing any HP-71 key) instantly results in a head crash, which in turn emits an intense stream of sub-atomic particles from friction fission. DEAP does not like to study this phenomenon, however, because every HP-71 that has run afoul this way was instantly reduced to plasma, rendering the date code unreadable and thus voiding the warranty.

As soon as this minor bug is exorcised, DEAP promises to announce the MAGGOT hard disk module to the press, at a price based on the 9114's list price times the storage capacity difference. At that price, I hope it comes with a copy of the THEOLOGY ROM!

4. USES OF PEEKUTIL

If you'd like to go spelunking through HP-71 files, but aren't sure what's where, here's everything you need to know, all in one handy place. Keep this page marked for future reference, or make a photocopy of it.

You'll need a copy of PEEKUTIL, written by Flavio Casetta. You can find it on Markov's swapdisks (CHHU02 and CHHU06) or you can get it by sending me one blank mag card and a self-addressed

stamped envelope. If you write PEEKUTIL on the card, no further note will be necessary. If you don't mind typing the whole lexfile into the MAKELEX program, you can find PEEKUTIL listed in the CHHU Chronicle, V2 N7 P13.

[Note added in 2015: PEEKUTIL is available online on the [📁 SWAP/chhu06 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy PEEKUTIL directly to your HP-71.]

First, assuming the file we wish to explore is called FRED, then do this:

```
A=HTD(ADDR$("FRED"))
```

This puts FRED's address into A. If you change memory in any way that might move FRED, be sure to recalculate A in the way shown above.

Now, for any filetype at all:

```
TEXT$(A,8) = "FRED      " (full 8-character file name)
RPEEK$(A+16,4) = FRED's filetype in hex
RED(PEEK(A+16,4),2^15) = FRED's filetype in signed decimal
PEEK(A+20) = FRED's file protection as follows:
  0 = no protection
  1 = SECURE only
  2 = PRIVATE only
  3 = EXECUTE ONLY (both SECURE and PRIVATE)
RPEEK$(A+22,4) = FRED's creation time in HHMM format
RPEEK$(A+26,6) = FRED's creation date in YYMMDD format
PEEK(A+32,5)+32 = FRED's total length in nibbles
ADPEEK$(A+32,5) = Hex address of file after FRED in CATalog
ADPEEK(A+32,5) = Decimal address of file after FRED in CATalog
```

If FRED is an SDATA file

```
PEEK(A+32,5) DIV 16 = number of records in FRED
RPEEK$(A+37+16*N,16) = Contents of Nth record (record 0 is first)
```

If FRED is a DATA file

```
PEEK(A+37,4) = Number of records in FRED
PEEK(A+41,4) = Number of bytes in each record in FRED
```

If FRED is a LEX file

```
PEEK(a+37,2) = FRED's lex ID in decimal
RPEEK$(A+37,2) = FRED's lex ID in hex
PEEK(A+39,2) = FRED's lowest token number in decimal
RPEEK$(A+39,2) = FRED's lowest token number in hex
PEEK(A+41,2) = FRED's highest token number in decimal
RPEEK$(A+41,2) = FRED's highest token number in hex
PEEK(A+43,5) = Distance to next linked lex table (0 if none)
  If that's not 0, then:
  ADPEEK(A+43,5) = decimal address of next linked lex table
  ADPEEK$(A+43,5) = hex address of next linked lex table
```

(NB: Linked lex tables are all within one lexfile)
 PEEK(A+48) = Speed Table existence flag (15 if none)
 If that's 0, then LET A=A+79 to skip over speed table
 PEEK(A+49,4) = Distance to Text Table (0 if none)
 If that's not 0, then:
 ADPEEK(A+49,4,-1) = decimal address of text table
 ADPEEK\$(A+49,4,-1) = hex address of text table
 PEEK(A+53,4) = Distance to Message Table (0 if none)
 If that's not 0, then:
 ADPEEK(A+53,4) = decimal address of message table
 ADPEEK\$(A+53,4) = hex address of message table
 PEEK(A+57,5) = Distance to Poll Handler (0 if none)
 If that's not 0, then:
 ADPEEK(A+57,5) = decimal address of poll handler
 ADPEEK\$(A+57,5) = hex address of poll handler
 A+62 = decimal address of Main Table

If FRED is a BASIC file

RPEEK\$(A+37,5) = Hex distance to the first SUB or END SUB
 If 00000, then RUN or RENUMBER and try again
 If FFFFF, then there are no SUBs nor END SUBs
 RPEEK\$(A+42,5) = Hex distance to first label or DEF FN
 If 00000, then RUN or RENUMBER and try again
 If FFFFF, then there are no labels nor DEF FNs
 RPEEK\$(A+47,2) = "F0", the End-Of-Line (EOL) marker
 VAL(RPEEK\$(A+49,4)) = first line number
 PEEK(A+53,2)-2 = length of first statement in nibbles
 If you LET L=PEEK(A+53,2)-2, then:
 PEEK\$(A+55,L) = hex contents of first statement
 RPEEK\$(A+55+L,2) = EOL or @
 If F0, it's an EOL followed by another line number
 If F4, it's an @ followed by another length byte
 and so on, until the end of the file is reached.

With these tools, you can explore any type of file. A few ideas that came from this appear below.

5. TOKENIZE in one keystroke

In the [CHHU Chronicle, V1 N3 P29](#), a program called TOKENIZE was listed, which enabled you to see how BASIC changes keywords into hex tokens internally. Here is the same idea, but on a simple key definition instead of a whole program:

```
KEY 'f4', 'J=PEEK(193885,5)@I=PEEK(J+53,2)-2@PEEK$(J+55,I);"(";STR$(I);)"':
```

After making this key assignment, merely pressing [f][4] (the SIN key) will show the tokenized version of the first statement of the current BASIC file, no matter what it is. This is followed by a nibble count, in parentheses. This is useful in comparing the byte length of different approaches to the same task.

6. LEXCAT lexfile cataloger

RUN LEXCAT (or CALL LEXCAT(F\$) with a filename in F\$), and see the hex address of the specified lexfile, the address of its poll handler(s), the address of the next file, and all the keywords in the lexfile with their execution code address (entry point), lex ID and token number, and function syntax. This is an invaluable program for those of us who use lexfiles a lot. It is based on a program by HP, with a few more bells and whistles, cleaner output, and much faster thanks to PEEKUTIL.

```
LEXCAT
-----
5 ! Requires PEEKUTIL & HPILROM
10 INPUT 'LEX File: ';F$ @ CALL LEXCAT(F$)
20 SUB LEXCAT(F$) @ ON ERROR GOTO 290 @ A1=HTD(ADDR$(F$))
30 IF PEEK(A1+16,2)#8 THEN DISP "ERR:Not LEX file" @ END
40 S=A1+37 @ F$=UPRC$(F$)
50 C=S+25+NOT PEEK(S+11)*79 @ T$=DTH$(C)&RPEEK$(S,6)
80 IF LEN(F$) THEN PRINT @ PRINT "*** ";F$;" ***" @
  PRINT "(";DTH$(A1);") File Header"
90 M=HTD(T$[8,9]) @ B=ADPEEK(C-13,4)
100 IF PEEK(C-5,5) THEN PRINT "(";ADPEEK$(C-5,5);") Poll Handler"
120 IF PEEK(B,2)=255 THEN 300
130 L=PEEK(B-1)+1 @ W$=TEXT$(B,L/2) @ S$=RPEEK$(B+L,2) @ D=C+(HTD(S$)-M)*9
140 PRINT "(";ADPEEK$(D+3,5);") ";T$[10];"/"; @ B=B+L+3
150 IF NOT HTD(S$) THEN PRINT "-- ";W$;" ffn" @ GOTO 110
160 PRINT S$;" ";W$; @ E=PEEK(D+8) @ IF NOT E THEN PRINT " postfix" @
  GOTO 110
170 IF E#15 THEN 250
180 E=RMD(ADPEEK(D+3,5,-2),1048576) @ P=PEEK(E) @ Q=PEEK(E+1)
190 IF Q THEN PRINT "("; ELSE PRINT " fn" @ GOTO 110
200 FOR I=1 TO Q @ IF I#1 THEN PRINT ",";
210 E=E-1 @ X=PEEK(E) @ IF I>P THEN PRINT "?";
220 IF X>11 THEN PRINT "#/$"; ELSE IF X>7 THEN PRINT "#"; ELSE PRINT "$";
230 IF RMD(X,4) THEN PRINT "(";
240 NEXT I @ PRINT ")" @ GOTO 110
250 IF NOT BIT(E,3) THEN PRINT " *P";
260 IF NOT BIT(E,2) THEN PRINT " *I";
270 IF NOT BIT(E,0) THEN PRINT " *K";
280 PRINT @ GOTO 110
290 DISP "ERR: ";ERRM$ @ BEEP @ END
300 IF KEY$="#38" THEN END
310 IF NOT HTD(T$[10]) THEN END
330 IF PEEK(S+6,5) THEN S=ADPEEK(S+6,5) @ F$="" @ GOTO 50
340 DISP "(";ADPEEK$(A1+32,5);") Next File Header"
350 END SUB
```

If run on SPLEX, you'll see (with varying addresses):

```
*** SPLEX ***
(80008) File Header
(80077) Poll Handler
(800D5) 52/24 LIST$(#,$)
```

(80281) 52/23 MODIFY
(808C4) Poll Handler
(80900) 53/55 MAXRC fn
(809F1) Next File Header

which means SPLEX's file header begins at hex address 80008; its first poll handler begins execution at hex 80077; the LIST\$ function starts execution at hex address 800D5, it has lex ID 52 (hex), token 24 (hex), and takes one numeric argument, followed by an optional string argument; the MODIFY keyword is a statement (no clue to its syntax!); there's another poll handler at 808C4; MAXRC is a function with no arguments; and the file ends at 809F0.

Titan File #13 (HPX Exchange, V1 N4, Jul 87 – Mar 88, pages 5-7)

by Joseph K. Horn <53>

Contents:

1. [A^B \(mod C\)](#)
2. [Decimal-to-Fraction](#)
3. [SYMBOLIX Update](#)
4. [Multiple-Column Text File Lister](#)
5. [PUZZLER – The Program That Think!](#)

1. A^B (mod C)

If A, B and C are small enough, you can find $\text{MOD}(A^B, C)$ by typing it. But if A^B is greater than $1E12$, accuracy is lost as digits fall off the end (round-off error). Here's a routine that finds $A^B \pmod C$ even when A^B is huge. It uses some rules of number theory to find the answer in "chunks", and even lets you know if the answer is exact or not.

```
10 INPUT "A^B mod C: A,B,C? ";A,B,C
20 CALL PMOD(A,B,C,P)
30 IF FLAG(INX) THEN DISP "Too Big" ELSE DISP P
40 SUB PMOD(A,B,C,P) @ P=1 @ A=RMD(A,C) @ CFLAG INX
50 IF A=1 OR FLAG(INX) THEN 70 ELSE IF RMD(B,2) THEN P=RMD(P*A,C)
60 IF NOT P THEN 70 ELSE B=B DIV 2 @ IF B THEN A=RMD(A*A,C) @ GOTO 50
70 END SUB
```

To use, Just input A, B and C. $A^B \pmod C$ will be displayed. If the program at any point loses digits due to round-off error, the program will halt with "Too Big" displayed.

Lines 40 through 70 are a subprogram, the heart of the program. It can be CALLED from any other program. CALL PMOD(A,B,C,P) with A, B and C as desired, and P a numeric variable; $A^B \pmod C$ will be returned in P, and FLAG(INX) will be clear if P is exact.

Examples: What is $37^{41} \pmod{21}$? Since 37^{41} is bigger than $1E12$, you don't get the right answer by typing $\text{MOD}(37^{41}, 21)$, which gives 5. The correct answer is 4. RUN the program, input 37,41,21 and see 4.

Try a more outrageous example: what's $\text{MOD}(12345^{98765}, 13579)$? You can't even begin to use MOD, because 12345^{98765} overflows! RUN the program, input 12345,98765,13579 and see the answer: 3054.

What are the last 6 digits of $123456789^{987654321}$? Input 123456789, 987654321, 1E6. See the answer: 933589.

In general, the program as written can handle any A or B, but C should not be larger than $1E6$. This routine's worst case runtime is just a few seconds. Bob Wilson, who sent me the number theory foundations of this routine, has rewritten it to work for any C, but it can take a long time to run. Real-world applications? Maybe. Cryptanalysts and number theorists enjoy working with large primes. Determining whether a large number is prime or composite is most easily done not by factoring but by applying Fermat's theorem. The above routine allows some simple experimentation with Fermat's theorem.

2. DECIMAL-TO-FRACTION CONVERSION

Many programs return answers in decimal form, like 2.390625, but it would be nice to know what fraction (A/B) is equal to that decimal mess.

Here is the fastest I've been able to cook up. In your main program, just CALL D2F(F,E,A,B) where F is the decimal fraction, E is how many digits of accuracy you want (1 through 12), and A & B are numeric variables. The result is returned in A and B.

```
10 SUB D2F(F1,E,A,B) @ F=F1 @ A1=1 @ B0=1 @ E=10^(-E)
20 C=INT(F) @ A=A1*C+A0 @ B=B1*C+B0 @ Q=A/B
30 IF ABS(Q-F1)<=E OR C=F THEN END
40 F=1/(F-C) @ A0=A1 @ A1=A @ B0=B1 @ B1=B @ GOTO 20
```

Example: What fraction is 2.390625 equal to? CALL D2F(2.390625,6,A,B) and then DISP A;B. It's 153/64. We used 6 digits of accuracy because that's all the digits of the decimal fraction we had. If you wish to force D2F to give an approximation instead of the real answer, then specify less digits. For example, what fraction approximates PI to 8 digits? CALL D2F(PI,7,A,B) and then DISP A; B. 103993/33102 is a good approximation of PI. I only use D2F as a subprogram, but you may wish to add a main program above it to automate the input and output.

3. SYMBOLIX UPDATE

FIRST: A program called SYMBOLIX was described in [an earlier Titan File](#) that claimed to do symbolic math (algebra & calculus) better than the HP-28. I would like to almost retract that statement. At the time, only the HP-28C existed, which was a machine that drove me nuts. SYMBOLIX does beat the 28C, as far as I'm concerned.

But meanwhile, they've come out with the HP-28S, which is almost the same machine but so much improved that I must admit I like it. The problems with the 28C were ridiculously little memory and no real way to control the menus from a program. The 28S solves these handsomely, and throws in a few new handy features as well.

All of the symbolic math problems that SYMBOLIX can do which the 28C couldn't, the 28S CAN! And, I'm sorry to tell, the 28S always seems to beat SYMBOLIX in a race, too... and by orders of magnitude. Oh, well; SYMBOLIX is still fun to play with. And its price CANNOT be beat!

SECOND: SYMBOLIX claimed to work in RADIANS mode. It also was touted as being unlistable since the line numbers were removed. Several folks noticed that certain runs errored out in RADIANS mode, so they went to the bother of POKEing the line numbers back in (*ghah*), in order to find the bug, and to their surprise, the bug then vanished! It turns out that we discovered a new HP-71 bug that HP didn't even know about: GOTO and GOSUB always recompiles when in RADIANS mode. So the missing line numbers caused the line-number search routine to fail, hence the error.

Due to the necessity of having the line numbers for SYMBOLIX to work correctly, and after corresponding with those concerned, we've decided to release SYMBOLIX in totally public-domain form. It is now un-PRIVATE. It now has line numbers (on every line!). It doesn't have any unlistably long lines any more. And the synthetix were all taken out (e.g. DESTROY LIST). All I ask is that you keep the first two lines (copyright & credits) intact. For a printed listing, send a business-

size envelope with your name and two stamps on it; it's over 12 pages long.

[Note added in 2015: SYMBOLIX is available online on the [📁 SWAP/swap11 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy SYMBOLIX directly to your HP-71. Its documentation is here: [📄 DOC](#)]

4. MULTIPLE-COLUMN TEXT FILE LISTER

When using TEXTfiles to store lots of little items, such as file names, words to be alphabetized, etc., it is a hassle to print the file out because you get one long, skinny column and tons of blank paper. What we need is a routine to print out TEXT files in multiple columns. Here 'tis.

```
10 ! COLUMNS ! Lists TEXT files in columns ! Joseph K. Horn ! 24 Mar 88
11 ! Uses FILESZR function (in EDLEX, TEXTUTIL, etc.)
20 CALL COLUMNS @ SUB COLUMNS @ DIM A$[132],F$[20]
30 INPUT "Filename? ";F$ @ UNSECURE F$ @ ASSIGN #1 TO F$
40 INPUT "Columns? ", "4";C @ INPUT "PWIDTH ", "80";W @ W=W DIV C
50 F=FILESZR(F$) @ L=F DIV C+NOT NOT RMD(F,C) @ F=F-1
60 FOR X=0 TO L-1 @ FOR Y=X TO F STEP L @ READ #1,Y;A$
70 A$[W]=" " @ PRINT A$; @ NEXT Y @ PRINT @ NEXT X @ END
```

Just RUN it, specify the name of the file to be listed, the number of columns desired, and the printer width. The NOT NOT in line 50 is an elegant trick.

5. PUZZLER – The Program That Thinx!

According to INSIGHT magazine, the only way we can program artificial intelligence is with a parallel-processing mainframe like The Connection Machine. Not even a Cray can hack it, they said. What would they say about a handheld that thinx? Imagine your HP-71 solving the following puzzler, given only the English clues...

Billy, Willy, Gerald and Oliver live (not respectively*) in a Blue house, White house, Green house and Orange house, and have as pets (not respectively*) a Bear, Walrus, Giraffe and Orangutan. Billy never saw the walrus that lives in the green house. Gerald never saw the bear. Who lives where with what pet?

* Note added in 2015: "Not respectively" here means that nobody's name, house color, or pet, begin with the same letter.

This type of "logic puzzle" is designed to exercise the mind and be a pleasant way to kill some time. Solving them on a machine that does all the thinking for you is certainly missing the point! But then, there are many crossword puzzle addicts who use these new Franklin dictionary machines to help with their puzzles. So why not?

The program listed below, called PUZZLER, solves any logic puzzle similar to the Billy, Willy, Gerald and Oliver puzzle above. It even solves [the famous "Who Owns the Zebra?" puzzle](#).

PUZZLER BASIC 1768 bytes

```
1 ! PUZZLER ! The Program that Thinx ! Joseph K. Horn ! Oct 1987
2 ! Solves "Who Owns the Zebra"-type logic puzzles ! Uses no lexfiles
10 CALL PUZZLER @ SUB PUZZLER @ OPTION BASE 1 @ STD
20 INPUT "How many groups? ";A
30 INPUT "Items per group? ";B
40 C=A*B @ D=C*(A-1)/2 @ INTEGER M(C,C),Q(D),R(D) @ DIM N$(C)[20],A$[96]
50 FOR J=1 TO C STEP B @ FOR K=J TO J+B-1 @ FOR L=J TO J+B-1
60 M(K,L)=-1 @ NEXT L @ M(K,K)=1 @ NEXT K @ NEXT J
70 INPUT "Auto/Quick? ", "A";A$ @ X=FLAG(3,UPRC$(A$)[1,1]="A")
80 FOR J=1 TO A @ FOR K=1 TO B @ DISP "Group";J;"item ";STR$(K);
90 INPUT ": ",CHR$(64+J)&STR$(K);A$ @ N$((J-1)*B+K)=UPRC$(A$) @
    NEXT K @ NEXT J
100 INPUT "Clue? ";A$ @ A$=" "&UPRC$(A$)&" "
110 IF POS(A$,"THINK") THEN 260
120 K=0 @ L=0 @ FOR N=1 TO C
130 V=POS(A$," "&N$(N)&"")+POS(A$," "&N$(N)&" 'S")+POS(A$," "&N$(N)&"S ")
140 IF NOT V THEN 150 ELSE A$[V,V+LEN(N$(N))]=" " @ IF K THEN L=N ELSE K=N
150 NEXT N @ T=1 @ IF POS(A$," NOT ") OR POS(A$,"N'T ") THEN T=-1
160 IF POS(A$," NEVER ") OR POS(A$," - ") OR POS(A$," NO ") THEN T=-1
170 IF NOT (K AND L) THEN DISP "What???" @ BEEP @ BEEP @ WAIT 2 @ GOTO 100
180 IF POS(A$,"?") THEN 490
190 IF NOT M(K,L) THEN 220
200 IF M(K,L)=T THEN DISP "Right!" ELSE DISP "Wrong!"
210 BEEP @ BEEP @ WAIT 2 @ GOTO 100
220 M(K,L)=T @ M(L,K)=T
230 IF T=1 THEN P=P+1 @ Q(P)=K @ R(P)=L
240 IF P=D THEN 450
250 IF NOT FLAG(3) THEN 100
260 CFLAG 1 @ IF P=0 THEN 350
270 FOR J=P TO 1 STEP -1 @ FOR K=1 TO C
280 IF M(K,Q(J))=M(K,R(J)) THEN 340
290 U=Q(J) @ V=R(J) @ SFLAG 1 @ IF M(K,U) THEN U=V @ V=Q(J)
300 T=M(K,V) @ IF T=1 THEN P=P+1 @ Q(P)=K @ R(P)=U
310 M(K,U)=T @ M(U,K)=T
320 DISP N$(K); @ IF T=1 THEN DISP " + "; ELSE DISP " - ";
330 DISP N$(U) @ BEEP @ IF P=D THEN 450
340 NEXT K @ NEXT J @ IF FLAG(1) THEN 260
350 FOR J=1 TO C STEP B @ FOR K=1 TO C STEP B @ IF J=K THEN 430
360 FOR L=J TO J+B-1 @ T=B @ FOR N=K TO K+B-1 @ T=T+M(L,N)
370 NEXT N @ IF T>1 THEN 420 ELSE SFLAG 1
380 FOR N=K TO C @ IF NOT M(L,N) THEN 400
390 NEXT N
400 M(L,N)=T @ M(N,L)=T @ DISP N$(L);" + ";N$(N) @ BEEP
410 P=P+1 @ Q(P)=L @ R(P)=N @ IF P=D THEN 450
420 NEXT L
430 NEXT K @ NEXT J
440 IF FLAG(1) THEN 260 ELSE 100
450 DISP "THE PUZZLE IS SOLVED!" @ BEEP 800,.5 @ WAIT 2
460 FOR J=1 TO B @ DISP N$(J);
470 FOR K=B+1 TO C @ IF M(J,K)=1 THEN DISP " + ";N$(K);
480 NEXT K @ DISP @ NEXT J @ CFLAG 1,3,4 @ END
... continued on next page ...
```

```

490 FOR K=K-RMD(K-1,B) TO K+B-1 @ DISP N$(K);
500 FOR L1=L-RMD(L-1,B) TO L1+B-1 @ IF NOT M(K,L1) THEN 530
510 IF M(K,L1)=1 THEN DISP " + "; ELSE DISP " - ";
520 DISP N$(L1);
530 NEXT L1 @ DISP @ NEXT K @ GOTO 100

```

RUN PUZZLER and input the number of groups and the number of items per group. In the Billy, Willy, Gerald and Oliver puzzle, there are three groups (men, houses, pets) and four items per group (B, W, G, O).

PUZZLER will ask if you prefer **Auto** or **Quick** processing. **Auto** mode means that PUZZLER thinks about every input after each one is typed in. This mode gives the impression of artificial intelligence the best; it seems to be thinking all the time. **Quick** mode means that PUZZLER lets you input the clues quickly, and then type **THINK** to force it to “think” about the clues. This mode is practical when there is a large number of groups and items.

Then input names for every item, as indicated. If you wish, you may use the spreadsheet-style names that are automatically supplied; or type your own names over them. We’ll use **BILLY**, **WILLY**, **GERALD**, **OLIVER**, **BLUE**, **WHITE**, **GREEN**, **ORANGE**, **BEAR**, **WALRUS**, **GIRAFFE** and **ORANGUTAN** as our inputs for the sample puzzle.

Then input the clues. You can usually type the clues in English directly, but sometimes you must do a little simplification so that PUZZLER won’t get puzzled. You must use the names that you supplied. PUZZLER looks for only logical “connections” or “disconnections”. The following are all equivalent statements:

```

BILLY NEVER SAW THE WALRUS
BILLY DOESN'T OWN THE WALRUS
BILLY DOES NOT OWN THE WALRUS
BILLY NOT WALRUS
BILLY - WALRUS

```

The presence of “NEVER” or “N'T” or “NOT” or “-” *anywhere in the input* [in red above] tells PUZZLER that the two items mentioned *anywhere in the input* [in blue above] are *not* related. Everything else in the input [in black above] is ignored.

And these are all equivalent:

```

THE WALRUS LIVES IN THE GREEN HOUSE
THE WALRUS IS IN THE GREEN HOUSE
WALRUS IS GREEN
WALRUS + GREEN
WALRUS GREEN

```

The *absence* of “NEVER” or “N'T” or “NOT” or “-” indicates that the two items mentioned *anywhere in the input* [in blue above] *are* related. Everything else in the input [in black above] is ignored.

Of course, we must input the “not respectively” clues:

```

BILLY - BLUE
BILLY - BEAR
BEAR - BLUE
WILLY - WHITE
etc.

```

Keep inputting clues until PUZZLER discovers the answer and prints it out.

Whenever PUZZLER discovers a logical connection or disconnection on its own, it prints it. If you don't have a video connected, or a printer acting as a video (DISPLAY IS PRINTER), then you'd better set the DELAY to something long enough to be useful, or change the program to pause after each display.

If you wish to examine the known relationships between two items, input them with a "?" between, like this:

BILLY ? BEAR

This will result in all the relations between *all the items in those two groups* to be listed. This is useful when you are solving a puzzle with clues like "The Englishman lives next door to the man who smokes Chesterfields."

The answer to the Billy puzzle is... well, I'll let you and your HP-71 figure it out.

Artificial intelligence? No, not really, but it IS intriguing.

For your entertainment, here's the puzzle that started this craze.

Who Owns The Zebra?

1. There are five houses.
2. The Englishman lives in the red house.
3. The Spaniard owns the dog.
4. Coffee is drunk in the green house.
5. The Ukrainian drinks tea.
6. The green house is immediately to the right of the ivory house.
7. The Old Gold smoker owns snails.
8. Kools are smoked in the yellow house.
9. Milk is drunk in the middle house.
10. The Norwegian lives in the first house.
11. The man who smokes Chesterfields lives in the house next to the man with the fox.
12. Kools are smoked in the house next to the house where the horse is kept.
13. The Lucky Strike smoker drinks orange juice.
14. The Japanese smokes Parliaments.
15. The Norwegian lives next to the blue house.

Now, who drinks water? And who owns the zebra?

In the interest of clarity, it must be added that each of the five houses is painted a different color, and their inhabitants are of different national extractions, own different pets, drink different beverages and smoke different brands of American cigarets [sic]. One other thing: in statement 6, *right* means *your right*.

First published in Life International Magazine on December 17, 1962

Titan File #14 (HPX Exchange, V1 N5, Apr-Jul 1988, pages 10-12)

by Joseph K. Horn <53>

Contents:

1. [Fraction Calculator](#)
2. [Complex Math without the Math ROM](#)
3. [Bulk File Transfer with ROMCOPY](#)
4. [Relocate](#)
5. [Lexfile Enable/Disable in BASIC](#)
6. [SORTLEX!](#)

1. HP-71 Fraction Calculator

The following program was written by request. Several readers asked for an HP-71 version of the HP-28 fraction package in the last issue (V1 N4 P7, left column). If you ever need to do quick keyboard fraction addition, subtraction, multiplication or division, here's your program:

[Note added in 2015: An improved version of FRAX, which allows decimal inputs and adds the power function (^), is available online on the [📁 HORN/MAIN03 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy FRAX directly to your HP-71.]

```
1 ! FRAX ! Fraction Calculator
2 ! by Joseph K. Horn ! 17 June 1988
10 N=0 @ D=1 @ STD
20 IF D=0 THEN STOP ELSE X=N @ Y=D
30 G=MOD(X,Y) @ X=Y @ Y=G @ IF G THEN 30 ELSE N=N/X @ D=D/X
40 IF N>=1.E+12 OR D>=1.E+12 THEN DISP "Overflow!" @ STOP
50 DISP STR$(N);",";STR$(D);
60 A$=KEY$ @ IF NOT LEN(A$) THEN 60
70 K=POS("-+/*.") @ IF K=5 THEN 160
80 IF K THEN DISP " ";A$;" "; ELSE DISP @ PUT A$
90 ON ERROR GOTO 95 @ INPUT "";X,Y @ GOTO 100
95 IF ERRN=40 THEN Y=1 ELSE 90
100 OFF ERROR @ ON K+1 GOTO 110,120,130,140,150
110 N=X @ D=Y @ GOTO 20
120 X=-X
130 N=Y*N+X*D @ D=Y*D @ GOTO 20
140 Q=X @ X=Y @ Y=Q
150 N=N*X @ D=D*Y @ GOTO 20
160 DISP @ DISP "Done" @ END
```

Input fractions this way: Numerator,Denominator [ENDLINE]

Input whole numbers as: Number [ENDLINE]

Press +, -, * or / after [ENDLINE] to begin a calculation.

Inputs and results are displayed as fully reduced fractions.

Press [.] to exit the program.

Example: What's 153/684 in reduced form?

Press: 153,684 [ENDLINE]

Answer: 17/76.

Example: What's $5/6 * 8/5$?
Press: 5,6 [ENDLINE] * 8,5 [ENDLINE]
Answer: 4/3.

Example: What's $12/34 - 56/78 + 7/221$?
Press: 12,34 [ENDLINE] - 56,78 [ENDLINE] + 7,221 [ENDLINE]
Answer: -1/3.

Example: What's $987/321 - 3$?
Press: 987,321 [ENDLINE] - 3 [ENDLINE]
Answer: 8/107.

Notice that there is no need to "clear" the fraction calculator. To begin a new calculation, just type it. Press [.] to exit the program.

If you have access to the KEYWAIT\$ function, you may replace the entire line 60 with A\$=KEYWAIT\$. This will lower battery drain considerably.

2. Complex Math without the Math ROM

A big chunk of the code in the HP-71 MATH ROM is devoted to performing complex math, that is, math on "complex numbers". A complex number is usually written like $3+4i$ (where "i" is loosely defined as the square root of -1).

For those who don't own a MATH ROM, and for those who own one and would like to see what's going on inside it, here are vanilla-BASIC subprograms that perform the MATH ROM complex functions.

To use, just CALL the subprogram. The parameters are the real and imaginary part of the complex number(s), followed by two variables that will be set to the real and imaginary part of the answer. The only exception to this is SUB CINV(a,b) which reciprocates $a+bi$ in place. Examples of all of them in use follow the listing. Here is one to give you the feeling:

Example: What is $(3+4i)*(5-6i)$?
Press: CALL CMULT(3,4,5,-6,a,b) @ a;b [ENDLINE]
See: 39 2
Answer: 39+2i.
Math ROM method: $(3,4)*(5,-6)$ [ENDLINE] and see (39,2).

```
10 SUB POLAR(X,Y,R,A) @ RADIANS @ A=ANGLE(X,Y) @ R=SQR(X*X+Y*Y)
20 SUB RECT(R,A,X,Y) @ RADIANS @ X=R*COS(A) @ Y=R*SIN(A)
30 SUB CADD(R1,I1,R2,I2,R,I) @ R=R1+R2 @ I=I1+I2
40 SUB CSUB(R1,I1,R2,I2,R,I) @ R=R1-R2 @ I=I1-I2
50 SUB CMULT(R1,I1,R2,I2,R,I) @ R=R1*R2-I1*I2 @ I=R1*I2+R2*I1
60 SUB CINV(R,I) @ X=R*R+I*I @ R=R/X @ I=-I/X
70 SUB CDIV(R1,I1,R2,I2,R,I) @ CALL CINV(R2,I2) @ CALL CMULT(R1,I1,R2,I2,R,I)
80 SUB CSQRD(R1,I1,R,I) @ R=R1*R1-I1*I1 @ I=2*R1*I1
90 SUB CABS(R1,I1,R,I) @ R=SQR(R1*R1+I1*I1) @ I=0
... continued on next page ...
```

```

100 SUB CEXP(R1,I1,R,I) @ RADIANS @ E=EXP(R1) @ R=E*COS(I1) @ I=E*SIN(I1)
110 SUB CSQRT(R1,I1,R,I) @ CALL POLAR(R1,I1,R0,A0)
120 CALL RECT(SQR(R0),A0/2,R,I)
130 SUB CALOG(R1,I1,R,I) @ CALL CEXP(R1*LN(10),I1*LN(10),R,I)
140 SUB CLN(R1,I1,R,I) @ CALL POLAR(R1,I1,R0,I) @ R=LN(R0)
150 SUB CLOG(R1,I1,R,I) @ CALL CLN(R1,I1,R,I) @ X=LN(10) @ R=R/X @ I=I/X
160 SUB C2R(R1,I1,R2,R,I) @ CALL POLAR(R1,I1,R0,A0)
170 CALL RECT(R0^R2,A0*R2,R,I)
180 SUB C2C(R1,I1,R2,I2,R,I) @ CALL CLN(R1,I1,X,Y)
190 CALL CMULT(R2,I2,X,Y,R1,I1) @ CALL CEXP(R1,I1,R,I)

```

Note! In all these subprograms, all angles are in radians!
The “C” before the subprogram name stands for “Complex”.

Examples:

1. CALL POLAR(3,4,r,a): converts rectangular (3,4) to polar (5,.9273).
2. CALL RECT(306,PI/3): converts polar (306,PI/3) to rectangular (153,265.0038).
3. CALL CADD(3,4,5,6,r,i): Adds (3+4i)+(5+6i) and gets 8+10i.
4. CALL CSUB(3,4,5,6,r,i): Subtracts (3+4i)-(5+6i) and gets -2-2i.
5. CALL CMULT(3,4,5,6,r,i): Multiplies (3+4i)*(5+6i) and gets -9+38i.
6. r=3 @ i=4 @ CALL CINV(r,i): Reciprocates 3+4i and gets .12-.16i. (CINV is the only subprogram that changes the values sent.)
7. CALL CDIV(5,6,3,4,r,i): Divides 5+6i by 3+4i and gets 1.56-.08i.
8. CALL CSQRD(3,4,r,i): Finds (3+4i)² and gets -7+24i.
9. CALL CABS(3,4,r,i): Finds ABS(3+4i) and gets 5.
10. CALL CEXP(3,4,r,i): Finds EXP(3+4i) and gets -13.1288-15.2008i.
11. CALL CSQRT(3,4,r,i): Finds SQRT(3+4i) and gets 2+i.
12. CALL CALOG(3,4,r,i): Finds 10⁽³⁺⁴ⁱ⁾ and gets -977.0962+212.7979i.
13. CALL CLN(3,4,r,i): Finds LN(3+4i) and gets 1.6094+.9273i.
14. CALL CLOG(3,4,r,i): Finds LOG₁₀(3+4i) and gets .699+.4027i.
15. CALL C2R(3,4,5,r,i): Raises (3+4i)⁵ and gets -237-3116i.
(C2R means Complex-To-Real-power).
16. CALL C2C(3,4,5,6,r,i): Raises (3+4i)⁽⁵⁺⁶ⁱ⁾ and gets -1.8609+11.8368i.
(C2C means Complex-To-Complex-power)

3. Bulk File Transfer with ROMCOPY

Ever want to copy more than one file to/from disk at a time? Who hasn't! There is a handy way, without using arcane archiving protocols or packing techniques.

HP developed a lexfile called ROMCOPY many years ago to assist software developers in making ROM images of their wares. You can find ROMCOPY on the SWAPDISKS. It can serve us normal HP-71 users too!

[Note added in 2015: ROMCOPY is available online on the [📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy ROMCOPY directly to your HP-71.]

ROMCOPY takes an entire IRAM (Independent RAM; FREEd PORT) and copies its contents onto disk (or tape) under a single filename. Then, it can later copy that file back into the IRAM anytime. This

allows us to make several ROMCOPYs on disk, and load them into IRAM as needed.

Let's suppose you've typed `FREE PORT(0)` on your unmodified HP-71. This created a 4K IRAM. Then you've filled it with files of all sorts. To save all those files on disk (or tape) in one step, under the name of, say, `PORT0`, all you need to do is type:

```
ROMCOPY :PORT(0) TO PORT0:TAPE
```

(Note: the `:TAPE` device specifier works equally well for the tape drive and the disk drive.) Now, to copy that file back into `PORT(0)` any time, just:

```
ROMCOPY PORT0:TAPE TO :PORT(0)
```

There are other features of ROMCOPY that are detailed in [📖 HP's documentation](#).

Another application: recovery from those severe "Memory Lost"s when IRAMs are lost too.

4. RELOCATE

Add this subprogram to your subprogram library, making sure that you type "00" as "oh zero":

```
10 SUB RELOCATE @ INPUT "RELOCATE ";F$
20 ON ERROR GOTO 30 @ COPY F$ TO 00 @ PURGE F$ @ RENAME 00 TO F$
30 END SUB
```

Then KEY "f3", "CALL RELOCATE": (of course, you may use any key other than f3). Pressing [f][3] will seem to activate a new RELOCATE command. Type any filename, and that file will be whisked from its current location in `CAT ALL` to the last position.

This is useful for two reasons. (1) Lexfiles often conflict. Making sure that the proper lexfiles are in the proper order can be a chore without RELOCATE. Just RELOCATE offending lexfiles, and they'll be placed at the end of the lexfile list. (2) EDTEXT gets VERY SLOW(!) when the file being edited has files after it in `CAT ALL`. Just RELOCATE before you EDTEXT, and it'll speed things up.

Note: I always use "00" (oh zero) as a scratch filename since nobody in their right mind would call a real file that name! It likewise makes a wonderful scratch variable, unless anybody has to read the listing! An equally useful/absurd name is "l1" (lowercase el, one).

5. Lexfile Enable/Disable in BASIC

DRIVELEX, JPCLEX and the JPCROM contain a wonderful pair of commands: DISABLE and ENABLE. When you DISABLE a lexfile, it is changed from a LEX file to a "D-LEX" file (Disabled lex? Dead lex?) which effectively turns off all of its keywords, messages, features, etc. It can be revived by ENABLE.

There are many reasons for wanting to turn lexfiles off and back on. The more lexfiles you have in memory, the slower the HP-71 responds, so DISABLE speeds things up. Many lexfiles have no

built-in mechanism for disabling their features except by purging them from memory, so `DISABLE/ENABLE` allows shutting them down without needing to copy them back into memory later. Some lexfiles conflict with others, and if `RELOCATE` (see above) doesn't solve the problem, the only solution may be by selective `DISABLE`. If you want a RAM lexfile to override a ROM lexfile only at certain times, then you must use `DISABLE/ENABLE`. And so on.

But it can be done without a lexfile! Here is a vanilla-BASIC program that converts LEX files to non-executable BIN files and back again:

```
1 ! DISABLE/ENABLE ! Joseph Horn ! 21 June 1988
2 ! Don't use on non-disabled BIN files!
10 INPUT "Filename: ";F$ @ CALL DLEX(F$) @ CAT F$
20 SUB DLEX(F$) @ ON ERROR GOTO 80 @ UNSECURE F$
30 A$=DTH$(HTD(ADDR$(F$))+16) @ T$=PEEK$(A$,4)
40 IF T$="402E" THEN POKE A$,"802E" @ GOTO 60
50 IF T$="802E" THEN POKE A$,"402E" ELSE 80
60 ON TIMER #1,0 GOTO 70 @ OFF
70 OFF TIMER #1
80 END SUB
```

Just run the program, and input the name of the lexfile you wish to disable. The display will turn off for a second, and then you'll see the file's `CAT` display (it's a BIN file now). Run it again, and it'll be turned back into a LEX file. If you input a nonexistent file, or a file that isn't LEX or BIN, or a `PRIVATE` file, then the program will have no effect.

You may also `CALL DLEX(f$)` from within another program, with `f$` containing the name of the file to disable/enable.

Be sure not to `ENABLE` a BIN file that really is originally a BIN file! It'll give you a Memory Lost for sure! And don't `RUN` a disabled LEX file, either!

Change the two "402E"s to "1000" and it'll change LEX files into TEXT files instead of BIN files. Either way, you can save disabled LEX files on card, tape or disk, and reload them into memory any time, and then enable them again.

6. SORTLEX!

On a disk full of goodies that I received from overseas, one lexfile stood out: `SORTLEX`. It's wonderful! Who wrote this gem? If it's on the next `SWAPDISK`, here's what you'll be able to do with it:

- sort `FAST!` (500 numbers or strings in less than 10 seconds);
- sort `DATA` and `SDATA` files in place;
- sort records by any field(s) in any order;
- sort entire file or any specified block;
- case dependent or case independent alphabetizing;
- make `DATA` and `SDATA` files `GROW` by any size;
- `ERASE` (delete; un-`GROW`) any block of records;
- rotate a block of records;
- identify the field(s) in any record;

- find number and length of records.
- find number of empty records.

I already have two useful programs that utilize SORTLEX. One of them is a subprogram that takes any numeric array and sorts it FAST, and the other one is a disk cataloger that lists a disk's contents by name and file type in multiple columns in alphabetical order, FAST! (My lexfile disk has 255 files on it, and it was read, sorted, and printed in no time flat!) These programs will be on the next SWAPDISK if SORTLEX is public domain software.

[Note added in 2015: SORTLEX is available online on the [📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy SORTLEX directly to your HP-71. Its plain-text documentation is available here: [📄 DOC](#)]

Titan File #15 (HPX Exchange, V1 N6, Aug-Dec 1988, pages 5-6)

by Joseph K. Horn <53>

JPC ROM Review

[Note added in 2015: The JPC ROM has undergone many revisions since it first appeared. You can download all its many versions, and all its related files, from the official JPC ROM web page here: <http://www.jeffcalc.hp41.eu/emu71/jpcrom.html>. If you install the latest version, you'll notice some differences between it and the following review which was based on version D.]

If you liked the PPC ROM for your HP-41, you're gonna love the JPC ROM for your HP-71! Just like its predecessor, the JPC ROM is designed for the programmer, since its keywords are of a general nature, not task specific. It was many years under development, resulting in one huge lexfile (over 26K) containing over 100 keywords and many poll handlers (system enhancements).

Unlike the PPC ROM, which was written in slow "user code", the JPC ROM is written entirely in Assembly Language, the fastest possible type of code for the HP-71 developer to use.

Even though the JPC ROM is from PPC Paris, a French users' club, the ROM comes with an Owner's Manual in English, and even a Quick Reference Guide that fits in the HP-71 leather case's pouch.

The JPC ROM is so huge that no review can do it justice. So what I'll do here is break its contents up into ten arbitrary chunks. All of the JPC ROM's keywords and poll handlers can be placed under one of these headings.

1. ADDRESS BOOK

KA puts you into an interactive address book system. It stores names, addresses, telephone numbers, and personal notes. The records are automatically sorted by last name, and you can add new names, browse, search, update, and delete. You can set a password for privacy. The addresses are stored in a file (of a new type, ADRS) which can be stored on disk, tape or cards and retrieved like any other file. This allows multiple address files, making possible such tricks as Brian Walsh's brainstorm of storing appointments, which KA sorts into time order for you.

If you want to use address files in your own programs, there are six functions that interact with ADRS files: ADCREATE, ADDELETE, ADFIND, ADGET, ADPUT, and ADSIZE. All the features of KA can be obtained via these keywords.

2. PRINTER

MODE sets a ThinkJet's pitch. ESC\$, PCR, PFF and PLF send escape, carriage return, form feed, and line feed. PBLIST is a "pretty" version of PLIST, with loops indented, and separations between logical chunks of code. PDIR is just like CAT with the down-arrow held down, and the output going to the printer (or optionally, to a file). BOLD ON/OFF, WRAP ON/OFF, UNDERLINE ON/OFF, PERF ON/OFF and PAGELEN work equally as well on the LaserJet as on the ThinkJet. BELL rings your printer's bell (if it has one).

3. HP-IL

MAXD, MEMD, MAXM and MEMM give data about a disk's or tape's memory. PARPOLL and SRQ

read a parallel poll and service request. NLOOP counts the devices on the loop. RREC\$ and WREC read and write whole records directly from disk or tape.

System enhancement: many previously cryptic or non-existent HP-IL error messages are automatically replaced by meaningful messages (like error 255029, "Write Protected").

4. MATH

ARR and COMB calculate permutations and combinations. Number theorists will like PRIM (first prime factor of), FPRIM (first prime after), NPRIM (number of primes between), and PHI (number of numbers less than and relatively prime to). There's also PGCD (greatest common divisor) and PPCM (least common multiple). FRAC\$ turns a decimal into a fraction.

5. STRING FORMAT

CENTER\$ centers a string within a given width, and FORMAT\$ spreads a string out to fill a given width. RED\$ trims all leading and trailing spaces off, and REDUCE\$ eliminates all extra spaces between words. REPLACE\$ hunts for and exchanges substrings, with wildcard ability. SPACE\$ gives a string of spaces or any other character or string. POSI is like POS but searches for an interval, not just a single value. CESURE tells where to break a string to keep whole words on separate lines (word wrap).

6. TIME/DATE

These functions are clones of the HP-41's Time Module functions. MDY and DMY set mm.ddyyyy or dd.mmyyyy mode. DATEADD is like DATE+; DDAYS and DOW are familiar; and DOW\$ which the HP-41 is mysteriously lacking. DATESTR\$ turns dates into HP-71 date format. HMS and HR parallel the HP-41 functions, with HMSADD and HMSSUB taking the place of HMS+ and HMS-.

7. GRAPHICS

MAP and MAP\$ perform byte-to-byte replacement mapping on a string or on an entire TEXT file, which is useful for many things besides just graphics. PAINT gives LCD pixel control (test, set, and clear). GLINE and GPSET create graphics in strings for ThinkJet or LaserJet printing.

8. PROGRAM STRUCTURES

Not content with just GOTO, GOSUB/RETURN, IF/THEN/ELSE and FOR/NEXT, they've given us EXIT (jumps out of a FOR/NEXT loop); FINPUT (formatted INPUT); multiple-line IF/THEN/ELSE/END IF; WHILE/END WHILE; REPEAT/UNTIL; LOOP/END LOOP; LEAVE (jumps out of a WHILE, REPEAT or LOOP loop); and SELECT/CASE/CASE ELSE/END SELECT. Heavy use of these structures generates highly readable and maintainable code. There's also VARSWAP, which swaps any two variables' values; it isn't really a structure, but it sorta feels like one.

9. SYSTEM CONTROL

ATTN ON/OFF disables the ATTN key. CONTRAST, STARTUP\$, and ENDUP\$ return system settings. ENDUP is like STARTUP but activates at power-down. DBLIST is like PBLIST and DDIR is like PDIR (see #2 above) but to the display instead of to the printer. EXECUTE "runs" a string, allowing non-programmable things to be programmed! FILESIZE works on RAM, ROM and disk/tape files. FIND is like FETCH on the HP-75; it looks for any substring in the line. FKEY is like PUT but works from the other end of the buffer. INVERSE toggles all the LCD pixels. SLEEP puts the 71 into low-drain mode until a key is pressed or alarm comes due. KEYSWAIT\$ is a

combination of SLEEP and KEY\$. LEX ON/OFF disables lexfiles. MARGIN sets the right margin, like the HP-75 does. EDIT now allows disk/tape file names, and lexfiles. MERGE now works on lexfiles. RENUMREM is a “smart” RENUMBER, designed to be used with PBLIST for “pretty” listings. ROMAN ON/OFF sets the alternate character set to the Roman 8 Extension Set, which is like the ThinkJet’s character set. SHRINK chops trailing nulls off TEXT files. STACK sets the command stack height.

System enhancements: The back-arrow key now acts like BACKSPACE in CALC mode (no need to press the f-shift). Pressing VIEW in USER mode shows the position of the cursor. The key repeat speed is automatically sped up (about twice as fast as normal). All HP-71, 75 and 41 filetypes are recognized by CAT, DDIR and PDIR. Automatic Memory Lost recovery is possible; JPC ROM looks for and executes SUB ML immediately after Memory Lost.

10. LOW-LEVEL UTILITIES

Assembly Language programmers and curious spelunkers will enjoy these. SYSEDIT is an interactive memory editor that lets you patch files, and a disassembler that makes lexfile dissection almost painless. There’s also OPCODE\$ and NEXTOP\$ for programmable disassembly. ENTRY\$ and TOKEN give data about keywords. PEEK\$ and POKE now work anywhere in memory, even in secure and private files. ADBUF\$ finds system buffers. ASC\$ makes any string printable (and displayable) by changing non-printable control codes into periods. ATH\$ and HTA\$ convert ASCII to hex.

System enhancement: Pressing SPC in EDTEXT jumps to the next assembler column (tab position). This can be switched off for other EDTEXT work.

THEY’RE NOT BUGS, THEY’RE FEATURES

In my version D of the JPC ROM, if FIND doesn’t find what it’s looking for, it sometimes says “ERR: Overflow” which makes no sense, and other times it hangs the machine for a long time, recoverable by INIT 1.

PGCD, PPCM, and PHI all lock up when given zeros to chew on. INIT 1 is the only way to recover.

The Owner’s Manual is beautifully laid out, but sparse on examples, and the pages are not numbered sequentially. The English mistakes are humorous, and cause no real confusion. The few actual mistakes (such as ADFIND allowing unquoted strings for filenames, an HP-71 impossibility in function arguments) are obvious and harmless.

THE HP-71 STANDARD

It’s sad that such a powerful ROM came so late in the life of the HP-71, which has already been dropped from HP’s catalog. During these last days of HP-71 software development, rallying around the JPC ROM is the obvious way to go. Remember how the PPC ROM unified HP-41 programs in those enthusiastic PPC days? The JPC can do the same for our HP-71 today.

JPC ROM Applications

Contents:

1. [Teeny-Tiny Calendar Program](#)
2. [Poll Handler Disassembler](#)
3. [Fast Disk Copier](#)
4. [Volume Label](#)
5. [Prime Factorizer](#)
(all of the above by Joseph K. Horn <53>)
6. [ALARMLEX](#), by ???
7. [LEXINPUT](#), by ??? (from France)
8. [LEXPRINT](#), by ??? (from France)
9. [MCOPLYEX](#), by ??? (HP maybe?)
10. [FINDLEX](#), by ???

The previous Titan File called all HP-71 users to rally 'round the JPC ROM. And it said that the ROM is useful for programmers. But it gave no examples of programs using JPC ROM functions.

So here goes.

1. TEENY-TINY CALENDAR PROGRAM

This short program uses the DOW (Day Of Week) function, plus YMD and DMY mode settings, to produce a one-month calendar of any specified month and year. The default is today's date. The SPACE\$ function is also used to print a row of asterisks; PRINT USING "20 '*'" would do the same, but much slower.

[Note added in 2015: JCAL is available online on the [📁 HORN/MAIN03 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy JCAL directly to your HP-71. Notice the "19" in line 10; it should be changed to "20" for the 21st century, of course.]

JCAL BASIC 293 bytes

```
10 INPUT "Year, Month? ", "19"&DATE$[1,2]& ", "&DATE$[4,5];Y,M
15 PRINT SPACE$("*",20)
20 PRINT "*";TAB(7);"JanFebMarAprMayJunJulAugSepOctNovDec" [M*3-2,
M*3]& " "&STR$(Y);TAB(20);"*"
25 PRINT SPACE$("*",20)
30 PRINT @ PRINT "Su Mo Tu We Th Fr Sa" @ F=FLAG(-53) @ STD
40 D=Y/1000000+M/100+1 @ DMY @ ON ERROR GOTO 80
50 FOR X=D TO D+30
60 PRINT TAB(DOW(X)*3+1+(X<10));STR$(IP(X));
70 NEXT X
80 PRINT @ IF NOT F THEN MDY
90 END
```

Three noteworthy features here. First, it is the only calendar printing program I know of that does

not perform a carriage return between weeks; it lets TAB handle that. TAB automatically moves to the next line whenever you TAB to a position less than the current position.

Secondly, the program requires DMY mode, but the user's MDY/DMY setting is preserved by storing it into F in line 30, and resetting it in line 80. This was painful for me. I don't believe in the "users"; only programs exist. Greetings, program. [Note added in August 2015: In the 1980's, everybody would have recognized that to be a tongue-in-cheek reference to the movie *Tron*.]

Lastly, months have different numbers of days in them. This program gets them all correct, but never calculates how many days are in the month. It's handled by the ON ERROR GOTO in line 40, which catches DOW trying to find the day-of-week for a nonexistent date (at the end of the month), and ends the program. (Don't let fear of the word "ERROR" frighten you away from using ON ERROR; it's a powerful feature that can often save code and time!) If a month actually has 31 days, then the ON ERROR is never activated; the FOR/NEXT loop simply ends.

2. POLL HANDLER DISASSEMBLER

The usual purpose of lexfiles is to add new BASIC commands. But they can also add new features to the machine. For example, the JPC ROM not only adds a bunch of new commands (like KA, the address book), it also lets you use EDIT and MERGE on lexfiles. This sort of system enhancement is done by "poll handlers". What's a poll handler?

When the HP-71 runs into certain internal problems (like trying to EDIT a non-BASIC file), it gets all excited, and begs all the lexfiles in memory for help. This is called polling the lexfiles. If nobody responds to the poll, the 71 despairs and you get an error message, like usual. But if a lexfile wishes to handle the poll, it grabs control of the HP-71, does whatever it wants to do, then hands control back to the built-in operating system, and tells it to calm down since the error was handled. (It is possible, and sometimes even desirable, for lexfiles to handle polls without letting anybody else know... but I won't delve into that. That's what VER\$ does, for example.)

Suppose I type EDIT MATHROM. This is supposed to cause an error, and does – unless you have a JPC ROM. In that case, the 71 says "Hey, I can't edit a lexfile! Anybody in here that can?" And the JPC ROM says, "No problem; just make the MATHROM the current file; no need for an error." Thus the poll gets handled.

Not all polls are caused by errors; some polls are sent out to the lexfiles during normal operating conditions. For example, at power-down the HP-71 sends out a special poll to see if anybody wants to do anything special at power-down. The ENDUP keyword in the JPC ROM depends upon this poll to operate.

The polls handled by the JPC ROM are all listed and explained very nicely at the back of the ROM's manual. But how do you find out what polls are handled by other lexfiles? By running the following program! By the way, it requires NIBBLEX (or equivalent), found on the LEXFL1 swapdisk. See [HPX V1 N3 P18, "Uses of PEEKUTIL"](#) for some background. (NIBBLEX and PEEKUTIL are functional equivalents, although their keywords have different names).

[Note added in 2015: This version of POLLEDIT is available online on the [📁 HORN/MAIN03 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy POLLEDIT directly to your HP-71. There is another version, slightly different, on the [📁 HORN/MAIN04 LIF Disk image](#); try both versions and see which you prefer. NIBBLEX and PEEKUTIL are both available online on the [📁 SWAP/lexfl1 LIF Disk image](#).]

POLLEDIT BASIC 422 bytes

```
1 ! PollEdit ! SYSEDTs a lexfile's poll handler entry point(s)
2 ! By Joseph K. Horn ! 9 Aug 1988
3 ! Requires NIBBLEX & JPC ROM
10 INPUT "Lexfile? ";F$ @ A=HTD(ADDR$(F$))+37
11 ! IF RNIB$(A-21,4)#"E208" THEN DISP "Not a lexfile!" @ STOP
20 L=NIB(A+6,5) @ IF L THEN L=OFFSET(A+6,5) ! save next lex table
30 IF NOT NIB(A+11) THEN A=A+79 ! skip speed table
40 IF NIB(A+20,5) THEN PUT "D" @ SYSEDT OFFSET$(A+20,5) ! poll handler
50 IF L THEN A=L @ GOTO 20 ! go to next lex table
60 END
```

Line 11 checks to make sure that the file is really a lexfile. It is commented out to allow the program to examine disabled lexfiles ("D-LEX"). If you wish to examine only enabled lexfiles, then remove the "!" at the beginning of line 11.

If the specified lexfile has no poll handlers, this program does nothing. If there is a lexfile, the JPC ROM's disassembler is started at the poll handler code. You may then disassemble the poll handler to see which polls are handled (its number is in CPU register B, B field).

For example, if you type MATHROM at the "lexfile?" prompt, you'll see SETHEX, which is the first instruction of its poll handler code. Press ENDLINE three times, and see LCHEX F2, then ?B=C B/GOYES nnnnn. This means that the MATHROM handles the #F2 poll; press ENDLINE again to see LCHEX 39, then ?B=C B/GOYES nnnnn. So it handles poll #39 too. What these polls do is found in the IDS, Volume I, just before the final index. One more ENDLINE shows RTNSXM, which normally marks the end of a poll handler's code. At that point you may press ATTN to exit the disassembler, and exit the program.

If the lexfile you're disassembling contains linked lex tables and more than one poll handler, POLLEDIT automatically dumps you into the next one when you press ATTN. Pressing ATTN while you're in the final poll handler exits the POLLEDIT program.

3. FAST DISK COPIER

"Disk1to2" is a rapid disk copier for those with two 9114 disk drives. I imagine that it would also work with two cassette drives. It uses the MASSCOPY command found in the MCOPYLEX file (also called MASSCOPY on some Public Domain software collections). MASSCOPY is a rapid method of copying entire disk records (256 bytes) to another disk (or device). [[📄 MCOPYLEX's documentation](#)]

[Note added in 2015: MCOPYLEX is available online on the [📄 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy MCOPYLEX directly to your HP-71.]

The charm of this disk copier is that it copies only the portions of the disk that are in use, not the blank tracks. The popular disk copiers make bit-image copies of the entire disk, which takes over four minutes. But this program often takes just seconds. For example, it copied a disk I got from Australia with 30 files on it, in 13 seconds!

If a disk is rather full, however, it may be faster to use one of the regular disk copiers, such as MCOPYD2 found on the swapdisks.

[Note added in 2015: Disk1to2 is available online on the [📁 HORN/MAIN03 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy Disk1to2 directly to your HP-71. Also, the MCOPLYD2 program mentioned above is available online on the [📁 SWAP/chhu04 LIF Disk image](#).]

DISK1T02 BASIC 529 bytes

```
1 ! Disk1to2 - Quick Disc Copier - Joseph K. Horn - 17 Jan 1989
2 ! Uses JPC Rom, MCOPLYLEX - Copies only used tracks.
3 ! Place master in drive(1), slave in drive(2), and RUN.
4 !
10 D1=DEVADDR("%16") @ D2=DEVADDR("%16(2)")
20 B=MAXD(D1)/8 @ E$=SPACE$(255,32) @ FOR D=2 TO B+1
30 P=POS(RREC$(D,D1),E$) @ IF P THEN 50
40 NEXT D @ DISP "First File is Corrupt!" @ P=1
50 MASSCOPY :D1,0,1,:D2,0 @ T=TIME @ MASSCOPY :D1,2,D-1,:D2,2
60 P=P*2-66 @ IF P<0 THEN D=D-1 @ P=448 @ B=B+2
70 E$=ATH$(RREC$(D,D1),1)[P+30,P+40] @ E=HTD(E$[1,3])+HTD(E$[9])-B
80 MASSCOPY :D1,B,E,:D2,B @ T=TIME-T
90 DISP "Done:";T DIV 60;"min,";IP(RMD(T,60));"sec." @ END
```

Note: The HP-71's HPIL module (both A and B versions) has a bug that allows a disk catalog to contain one too many entries. If this occurs, the first record of the contents of the first file on the disk gets erased. If you run Disk1to2 on such a disk, it'll tell you "First File is Corrupt!" and then it'll copy the disk anyway.

4. VOLUME LABEL

There are many ways of copying the current file to disk. If the disk drive is the first device on the loop, you can say COPY TO: 1, which is the fastest of all. Or you can say COPY TO :MASSMEM (or COPY TO :TAPE, which is exactly the same thing), which goes to the first drive on the loop, wherever it is. Or you can say COPY TO :DD if you used ASSIGN IO to call your disk drive DD. (I never do. It's allowed, just to make HP-75 owners feel at home, but it's slow and unnecessary). Or you can say COPY TO %16 because the drives are of device type 16. Or COPY TO :HP9114 if you have an HP9114.

But my favorite way (because I don't have to press either shift key) is COPY TO .A when one of my own disks is in there. All my disks have volume label "A", and I use it all the time. So RUN FRED.A means "Copy FRED off the disk with volume label A and run it."

Here's a program to see (and optionally change) a volume label:

[Note added in 2015: VOLUME is available online on the [📁 HORN/MAIN04 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy VOLUME directly to your HP-71.]

VOLUME BASIC 493 bytes

```
10 ! Disc Volume Label view & update
11 ! Joseph K. Horn - Uses JPC Rom
... continued on next page ...
```

```

12 ! Just RUN for drive 1; CALL VOLUME(n) for drive n.
20 CALL VOLUME(1) @ SUB VOLUME(N)
30 D=DEVADDR("%16("&STR$(N)&")) @ DIM A$[6],B$[6] @
   A$=RREC$(0,D)[3,8] @ B$=A$
40 FINPUT A$,"Label: ["&A$&"]", "8pu",A
50 IF A THEN A$=REPLACE$(UPRC$(A$)," ","") @ A$[7]=" "
60 IF A$=B$ THEN DISP "No Change Made" @ STOP
70 ON ERROR GOTO 90 @ CREATE TEXT RED$(A$)&"00:MAIN"
80 OFF ERROR @ PURGE RED$(A$)&"00:MAIN" @ GOTO 100
90 BEEP @ DISP "Invalid; Try Again" @ WAIT 1 @ OFF ERROR @ A$=B$ @ GOTO 40
100 WREC RREC$(0,D)[1,2]&A$&RREC$(0,D)[9],0,:D
110 DISP "Volume Label Updated"

```

If you haven't used `FINPUT` yet in your own programs, check it out. It's powerful. It is used here to make sure that the user doesn't type a volume label longer than six characters long (the maximum allowed by HP). The `A$[7]=" "` in line 50 is a trick to tack spaces onto strings less than 6 characters; its purpose here is not to shorten strings longer than 6 characters.

5. PRIME FACTORIZER

Here's a rapid routine that factors any integer into its primes:

```

10 SUB PF(Q) @ N=Q @ DISP " "&STR$(N)&"=";
20 E=1 @ F=FACTOR(N)
30 N=N DIV F @ IF NOT RMD(N,F) THEN E=E+1 @ GOTO 30
40 DISP STR$(F); @ IF E#1 THEN DISP "^";STR$(E);
50 IF N#1 THEN DISP "*"; @ GOTO 20 ELSE DISP

```

Now assign `"CALL PF(VAL(DISP$))"` to a key. This is now your [PF] key. Type a number and press [PF] to see its prime factorization.

For example, press 12343211 [PF] and see 12343211=103*293*409. Press 43212344 [PF] and see 43212344=2^3*7*197*3917. Of course, "^" means "raised to the power of," and "*" means "times."

6. ALARMLX

Note: Over the years, many folks have asked for an HP-41 Time Module style alarm lexfile for the HP71. If you can hunt down a lexfile called `ALARMLX` or `ALARMLX` or something like that, you'll have it. It's floating around on Public Domain software disks. A version 2174 bytes long has been around for many years but it had many bugs that caused Memory Lost. The debugged version is 2180 bytes long.

It's great. It lets you set multiple alarms, by time and date. You specify a "Command Line" for each alarm, to be executed when the alarm comes due. You can optionally set a repeat interval like on the HP41, for repeating alarms. But you can also specify how many times it should repeat before clearing itself automatically! And you can specify whether each is a "hard" or "soft" alarm (interrupting or non-interrupting).

You can clear alarms selectively or all of them at once. You can catalog the alarms and see their contents. The only thing you can't do is save alarms onto external media, since they're saved in a buffer, not a file. This also limits their total size to 4K (not a problem). It seems to be immune to the HP-71's clock bug; I've not yet seen it miss an alarm. Now I can finally retire my HP-41, which for several years was used only at bedside as my alarm clock!

If you can't find it on any of your PD disks, please send a POST CARD requesting it, and I'll send it on two mag cards, courtesy of John Dearing, who donated a mountain of mag cards to the PD cause.

[Note added in 2015: The debugged 2180-byte lexfile called ALARMLX is available online on the [📄 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy ALARMLX directly to your HP-71. WARNING! That disk also contains a file called ALARMLEX; you don't want that one.]

ALARMLEX SYNTAX GUIDE: [or [📄 download its complete documentation here](#)]

ALARMCAT — press down-arrow to review all pending alarms.

ALARMOFF — deletes all pending alarms. The alarm buffer is cleared from memory. HP-71 configuration without ALARMLEX in the lex buffer also causes the alarm buffer to be cleared from memory.

DELALARM date\$;time\$ — deletes the alarm from the alarm buffer with the specified date and time. Example: DELALARM "89/02/13";"06:00:00" deletes the alarm which is set to activate at 6 AM on 13 Feb 1989. If there are several alarms set for that time, the one most recently set is deleted. Note well: semicolons are used, not commas, between ALARMLEX arguments.

SETALARM date\$;time\$;command\$[;repeat\$;reps] — sets an alarm for the specified time and date. When the alarm comes due, it executes commands by placing command\$ in the command stack as if the user had typed it and pressed ENDLINE. If a repeat\$ is specified, the alarm will repeat after that much time has elapsed, reps times, at which time the alarm will clear itself. If no repeat\$ is specified, the alarm will clear itself as soon as it activates. Running programs will be interrupted by SETALARM alarms.

SETSOF T uses the same syntax as SETALARM, but it sets alarms that will not interrupt running programs.

The hex dump of ALARMLEX (for input with the LEXINPUT program below) appears at the end of this Titan File. [See above for download location.]

7. LEXINPUT

This is a program from France, an improvement over the old MAKELEX program. It lets you type lexfiles into the HP-71 from printed listings. It is better than MAKELEX because it has better checksums (almost fool proof) and creates files of exactly the correct size. MAKELEX always tacked at least 5 extraneous nibbles onto each lexfile.

Just run it, and input the number of NIBS in the target file (see its listing, at the top). Then input each row's nibs (omit the spaces), press ENDLINE, then that row's checksum, press ENDLINE, and then do the next row, etc. You can tell by the dashes how much to put in. When the program ends, it'll turn off and back on, to add the new file to the lexfile chain. If you decide to bail out early,

PURGE AH, the scratch file used by LEXINPUT.

[Note added in 2015: LEXINPUT is available online on the [📁 HORN/MAIN03 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy LEXINPUT directly to your HP-71.]

LEXINPUT BASIC 526 bytes

```
10 INTEGER N,X,B,M,Z
20 DIM A,A$,C$[16],D$[3],S @ A$=""
30 DEF FNP$
40 IF X#N DIV 16 THEN B=16 ELSE B=RMD(N,16)
50 IF NOT FLAG(1) THEN FNP$="-----"[1,B] ELSE FNP$=C$
60 END DEF
70 SFLAG -1 @ PURGE AH @ LC OFF
80 INPUT "Nibs? ";N
90 CREATE DATA AH,1,CEIL((N-37)/2-4) @ A=HTD(ADDR$("AH"))
100 FOR X=0 TO N DIV 16
110 GOSUB 190
120 IF X=0 OR X=1 THEN A$=A$&C$ @ GOTO 140
130 IF X=2 THEN POKE DTH$(A+37),C$[6,B] ELSE POKE DTH$(A+16*X),C$[1,B]
140 NEXT X
150 POKE DTH$(A),A$
160 ON TIMER #1,1 GOTO 180
170 BYE
180 END
190 CFLAG 1
200 DISP DTH$(X)[3]; @ INPUT ": ",FNP$;C$
210 DISP DTH$(X)[3]; @ INPUT " sum: ", "---";D$
220 M=S
230 FOR Z=1 TO LEN(C$) @ M=NUM(C$[Z])+M+1 @ NEXT Z
240 IF D$#DTH$(RMD(M,4096))[3] THEN BEEP @ SFLAG 1 @ GOTO 200
250 S=M
260 RETURN
```

8. LEXPRINT

This is the counterpart of LEXINPUT. Run LEXPRINT, input the name of the lexfile you wish to print, and a listing of it will be LEXPRINTed.

[Note added in 2015: LEXPRINT is available online on the [📁 HORN/MAIN03 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy LEXPRINT directly to your HP-71.]

LEXPRINT BASIC 496 bytes

```
10 INTEGER Z,N,X,B,Y
20 DIM A$[16],A,S @ A$=""
30 DEF FNP$
40 FOR Z=1 TO LEN(A$) @ S=NUM(A$[Z])+S+1 @ NEXT Z
50 FNP$=DTH$(RMD(S,4096))[3] @ END DEF
60 INPUT "Filename? ";N$ @ A=HTD(ADDR$(N$))
70 A$=PEEK$(DTH$(A+32),5)
80 N=HTD(A$[5]&A$[4,4]&A$[3,3]&A$[2,2]&A$[1,1])+32
90 PRINT UPRC$(N$);" : ";N;"Nibs (";STR$(CEIL(N-37)/2);" bytes)" @ PRINT
100 PRINT "Row";TAB(14);"Hex Dump";TAB(34);"Check";TAB(35);"Sum" @ PRINT
110 FOR X=0 TO N DIV 16
120 IF X#N DIV 16 THEN B=16 ELSE B=RMD(N,16)
130 A$=PEEK$(DTH$(A+16*X),B) @ PRINT DTH$(X)[3];" ";
140 PRINT A$[1,2];" ";A$[3,4];" ";A$[5,6];" ";A$[7,8];" ";
150 PRINT A$[9,10];" ";A$[11,12];" ";A$[13,14];" ";A$[15,16];" ";
160 PRINT TAB(35);FNP$
170 NEXT X
```

9. MCOPLYEX

This lexfile adds just one keyword to BASIC: MASSCOPY. It copies records (blocks of 256 bytes) from one device to another on the loop.

Syntax: MASSCOPY :A,s,n,:B,t where :A is the source device, s is the starting record number (0 thru 2643 for a 9114 drive), n is the number of records to be copied, :B is the destination device, and t is the target record (where the records begin to get recorded).

Example: MASSCOPY :TAPE(1),2,5,:TAPE(2),153 copies from drive 1, starting at record 2, for five records (therefore stopping with record 6), to drive 2, recording onto record 153 and following (through record 157).

MASSCOPY uses all of available memory as one huge buffer. The more memory you have in your HP-71, the faster MASSCOPY will work. Although large tasks with large amounts of memory appear to be slowed down by the drives timing out, and then turning back on again, the overall performance is faster than when MEM is small.

[Note added in 2015: MCOPLYEX is available online on the [📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy MCOPLYEX directly to your HP-71. [📁 MCOPLYEX's complete documentation.](#)]

MCOPYLEX : 920 Nibs (441.5 bytes)										
Row	Hex Dump								Check Sum	
										01C 0B 46 46 02 40 20 37 22 51F
										01D 07 E2 07 24 07 62 07 A3 884
000	D4	34	F4	05	95	C4	54	85	389	01E 07 E1 07 A0 07 61 07 A2 BF1
001	80	2E	00	40	91	20	80	51	6D6	01F 00 38 4A 20 77 A0 52 97 F59
002	87	30	0D	51	C1	C0	00	00	A39	020 00 18 FB 39 40 8F EA 23 2F3
003	F7	10	00	00	00	00	00	00	D67	021 04 00 8D B2 E2 08 F9 DF 69F
004	06	20	00	DF	D4	14	35	35	0D6	022 30 59 08 D5 3E 20 86 09 A12
005	34	F4	05	95	1C	1F	FE	E1	486	023 08 D5 3E 20 87 39 08 D5 D99
006	00	B6	10	07	2B	07	5B	01	7E9	024 3E 20 8D 27 13 07 72 07 103
007	F1	78	F2	14	51	61	71	C0	B64	025 83 07 82 07 03 07 02 07 449
008	1F	67	8F	21	41	16	17	0B	EDF	026 82 07 F0 07 02 07 01 08 799
009	01	FB	78	F2	14	11	61	7B	266	027 D3 03 50 20 8D 95 75 02 B02
00A	70	7E	70	1F	69	8F	21	45	5F0	028 01 4B 8D 22 95 02 03 1C E70
00B	16	17	A8	0A	F2	1F	B7	8F	99F	029 2A EA 8F BE C2 01 71 14 221
00C	21	47	BF	2B	F2	BF	2B	F2	D65	02A B0 10 61 37 1F 0F 7F 21 5A2
00D	BF	2D	61	0B	1F	17	8F	21	10C	02B 45 1D 5F 07 15 57 1E 50 91F
00E	47	10	A1	F6	78	F2	14	3D	499	02C 8F 15 17 1D 51 D9 80 F5 CB6
00F	81	F6	98	F2	14	71	09	71	815	02D 26 A8 25 50 A0 E2 7A 82 03D
010	A1	17	54	04	A4	75	91	0C	B8A	02E A0 E1 5D 70 42 03 2C FB 3DA
011	70	04	E3	8D	84	A8	08	DD	F2E	02F 8F AB 81 15 43 31 FF D5 78A
012	2F	90	46	72	F3	08	20	94	2A2	030 D0 E4 15 F5 96 A1 29 65 B1F
013	3C	03	3A	30	06	16	0D	B0	615	031 61 BF 6F 69 E2 C0 F6 F6 EEB
014	38	F8	71	F0	AC	08	D3	22	9B3	032 B6 25 51 17 A5 AD 33 32 26E
015	B1	89	0A	38	91	53	89	20	D28	033 00 8D A9 39 01 75 14 7D 5ED
016	38	84	52	79	31	26	B3	07	08E	034 50 71 35 17 41 37 06 14 934
017	F2	04	51	79	21	19	76	05	3EF	035 7C 90 61 F5 08 F2 15 37 CB9
018	70	88	4A	02	48	0C	02	38	759	036 17 F1 5F 7D 52 7B 06 44 04C
019	0C	12	23	1F	F2	08	DA	93	AEC	037 00 52 69 0E 20 80 D5 1E 3C0
01A	90	86	C6	20	61	36	1B	24	E57	038 5F 7F 15 77 1D 0F 06 14 752
01B	4F	21	56	41	34	07	94	AC	1D3	039 71 35 07 01 8F2

10. FINDLEX

The JPC ROM's `FIND` command has a bug in revision D. If you load the following lexfile into memory, the module's `FIND` command will work properly.

[Note added in 2015: FINDLEX is available online on the [📁 SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy FINDLEX directly to your HP-71. However, you will not need this lexfile if your version of the JPC ROM is newer than version D.]

FINDLEX : 671 Nibs (317 bytes)												
Row	Hex Dump								Check Sum			
										014	35 1B 09 8F 21 46 10 91	A88
										015	84 14 6E A4 D4 CE 80 D0	E35
000	64 94 E4 44 C4 54 85 02	377	016	F6 D7 54 2D 90 61 37 06	1BB							
001	80 2E 00 65 22 20 80 51	6C5	017	13 7D BD 57 01 10 71 35	52E							
002	F7 20 01 EB 4B 40 00 00	A36	018	07 D5 56 2C D1 19 13 44	8A9							
003	F7 10 00 00 00 00 00 00	D64	019	51 14 A1 71 14 F9 62 9C	C26							
004	00 20 00 77 64 94 E4 44	0BC	01A	CD 52 F1 12 13 11 12 AF	FAE							
005	B4 1F F8 DD 97 30 9F FF	49D	01B	2D 61 33 EA 20 11 B1 35	326							
006	F8 F6 81 F0 8F 13 DB 08	856	01C	8A 86 71 03 8F 96 F4 01	6B0							
007	A8 84 1B B8 8F 21 40 13	BE1	01D	BF 85 F2 14 61 34 1F 1C	A4B							
008	71 35 16 41 44 8F 8B B8	F67	01E	6F 2A D0 14 3E A1 1B CA	E04							
009	18 FE EB 60 40 18 F6 27	30A	01F	81 C1 00 D9 06 7C 00 B1	181							
00A	70 5D 03 1F 38 D3 93 90	68B	020	C3 E3 B1 E3 FF 07 13 58	52E							
00B	8F 99 97 05 11 20 33 8E	A07	021	FE 0C 10 07 D5 8F 61 79	8D2							
00C	00 8D A9 39 01 F8 E7 F2	DAC	022	01 18 8F 1C 69 08 F8 98	C64							
00D	D2 15 F3 8F 4E FF 05 CD	180	023	10 8D E7 30 08 FC 2D E0	00C							
00E	13 7D 51 37 8F 13 00 18	4EA	024	D7 8F 13 00 12 08 B3 60	37F							
00F	B3 70 D4 13 1D B8 FD 3D	8A2	025	69 6E 11 B1 37 10 B8 FD	71D							
010	E0 13 71 35 10 B8 FF 6F	C3E	026	A6 70 11 B1 35 49 08 D8	A99							
011	40 20 AF 21 BF 85 F2 14	FD0	027	4A 80 63 9E 15 27 15 77	E10							
012	21 A1 C6 F1 46 E2 81 EE	379	028	97 60 01 6F 17 FC D5 AE	1C3							
013	6D 4E AD 8C 41 46 10 A1	71E	029	15 21 15 71 91 60 00 3	4CC							

[Note added in 2015: The debugged 2180-byte lexfile called ALARMLX is available online on the [SWAP/lexfl1 LIF Disk image](#). If you have a PIL-Box, you can download the disk image and copy ALARMLX directly to your HP-71. [Download its complete documentation here](#). WARNING! That disk also contains a file called ALARMLEX; you don't want that one.]

ALARMLEX : 4397 Nibs (2180 bytes)												
Row	Hex Dump								Check Sum			
										089	49 5F 21 42 8B A6 06 86	888
										08A	C1 B9 95 F2 14 41 34 11	C00
000	14 C4 14 25 D4 C4 54 85	37D	08B	2C C4 A1 81 C1 4F 14 CC	FB0							
001	80 2E 00 51 91 20 80 51	6CC	08C	C4 C0 17 11 61 6E EF DB	367							
002	D0 11 0E A1 05 00 00 00	A1E	08D	13 51 C9 1C B1 CD 20 34	6F4							
003	FB 30 06 51 07 46 00 00	D76	08E	62 00 01 12 C2 13 31 30	A30							
004	0A DD 00 D3 10 06 01 0D	0F2	08F	D5 32 61 88 F5 29 11 85	DAA							
005	62 03 0A 00 D9 30 A8 20	459	090	40 91 FE B6 F2 15 53 87	140							
006	0D C4 0E 62 00 DF 14 C4	7F7	091	56 06 6A B7 60 38 F8 44	4C8							
007	14 25 D4 34 14 45 10 F1	B58	092	10 8F BF 19 08 D0 26 20	84F							
008	4C 41 42 5D 4F 46 46 42	ED7	093	1F 19 8F 21 47 D7 00 8D	BE3							
009	0F 44 54 C4 14 C4 14 25	24D	094	A9 39 06 CC B6 CB BD 82	FB2							
00A	D4 30 F3 55 44 51 4C 41	5C5	095	01 37 C2 10 B1 12 34 62	308							
00B	42 5D 44 0D 35 54 45 35	932	096	00 0C AD 6C 0D 53 4F FF	6CB							
00C	F4 64 45 50 1F F8 DB 2E	CE6	097	00 8B E0 DD 41 01 32 61	A44							
... continued on next page continued on next page ...		

00D	20 8F AB 63 03 12 F9 66	073	098	88 F1 1A 11 50 C1 71 1B	DC3
00E	BE 8F 8E C2 08 FA B6 30	444	099	17 8F 21 5A B1 00 11 BD	14D
00F	31 2F 96 64 D8 F8 EC 20	7ED	09A	72 B1 10 15 FB 87 24 49	4CA
010	8F AB 63 03 12 F9 62 82	B7E	09B	9A 84 20 34 D1 00 01 33	825
011	31 0F 96 28 13 10 49 62	EDB	09C	CA 13 11 43 13 7C 21 35	B8F
012	F0 31 12 96 26 06 F9 F8	262	09D	17 81 37 13 58 B3 5C 87	F07
013	D2 71 30 8F 8E C2 08 FA	612	09E	2C 01 13 13 16 45 B6 2D	273
014	B6 30 31 2F 96 65 38 F8	99C	09F	09 16 0C 60 90 13 7C EC	5FB
015	EC 20 8F D9 63 08 D2 71	D40	0A0	ED 51 13 EA D6 23 32 00	988
016	30 8F AB 63 03 12 F9 66	0CE	0A1	08 1E 20 D7 11 31 30 18	CE5
017	01 8F 8E C2 08 DD 97 30	472	0A2	1D 21 19 CA 13 11 C1 CF	07E
018	6A 4F 8D 39 45 00 0A 00	7F5	0A3	42 11 4E 14 D1 81 1C 16	3ED
019	10 04 45 96 D6 56 02 CE	B71	0A4	EE FD 91 35 65 FA 83 7F	7BD
01A	11 08 45 F6 F6 02 D6 16	EEF	0A5	F1 17 FF 08 85 26 C8 82	B5A
01B	E6 97 02 DE 10 37 C8 02	27D	0A6	03 26 18 8F AB 81 15 57	EDA
01C	0E 52 CB 03 0D E1 E8 EC	640	0A7	8A 82 61 37 13 5C 21 0B	24F
01D	21 40 2E 4F 60 2D E1 03	9BD	0A8	17 16 A0 F2 03 4D 10 00	5B3
01E	7C B0 50 EC ED D1 CB 06	D8F	0A9	13 31 30 CA 13 11 43 34	906
01F	0E 6E D0 0C B1 70 DE 18	142	0AA	62 00 0C AD 81 81 32 61	C74
020	02 07 16 37 47 02 46 57	48F	0AB	88 F5 29 11 75 61 66 3B	FF0
021	56 C1 18 0D 00 3E 3D 32	80F	0AC	20 33 30 EA 28 8F CB 39	38A
022	34 3C 51 90 E6 E5 25 56	B92	0AD	06 12 B2 03 26 18 8F 14	6EE
023	07 56 16 47 CE 0D 13 44	F0E	0AE	A1 12 03 34 0E A6 CD F0	A86
024	16 47 56 C0 1E 14 14 C6	287	0AF	02 B1 FD 07 F2 15 31 1F	E15
025	16 27 D6 CF F1 39 38 36	621	0B0	55 7F 21 57 19 1A 3E 99	1A1
026	30 23 4F 60 79 72 79 67	98F	0B1	AE D8 5E 20 1F 14 9F 2D	560
027	68 64 72 69 7D 41 43 54	CFF	0B2	B1 45 17 F0 71 45 17 F0	8D9
028	5E 49 50 2A 40 2D 49 43	07C	0B3	71 45 17 F0 71 45 31 4C	C44
029	48 41 45 4C 42 03 35 0E	3E3	0B4	8F AF 53 13 26 18 8F AB	FF7
02A	A6 C9 62 03 36 0E A6 09	76F	0B5	81 15 E3 17 1A F2 AF 01	388
02B	62 03 38 0E A6 48 62 03	AD8	0B6	5F B1 F5 57 F2 15 BB 97	73F
02C	37 0E A6 87 6B BE FF 40	E9C	0B7	61 23 15 D8 FC 46 31 57	AC0
02D	EF F0 88 53 6F 00 8A EF	265	0B8	11 F1 49 F2 14 71 08 64	E2D
02E	F1 FD FF 08 8F 28 1F 08	631	0B9	EB 64 60 32 61 88 FA B8	1D1
02F	F6 CF 21 42 A1 F1 78 F2	9DF	0BA	11 54 52 01 4B 31 20 0E	525
030	15 DB 16 18 F6 81 F0 8F	D84	0BB	62 96 A9 08 4E 6C 30 1B	8B6
031	46 E2 14 F8 8E DC D0 9F	14F	0BC	4E 6F 21 42 C4 59 08 D7	C4C
032	E0 91 FD 78 F2 15 9B 16	4F7	0BD	48 61 1B C0 BF 21 4E 96	FE8
033	18 E6 CD 08 F2 B5 21 8E	8A9	0BE	A6 18 F5 A1 7E 20 34 C4	381
034	0D D0 13 21 BE B6 F2 15	C45	0BF	B1 E0 60 31 F1 79 F2 14	707
035	63 0A 13 21 F1 78 F2 AF	FDB	0C0	70 61 CF 14 70 61 CF 14	A8F
036	21 5F B8 E0 8D 08 FB BC	3A9	0C1	70 61 CF 14 7D 70 06 5C	E1B
037	E0 AF 01 FD 78 F2 15 BB	771	0C2	E2 03 26 18 8F AB 81 15	1A6
038	2B A1 21 F1 78 F2 15 DB	B14	0C3	DE 8A 88 11 71 AF 01 5B	551
039	87 25 1A F0 11 19 96 60	E83	0C4	B3 05 8F D0 92 10 12 0A	8CD
03A	6B 20 68 1F 66 B6 AF 21	220	0C5	F2 1F 55 7F 21 5D B3 26	C6C
03B	BD 78 F2 15 CB 16 B1 44	5CA	0C6	18 8F 14 A1 10 10 38 D3	FDE
03C	63 D0 65 31 6C DE 8F 68	974	0C7	03 50 31 02 14 C1 61 01	31D
03D	1F 02 01 4A 31 0F 96 2D	CF2	0C8	31 FF 63 FF 8F 92 23 11	6C2
03E	C3 14 F9 62 4C 31 CF 96	097	0C9	03 8F 25 23 18 F1 6C 21	A3B
03F	2B B1 61 8F 68 1F 01 61	420	0CA	72 63 15 47 16 F0 11 1B	DA0
040	8F 68 1F 08 FC 09 E0 04	7C6	0CB	8F 53 33 18 F1 6C 21 2A	12B
041	43 B8 F3 22 B1 59 A3 41	B4E	0CC	31 F2 24 31 F2 78 33 15	494
042	00 00 8B A5 08 51 10 31	EA1	0CD	47 16 F0 11 B1 78 F2 7B	821
... continued on next page continued on next page ...		

043	33	09	1F	EB	6F	21	55	31	22B	0CE	14	3F	14	C6	16	27	D6	02	B99
044	31	8F	BD	3B	11	B9	95	F2	5DB	0CF	02	16	15	47	16	F3	54	70	EF3
045	13	71	35	14	48	F4	6E	21	948	0D0	2F	F1	5C	51	F1	78	F2	8F	2AC
046	56	06	46	F9	7A	9F	1B	D7	CF7	0D1	E0	C1	07	BF	3A	F2	15	FB	66E
047	8F	21	5C	B1	6B	11	31	40	075	0D2	17	B0	4B	F6	BF	68	1E	76	A23
048	1B	99	5F	21	46	13	58	FB	40B	0D3	5F	20	37	02	F6	E6	02	15	D9B
049	D3	B1	8F	13	DB	01	B9	95	7B7	0D4	C7	16	77	46	F7	33	F3	41	125
04A	F2	13	71	35	14	43	40	C0	B16	0D5	78	F2	13	71	0A	04	77	F2	4A3
04B	00	8B	E1	41	02	20	32	61	E6B	0D6	20	3F	25	56	07	56	16	47	804
04C	88	FA	B8	11	5B	08	A8	60	20C	0D7	02	FF	AF	A7	EA	21	B1	78	BCC
04D	6C	74	11	23	46	20	00	CA	577	0D8	F2	11	A1	37	AF	21	5F	B1	F6A
04E	D8	20	32	61	88	FD	79	11	8FD	0D9	7B	97	A0	60	4B	F6	BF	68	322
04F	4D	45	21	67	20	20	33	90	C51	0DA	1E	72	EE	7A	CE	31	C2	14	6D9
050	EA	6B	34	20	33	81	00	6F	FD3	0DB	C1	61	7C	BE	11	B2	B9	1A	A88
051	24	20	33	10	EA	63	24	20	329	0DC	33	21	3E	30	24	41	69	78	DE5
052	33	20	EA	67	14	31	10	86	68C	0DD	23	79	21	54	71	6F	20	37	146
053	34	30	B8	51	0B	6A	20	20	9F3	0DE	02	16	E6	46	15	C7	16	77	4B9
054	13	21	BE	B6	F2	15	63	0A	D81	0DF	18	E0	13	5E	4F	6F	F1	5C	86A
055	13	28	71	4D	31	00	86	36	0DA	0E0	51	65	07	17	96	75	16	5D	BD5
056	03	12	01	4D	17	11	B1	78	435	0E1	CC	5E	FF	55	EF	F8	4C	08	FC3
057	F2	15	EB	15	DB	17	B1	BD	7F7	0E2	20	1F	17	8F	23	F0	20	21	332
058	78	F2	15	EB	15	DB	17	B1	BA2	0E3	4C	61	62	7D	63	61	55	71	6A5
059	6B	14	61	45	17	41	B9	95	F15	0E4	7F	3D	16	47	16	C6	F6	76	A44
05A	F2	14	61	34	11	A1	45	17	275	0E5	FF	15	DD	1C	F8	FE	0C	10	E1F
05B	42	23	30	00	02	0C	E4	81	5CA	0E6	8F	E9	22	03	4F	FF	80	CE	1E8
05C	81	ED	A1	4E	14	D1	61	17	960	0E7	5D	F8	FE	92	20	32	61	88	583
05D	1C	C5	1F	31	FF	14	D7	04	D07	0E8	FA	B8	11	56	88	A8	18	13	917
05E	61	BE	B6	F2	15	63	0A	87	0A4	0E9	71	35	C2	10	88	40	20	14	C68
05F	49	08	D8	4A	80	60	23	03	410	0EA	B3	12	00	E6	29	6A	50	85	FDf
060	00	EE	50	00	7D	A1	35	14	789	0EB	01	71	7F	1E	14	31	79	8F	362
061	7E	A1	31	30	31	55	00	3D	AF3	0EC	FA	CE	00	48	0F	68	0F	C8	722
062	B1	0A	31	4C	8F	C4	63	15	E86	0ED	0F	58	0F	A8	0F	48	0F	88	ACC
063	B1	AF	22	03	10	3A	FA	30	216	0EE	0F	38	0F	68	0F	28	0F	48	E63
064	58	F7	19	21	5A	23	26	18	589	0EF	0F	18	0F	22	CA	F0	A9	A2	213
065	8F	AB	81	15	B1	8A	86	11	925	0F0	0A	F2	3D	F0	F0	F0	F0	F0	5D1
066	71	AF	01	5B	B3	05	8F	D0	CC8	0F1	F0	F0	0E	72	3D	03	03	03	94B
067	92	11	1A	D7	00	70	8F	00	037	0F2	03	03	03	03	0E	7A	2C	90	CB2
068	04	20	31	CF	96	14	93	1B	3AD	0F3	EF	03	10	00	D0	D8	80	0F	03F
069	19	61	6E	31	EF	96	15	53	735	0F4	20	70	C0	15	47	16	F3	D0	3B0
06A	18	F9	61	33	31	9F	96	1A	ABE	0F5	24	79	6D	65	63	7F	F1	5C	750
06B	23	1B	F9	61	82	31	AF	96	E50	0F6	D7	89	11	F1	78	F2	77	E0	AEF
06C	11	53	1D	F9	61	31	96	94	1C5	0F7	1F	17	8F	28	60	51	39	35	E66
06D	00	08	C4	E9	06	71	44	57	534	0F8	F6	66	47	02	15	D9	17	93	1E2
06E	32	61	88	FF	F8	11	7E	0F	8DE	0F9	F3	4F	6D	6D	61	6E	64	6F	5A1
06F	00	86	5B	FD	B1	08	31	DC	C83	0FA	F1	55	71	F1	78	F2	8F	E0	94B
070	8F	10	63	17	3F	E1	18	D7	016	0FB	C1	08	FE	92	20	34	FF	F4	CFC
071	00	DB	10	87	3E	E1	B3	E6	3BA	0FC	0C	E5	DF	7A	41	1C	41	43	0A0
072	F2	14	2C	44	B4	2B	1F	D0	759	0FD	17	41	37	13	5C	2E	6E	61	41C
073	7F	21	53	11	F5	57	F2	15	AD8	0FE	09	71	70	8F	E9	22	01	18	78E
074	71	91	A7	09	96	92	11	8D	E53	0FF	11	18	BE	90	13	16	D9	E8	B1E
075	70	08	65	A9	DB	10	83	14	1CE	100	D8	4A	80	AF	52	8A	E9	81	ED5
076	C8	FA	F5	31	11	8D	70	06	56A	101	28	12	BF	5B	F5	0D	88	0C	283
077	0C	11	F1	98	F2	11	81	45	8E3	102	E8	16	81	60	11	37	8F	53	5F8
078	1F	55	7F	2A	F2	15	DB	8F	CA6	103	4B	11	0C	1F	17	8F	21	51	979
... continued on next page continued on next page ...									

079	34	42	03	14	C8	F1	06	31	007	104	78	FE	0C	10	11	C8	FB	14	D21
07A	32	61	88	FA	B8	11	52	C8	398	105	B1	13	50	18	FE	0C	10	8F	0B3
07B	A8	DB	17	E1	7F	AF	01	43	751	106	E9	22	01	B3	E6	F2	14	2C	448
07C	10	21	74	13	71	34	20	1F	A9C	107	44	A2	8F	EE	01	08	FE	21	7F2
07D	08	4F	2C	C4	61	81	C1	4E	E37	108	20	8F	3E	32	0B	2E	00	33	B6E
07E	14	D1	61	17	1C	C5	1F	32	1B8	109	5C	F0	06	FC	F8	4C	62	6F	F34
07F	61	83	26	18	8F	AB	81	1D	54A	10A	DC	BF	F6	CB	FF	7F	9B	56	343
080	21	4F	0A	86	01	58	61	50	8B0	10B	06	D4	F6	76	B1	1B	13	51	6CA
081	85	32	B1	71	15	FB	17	B1	C36	10C	12	1C	F1	37	8B	6E	11	35	A51
082	5B	B1	BD	78	F2	15	8B	A1	FEF	10D	10	B3	F2	4A	3D	42	51	4C	DDE
083	21	B1	78	F2	15	CB	17	B1	382	10E	41	40	21	55	70	0A	FA	AF	16A
084	47	CE	1B	98	8F	21	44	8A	72B	10F	22	43	23	A2	AF	70	11	32	4D2
085	A2	18	51	6F	00	17	11	7B	A9C	110	8F	B1	DE	01	33	8F	B1	DE	89D
086	17	B8	55	17	41	37	13	5D	E0C	111	01	04	11	48	FA	0D	E0	13	C14
087	7A	F0	14	31	02	87	51	51	170	112	38	FA	0D	E0	13	20	3		EF5
088	74	1B	99	5F	21	46	E2	1B	502										
... continued in second column, top ...																			