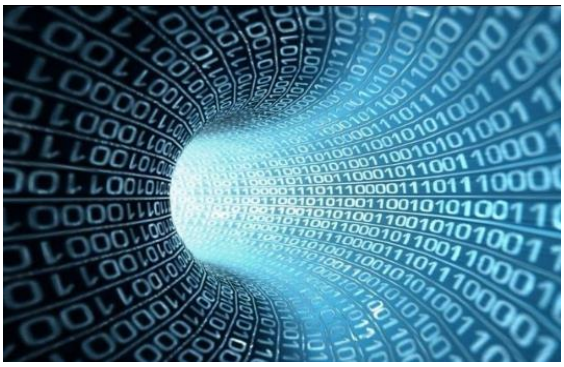
HP-41 Cryptography Module



Overview

This module comprises a selection loosely grouped around the Cryptography subject. Don't expect a new implementation of the Enigma machine (besides that one was eventually defeated anyway) – but a grouping of functions and User programs that touch upon the data scrambling and encoding areas. Some programs have been adapted from the German Vieweg series books, modified to take advantage of the extended functions set.

The Page scrambling and the Password Activation functions were already available in the PowerCL, and have been now included here for all users to enjoy. Remember that they need the target ROM to be in Q-RAM to work properly, as they modify its actual contents. Thus they implicitly require the HEPAX/NoVoRAM, an MLDL2k or the 41-CL to be fully realized.

Amongst the new material, MCODE functions to alter the ALPHA contents by adding or subtracting a certain number of units to each of the characters (constant shift), or by a progressive altering that increases the shift value for character as the text moves on.

Some functions use routines from the ***Library#4 Module*** – which therefore must be installed on your system. There is a check for the Library#4 presence upon initialization (Calculator "ON") that will present a warning message if it's not present:

Both the Library#4 module and its manual can be downloaded from the HP-41 Archive Page, graciously hosted by Warren Furlow. This module is pretty much self-contained, as it includes all the auxiliary functions used in the FOCAL programs. - The only exception is the user program-encrypting program "***CRYPTO***", which requires the original CCD Module to be plugged in .

The table below shows the function names in alphabetical order with a brief description. The Authors and sources are listed below for a complete program description and detailed user instructions in case you're interested.

XROM	Function	Description	Author	Source
10,00	-CRYPTO 41	Section Header	*Ángel Martin*	This project
10,01	ΣDGT	Sums mantissa digits	*Ángel Martin*	RAMPage Module
10,02	AREV	Reverses ALPHA Content	*Frans de Vries*	Data File V10 N8 p8
10,03	CRPTPG _ _ _	Encrypt/Decrypt Page	*Ángel Martin*	PowerCL Extreme
10,04	DSHFT _ _	Down-Shift Alpha (Constant)	*Ángel Martin*	This project – see below
10,05	DSHFT+ _ _	Down-Shift Alpha (Progressive)	*Ángel Martin*	This project – see below
10,06	FLCOPY	Copy ASCII File	*Ángel Martin*	PowerCL Extreme
10,07	FOG	Subroutine for CRPTPG	*Derek Amos*	PPC CJ, V12N5 p3

10,08	NOTA	NOT ALPHA	*Ángel Martin*	This Project – see below
10,09	NOTFL	NOT X-Mem File	*Ángel Martin*	This Project – see below
10,10	RENMFL	Rename File	*Sebastian Toelg*	RAMPage Module
10,11	RETPFL	Retype File	*Ángel Martin*	RAMPage Module
10,12	RGCHR	Registers as Characters	*Ángel Martin*	This Project – see below
10,13	RGNUM	Registers as Numeric	*Ángel Martin*	This Project – see below
10,14	SECURE	Enables password lock	*Nick Harmer*	Data File V9 N5 p12
10,15	UNLOCK	Disables password lock	*Ángel Martin*	PowerCL Extreme
10,16	UNSHFT _ _	Up-Shifts Alpha (Constant)	*Ángel Martin*	This project – see below
10,17	UNSHFT+ _ _	Up-Shifts Alpha (Progressive)	*Ángel Martin*	This project – see below
10,18	WORKFL	Recalls Work File Name	*Ángel Martin*	PowerCL Extreme
10,19	XPASS _	Changes Password	*Nick Harmer*	Data File V9 N5 p12
10,20	-USER PRGM	Checks for Library#4	*Ángel Martin*	This project
10,21	*"A>REG"*	Saves ALPHA in Data Registers	*JM Baillard*	hp-41 Programs
10,22	*"AS>DT"*	Copies Text File to Data File	*Harald Schumny*	Vieweg Book #5
10,23	*"CDA"*	Codes ALPHA	*Unknown*	Swap Disks
10,24	*"CDRGX"*	Codes Registers by X,Y	*JM Baillard*	hp-41 Programs
10,25	*"CRYPTO"*	Encrypts Program (w/ CCD Mod)	*Ángel Martin*	Author's Collection
10,26	*"CYPHER"*	Encrypts Data Registers	*Ángel Martin*	This project – see below
10,27	*"DCA"*	Decodes ALPHA	*Unknown*	Swap Disks
10,28	*"DCRGX"*	Decodes Registers by X	*JM Baillard*	hp-41 Programs
10,29	*"DECPHR"*	Decrypts Data Registers	*Ángel Martin*	This project – see below
10,30	*"DT>AS"*	Copies Data File to Text File	*Harald Schumny*	Vieweg Book #5
10,31	*"INVRG"*	Inverts Register Contents	*Frank Altensen*	Vieweg Book #7
10,32	*"INVRGX"*	Like INVRG on bbb.eee range	*Ángel Martin*	This project – see below
10,33	*"NUMRIK"*	Register Conversion to Numeric	*Martin/Altensen*	Vieweg Book #7
10,34	*"RNDMZ"*	Register Randomization	*Ángel Martin*	This project – see below
10,35	*"SHFLRG"*	Shuffles Mem Registers	*Frank Altensen*	Vieweg Book #7
10,36	*"UNUMRK"*	Undoes Numeric Conversion	*Martin/Altensen*	Vieweg Book #7
10,37	*"URNDMZ"*	Undoes Randomization	*Ángel Martin*	This project – see below
10,38	*"USHFRG"*	Un-shuffles Mem Registers	*Frank Altensen*	Vieweg Book #7
10,39	*"CRPTAS"*	Manages Text File Encryption	*GeirIsene*	Geir's Pages
10,40	*"DECR"*	Decrypts Text File	*GeirIsene*	Geir's Pages
10,41	*"ENCR"*	Encrypts Text File	*GeirIsene*	Geir's pages
10,42	*"VIEWCR"*	Views Encrypted Text File	*GeirIsene*	Geir's Pages
10,43	-CODE THRY	Section Header	*Ángel Martin*	This Project
10,44	2^X-1	Power of 2 minus one	*Ángel Martin*	SandMath Module
10,45	LD2	Digital Logarithm base-2	*Ángel Martin*	This Project – See below
10,46	*"CODING"*	Coding Theory Main	*Michael Schilli*	PRISMA Jan 1990, p23
10,47	*"HUFFM"*	Huffman Method	*Michael Schilli*	PRISMA Jan 1990, p23
10,48	*"SHANN"*	Shannon-Fano Method	*Michael Schilli*	PRISMA Jan 1990, p23
10,49	*"PUSH"*	Subroutine for CODING	*Michael Schilli*	PRISMA Jan 1990, p23
10,50	*"POP"*	Subroutine for CODING	*Michael Schilli*	PRISMA Jan 1990, p23
10,51	*"*SZ"*	Size Check	*Michael Schilli*	PRISMA Jan 1990, p23
10,52	*"SEMIO"*	Semiotic Analysis	*Martin/Altensen*	Vieweg Book #7
10,53	*"VREG"*	Views bbb.eee registers	*Ángel Martin*	RAMPage Module

10,54	A<>RG _ _	Swaps ALPHA and Registers	*Ángel Martin*	RAMPage Module
10,55	ARCLI	Alpha RCL Integer	*W&W GmbH*	OS/X Manual
10,56	CEIL	Ceiling Function	*Ángel Martin*	SandMath Module
10,57	FLOOR	Floor Function	*Ángel Martin*	SandMath Module
10,58	PMTA	Prompt ALPHA Text	*W&W GmbH*	OS/X Manual
10,59	RAND	Random Number from Seed	*Håkan Thörngren*	PPC CJ V13N4 p31
10,60	SEED	Enters Seed	*Håkan Thörngren*	PPC CJ V13N4 p31
10,61	MREV	Mantissa Digit Reversal	*Ángel Martin*	Recurse Module
10,62	X>$	Numeric to String	*VL Electronics*	HEPAX Module
10,63	ADR _ _ _ _	Address Coding	*Ángel Martin*	This module

`

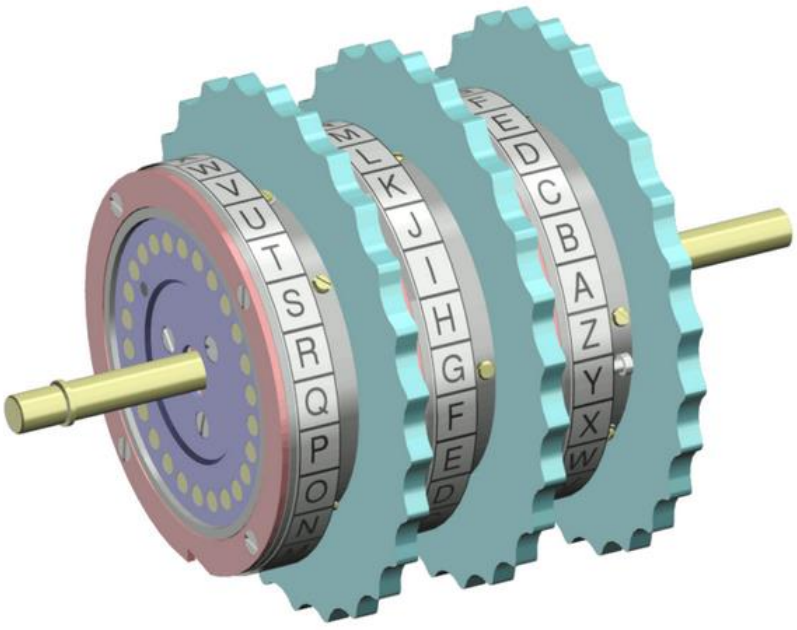


Image Credit: Emigma rotor set Wapcaplet.
https://brilliant.org/wiki/enigma-machine/

1 – Crypto Function Set

Below is a brief description for some of the functions of the module for your convenience – As always, you're encouraged to check the documentation provided in the linked documents for all the other functions and programs.

DSHFT_ and USHFT_ ;Char. Constant Shift

These functions apply a *constant shift* to encode all characters in the ALPHA registers, given by the value in the prompt. The shift is either added to or subtracted from the original char value, depending on the function used. They are inverse from one another, so you can recover the text encoded by one using the other with the same parameter.

Like it is the case with the other prompting functions, they will take the prompt argument from the X registers if executed as part of a user program.

Examples: Convert the following text in ALPHA by adding three to the character values, then recover the initial text subtracting the same amount from them.

ALPHA, "MARY HAD A LITTLE LAMB", ALPHA

```
USHFT 03 =>"**PDU/#KDG#D#OLWWOH#ODPE"
DSHFT 03 =>"MARY HAD A LITTLE LAMB"
```

DSHFT+_ and USHFT+_ ; Char. Progressive Shift

These functions apply a *progressive shift* to encode the characters in ALPHA, beginning with the value entered in the prompt. The shifts are either added to or subtracted from the original char value, depending on the function used. They are inverse from one another, so you can recover the text encoded by one using the other with the same parameter.

Like it is the case with the other prompting functions, they will take the prompt argument from the X registers if executed as part of a user program.

Examples: Convert the same text from the example before using a progressive shift to the character values, starting with the value 003. Undo the changes by applying a progressive reduction to the converted text.

ALPHA, "MARY HAD A LITTLE LAMB", ALPHA

```
USHFT+03 =>"**eX**4CSU0 P.YU_^UM'RFQE"
DSHFT+ 03 =>"MARY HAD A LITTLE LAMB"
```

NOTA and NOTFL ; Inverting Data

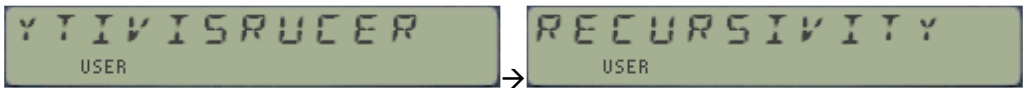
These functions use the simplest pattern for a character code modification; i.e. a logical NOT operation, either on the contents of the ALPHA registers or in an ASCII file located in Extended-memory. The functions will therefore invert the text and can be used twice to recover the original contents again.

NOTA will not change the NULL characters – and therefore the encoded text will have the same length as the original. This is not the case for the shifting functions, which work on a complete register basis – and thus the encoded text will have a total length multiple of seven.

NOTFL is the equivalent function for ASCII files. The ASCII file name is expected to be in ALPHA before calling **NOTFL**. Be aware that it will also invert the record-definition characters within the ASCII file records, and therefore *you should not try to Edit or View the file contents while it is encrypted!* Refer to the user program "***CRPTAS"*** for a more flexible encryption of ASCII files that respects the control characters and therefore is compatible with the file editing functions like ED.

AREV and A<>RG _ ; ALPHA Utilities

AREV is a super-fast function that will reverse the complete contents in ALPHA (up to 24 characters), turning it into its mirror-image. For instance:



A<>RG will dump the ALPHA contents into four consecutive registers, starting by the register number entered in its prompt. Use it to temporarily save ALPHA and restore it back with a second execution of the function using the same parameter. Note that the data in the registers should not be recalled, as that would normalize the contents and therefore void the original formatting.

Like it is the case with the other prompting functions, they will take the prompt argument from the X registers if executed as part of a user program.

Example.- The short program below combines four of the functions described above to create a convoluted encryption of the ALPHA register, then reverses itself to undo the encoding. Note how in a program the shifting functions take the shift value from the X register – and not the prompt.

Line	Instruction	Line	Instruction
01	**LBL "CD"**	08	**LBL "DCD"**
02	**AREV**	09	**NOTA**
03	3	10	3
04	**USHFT+**	11	**DSHFT+**
05	**NOTA**	12	**AREV**
06	AVIEW	13	AVIEW
07	RTN	14	END

This idea is the foundation of the user programs **CYPHER** and **DECPHR**, which will be described later in the manual.

RGCHR and RGNUM ; Changing Register Data Types

This pair of functions do a mass-conversion of all data registers; changing the data type for the values stored in them - to Alpha strings and back to numeric. They work on ALL data registers as defined by the calculator SIZE. If a given register is already in the target format it will be ignored. A related function is **X>$**, which does the conversion to ALPHA data type on the X-Register only.

Note that while the conversion to ALPHA DATA (function **RGCHR**) is always a safe one, the reverse one (function **RGNUM**) is likely to result into a non-Normalized Number (NNN) and therefore the value will be erased if you use the standard memory functions on the corresponding register, due to the normalization routines called by them. For a more comprehensive data type conversion you should refer to the user programs "**NUMRIK**" and "**UNUMRK**" described later in this document.

RETPFL and RENMFL ; Changing X-Mem Files

Another simple way to disguise the information contained in an ASCCI file is by modifying the file type so that the respective file editors (and functions) won't be able to open it or view it. This can be done with **RETPFL**, which will change the file type to the value in the X register (x<=14). You should avoid using other types reserved for other file types, such as DATA (x=2) and Program (x=1).

You can also use **RENMFL** to further clear your tracks after changing the file type – so there's no resemblance to the original ASCII file even in the new file name.

Be careful not to attempt file editors with "masked" file types!

FLCOPY and WORKFL ; Copying Files and WorkFile

There is no function in the X-Functions Module that allows for a direct File Copy – but this limitation is overcome with **FLCOPY**. You can use it to copy any file type to another of the same type, *provided that the destination file has already been created in X-Memory – obviously as a blank file but with the same size*. The syntax for **FLCOPY** follows the intuitive format "FROM,TO" string in Alpha.

Lastly, **WORKFL** will retrieve the name of the current work file to ALPHA – so you don't need to store it during intermediate calculations that would modify the ALPHA registers.

Even if not directly related to cryptography, these functions become very useful for user programs to facilitate the FOCAL code.

ΣDIG and MREV ; Mantissa functions.

This pair of minimalistic routines add some tricks to the masking and disguising toolkit. You can use them to further enhance the camouflaging tactics for data values in X, and by extension all data registers as well.

ΣDIG pushes the stack and puts the sum of all mantissa digits in X,

Example: PI, XEQ "ΣDIG" => 40.000000000

Header	AE01	087	"G"	
Header	AE02	009	"i"	**Sum of Mantissa Digits**
Header	AE03	004	"D"	
Header	AE04	04E	"Σ"	Angel Martin
ΣDIG	AE05	0B1	?NC XQ	Mantissa Digit SUM
	AE06	10C	->432C	[SDGT4]
	AE07	17D	?NC GO	[BIN-BCD] plus [RCL]
	AE08	0C6	->315F	[ATOX20]
ΣDGT4	432C	0F8	READ 3(X)	
	432D	00E	A=0 ALL	initial sum =0
	432E	39C	PT= 0	
NXTDGT	432F	33C	RCR 1	
	4330	3C6	RSHFC S&X	
um	4331	3C6	RSHFC S&X	
	4332	146	A=A+C S&X	add to previous sum
	4333	3DC	PT=PT+1	
	4334	0D4	?PT= 10	
	4335	3D3	JNC -06	[NEXTD]
	4336	3E0	RTN	

MREV reverses the mantissa digits of th e value in X – leaving the sign and exponent unaltered.

Example: PI, XEQ "MREV" => 4.562951413

Header	A9FF	096	"V"	
Header	AA00	005	"E"	**Mantissa Reversal**
Header	AA01	012	"R"	
Header	AA02	00D	"M"	Ángel Martin
MREV	AA03	0F8	READ 3(X)	
	AA04	10E	A=C ALL	safekeep C.MS and C.X
	AA05	2DC	PT= 13	counter
	AA06	250	LD@PT- 9	will do 10 times
	AA07	01C	PT= 3	fixed position
	AA08	3FA	LSHFA M	scroll A.M left
	AA09	102	A=C @PT	copy digit
	AA0A	3DA	RSFHC M	next C.M digit
	AA0B	27E	C=C-1 MS	decrease counter
	AA0C	3E3	JNC -04	no, do next
	AA0D	0AE	A<>C ALL	yes copy result to C.M
	AA0E	331	?NC GO	Overflow, DropST, FillXL & Exit
	AA0F	002	->00CC	[NFRX]

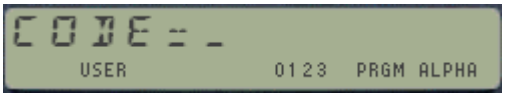
FOG and CRPTPG ; Q-RAM Encryption

If you're concerned about the security of the data and programs held in your RAM pages here is the ultimate encryption facility to completely cover your tracks and protect the system to the paranoia stage.

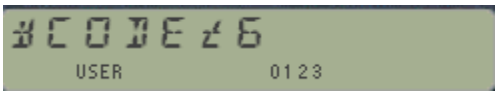
FOG will scramble the RAM contents using a 6-characters long encryption key provided in ALPHA, starting from the address in the S&X field of X (thus a NNN is expected), and until the bottom of that page. Repeating the operation with the same encryption code will restore the contents to its original state, so the operation is reversible – as long as you remember the key used to encrypt it in the first place.

CRPTPG is a nice and easy driver program for **FOG** that takes care of preparing the required inputs for you. No additional "precautions" are added, so the "ADR _ _ _ _" input will accept any HEX characters and not only valid addresses.

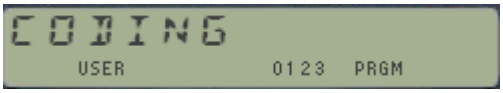
The program will prompt for a user key, which must be 6-chars long exactly. This gets enforced by the following error message when needed:



and



As these functions only operate in RAM the OS area is safe, even if you attempt to encrypt it. Ditto for every page plugged to a module in Flash memory – but watch out for the RAM-plugged pages (like HEPAX RAM, or any other module you have residing in sRAM). That's why the confirmation message "OK? Y/N" will also be prompted when calling this function – even in a program.



Needless to say, things can get hairy pretty quickly if you mess with critical areas, like the polling points. **FOG** will not modify the contents of locations 0xpFF4 and above within the page, but that doesn't guarantee a trouble-free result – because *if polling points are active who knows what will be there where they're pointing at AFTER the encryption!*

ADR ; Address Coding

This small routine is used in CRPTPG to prompt for the start address in the page to encrypt. You can also use it in your own routines. The result is left in register Q as an NNN.

SECURE, **UNLOCK**, and **XPASS** ; Calculator Security

Here we have a nice practical application of advanced system control. Use these functions to manage a password-protection scheme for your machine – so nobody without authorized access can use it.

They were published back in 1990 (Data File V9 N5 p12) by Nick Harmer, and implemented in Q-RAM devices (a.k.a MLDL). Obvious caveat there was that removing the MLDL from the machine dismantled the whole scheme – and the same applies in this implementation..

The protection works as follows:-

1. Function **SECURE** activates the security by setting the protection flag. The execution also switches off the machine. This sets up a process executed on each CALC_ON event, causing to prompt the user for the password during the start-up process.
2. Function **UNLOCK** deactivates the security by clearing the protection flag.
3. Function **XPASS** allows the user to change the password from the default one to his/her favorite one. The length of the password is limited to six (6) characters.

Enter code (up to 6 chrs. long) and end with [**R/S**]

Inputting the password is very simple but very unforgiving as well: at the prompt "PASSWORD=?" just type the letters one by one until completing the word, and you're done. If you make a mistake the machine will switch itself off and it'll be "groundhog day" all over again – until you get it right.

Each keystroke will be acknowledged by a short tone, but no change to the display – so nothing like "*****" as you type the word. If the wrong letter is entered a lower-pitch sound will be heard and the calculator will go to sleep.

Be especially careful when entering a new password code – as there is no repeat input to confirm the entry; so whatever key combination you type will be taken when ending the sequence with R/S. The initial password ("factory default", so to speak) is "*CACA*".

Enter code (up to 6 chrs. long) and end with [**R/S**]

2 – Related Math Functions

2^X-1 and LD2 ; base-2 Math.

Two small utility functions to calculate digital logarithm (base 2) and powers of 2 with more accuracy than the standard functions – using the 13-digit routines in the OS.
Both functions take the argument from the X register and save it to LastX, and the result is placed in X replacing the original argument. No other stack or data registers are used.

Being quasi-inverse of one another, they verify the relation: x = LD2 [2^X-1(x) + 1]

STACK	INPUT	OUTPUT
Y	y	y
X	x	LD2(x) / 2^X-1 (x)
L	l	x

CEIL and FLOOR ; Ceiling and Floor integers

The floor and ceiling functions map a real number to the largest previous or the smallest following integer, respectively. More precisely, floor(x) = [x] is the largest integer not greater than x and ceiling(x) =]x[is the smallest integer not less than x.

This implementation uses the native MOD function, through the expressions:

CEIL (x) = [x – MOD(x, -1)]; and FLOOR (x) = [x – MOD(x, 1)].

SEED and RAND ; Random Numbers.

The Crypto ROM includes its own facility to handle Random and pseudo-random numbers. The model uses the combination of a **SEED** value, which can be entered by the user or taken from the current time using the Timer in the CX. Then the **RAND** function obtains a derived pseudo-random number based on the current seed.

The application of pseudo-random numbers to cryptography is obvious is we associate the user-provided seed value to a given user key for an encoding process, and the function **RAND** to the different factors derived from the seed. Note that *the seed value must be between 0 and 1* for**SEED** to work properly.

These functions are implemented using a dedicated buffer to store the seed and current random values. The buffer is created automatically by the module upon start-up when the calculator is switched on. The buffer id# is number "9".

Example: Register Randomization.

The listing below combines the functions described above to randomize all data registers by means of a direct multiplication by a random number derived from a user seed. It then reverses itself to undo the encoding. Only the core section is listed – but note how no data registers are used for the program itself and thus all main memory will be converted.

Line	Instruction	Line	Instruction	Line	Instruction
01	**LBL "RNDMZ"**	13	**LBL "URDMZ"**	25	ENTER^
02	XEQ 05	14	XEQ 05	26	LOG
03	LBL 00	15	LBL 02	27	**CEIL**
04	**RAND**	16	**RAND**	28	10^X
05	ST* IND Y	17	ST/ IND Y	29	/
06	RDN	18	RDN	30	**SEED**
07	ISG X	19	ISG X	31	SIZE?
08	GTO 00	20	GTO 02	32	E
09	LBL 10	21	GTO 10	33	-
10	"READY"	22	LBL 05	34	E3
11	PROMPT	23	"KEY=?	35	/
12	RTN	24	"PROMPT	36	END

Using repeat calls to the **RAND** function within the register loop ensure that the "key" value used varies from register to register, so it cannot be back-calculated by direct register comparisons.

Be aware that for the decryption process you'll need to provide the same key used for the encryption (a.k.a. the "anchor" for the initial step – otherwise there won't be possible to recover the initial values.

You can always use the program "**VREG**" to review the actual register contents entering the control word "bbb.eee.ii" in the X register.

3 – User programs.

CRPTAS ; ASCII Files Encryption (*by Geir Isene*)

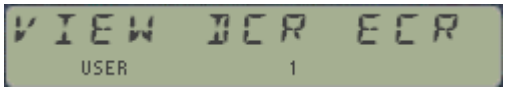
The Function **NOTFL** was described in the first section of the manual – indicating that there was an instable encryption since it also modifies the control characters within the file records and thus it couldn't be reviewed. The user programs described below overcome that limitation and provide a general-purpose capability as follows.-

These programs encrypt and decrypt an ASCII Extended Memory (XM). You can also temporarily view an encrypted file without substituting the file with the decrypted version.

The CRYP program implements the Vigenere cipher with a key of your choice of up to 300 characters. By default it uses the range of characters from ASCII code 32 (space) to ASCII code 90 ("Z"), but you can choose any range above ASCII code 32 – for example a range value of 90 to include lower case characters (ASCII code 122 is "z").

If the key is at least as long as the file to be encrypted, you will actually get perfect security for the encrypted file. You will have what is known as the One-time pad.

CRYPT is a menu-driven program, which present the following choices:



Here are the three functions implemented:

Function	Description
ENCR	Encrypts an ASCII file. The file name must be in Alpha when you execute ENCR. The program first prompts for the character range (default 58 – press R/S to accept the default). It then prompts for the key (Alpha is set to ON). Enter the key to use for the encryption 24 characters at the time. As long as you fill the Alpha register with characters for the key, it will keep asking for more key – until you enter fewer than 24 characters – then it will commence to encrypt the file. The program resizes the memory to accommodate for a large key if necessary. After encrypting the file with the key, you get the message "DONE" along with a beep. All traces of the key have then been removed. Pressing R/S again will launch the EDitor (**ED**) with the encrypted file.
DECR	Decrypts the file (name in Alpha. Prompts for the range and the key (use the same range and the same key as when you encrypted the file. Again you will get a "DONE" and a beep when the process is completed. Pressing R/S again will launch the EDitor (ED) to let you view and edit the decrypted file.
VIEWCR	Lets you view an encrypted file. Instead of actually decrypting the whole file, you get to view each successive record. The program sounds a Tone 9 upon showing each record. The file remains encrypted in Extended Memory. After the last record is viewed, the program displays "DONE" and gives a beep. All traces of the key is removed.

Program listing.-

```
01*LBL "CRPTAS"
02*LBL D
03 "VIEW DCR ECR"
04 AVIEW
05 CLA
06 STOP
07 GTO D
08*LBL "VWCR"
09*LBL A
10 2
11 GTO 03
12*LBL "DECR"
13*LBL C
14 1
15 GTO 03
16*LBL "ENCR"
17*LBL E
18 0
19*LBL 03
20 X<>F
21 CLX
22 SEEKPTA
23 58
24 "RANGE? (58)"
25 PROMPT
26 STO 03
27 4.003
28 STO 01
29*LBL 10
30 "KEY? "
31 PMTA
32 24
33 ALENG
34 X=Y?
35 SF 05
36  E3
37 /
38 ST+ 01
39 RCL 01
40 FRC
41  E3
42 *
43  E
44 +
45 SIZE?
46 X<>Y
47 X>Y?
48 PSIZE
49*LBL 00
50 ATOX
51 STO IND 01
52 ISG 01
53 GTO 00
54 FS?C 05
55 GTO 10
56 XEQ 13
57*LBL 01
58 SF 25
59 GETREC
60 FC?C 25
61 GTO 14
62 ALENG
63 STO 02
64*LBL 02
65 ATOX
66 32
67 -
68 RCL IND 01
69 FS? 00
70 -
71 FC? 00
72 +
73 RCL 03
74 MOD
75 32
76 +
77 XTOA
78 ISG 01
79 GTO 12
80 XEQ 13
81*LBL 12
82 DSE 02
83 GTO 02
84 FS? 01
85 TONE 9
86 FS? 01
87 PROMPT
88 FS? 01
89 GTO 01
90 DELREC
91 INSREC
92 GTO 01
93*LBL 13
94 RCL 01
95 FRC
96 4
97 +
98 STO 01
99 RTN
100*LBL 14
101 CF 00
102 CF 01
103 RCL 01
104 FRC
105  E
106 +
107 CLRGX
108 CLST
109 "DONE"
110 BEEP
111 AVIEW
112 CLA
113 WORKFL
114 STOP
115 SEEKPT
116 ED+
117 END
```

CDRGX and **DCRGX** ; Coding Text in Registers (*by JM Baillard*)

These programs allow coding and decoding a text message that has been stored into data registers by groups of 1 to 6 characters.The key is simply a random seed input by the user in the X register. The same one is used for coding and decoding.When the program stops, the coded message has replaced the original (or vice versa). The register range must be entered in the Y-register to define the length of the message.

STACK	INPUTS	OUTPUTS
Y	bbb.eee	/
X	key	bbb.eee

Example: You have stored:

"MYNAME" in register R11
"ISBOND" in register R12
"JAMES " in register R13
"BOND" in register R14

If your key is r = 1 (a more sophisticated key is recommended...) then:

11.014 ENTER^ 1 XEQ "CDRGX" => 11.014 --- Execution time = 17s ---

The coded message is now in registers R11 thru R14 (many of the characters are displayed as starbursts). To decode this message, and since the register range is already in the stack:

ENTER^ , 1, R/S => 11.014 and the original alpha strings are in R11 to R14 again.

Remarks:

- avoid the null character since it disappears when shifted into the leftmost position; otherwise, all the characters with an ASCII code from 1 to 255 are allowed.
- These programs use the random number generator:{ R-D FRC }, so choose r so that 180*r/PI is not an integer.
- Using this program several times with several keys could produce an almost unbreakable encryption.
- For a 600 character message (stored in 100 registers), execution time = 7mn40s.

Program listing.-

```
01*LBL "CDRGX"
02 CF 01
03 GTO 05
04*LBL "DCRGX"
05 SF 01
06*LBL 05
07 STO 00
08 X<>Y
09 STO 01
10 STO 03
11 255
12 STO 02
13*LBL 01
14 CLA
15 ARCL IND 01
16 ALENG
17*LBL 02
18 ATOX
19 RCL 00
20 R-D
21 FRC
22 STO 00
23 RCL 02
24 *
25 INT
26 FS? 01
27 CHS
28 +
29 RCL 02
30 MOD
31 X=0?
32 X<> L
33 XTOA
34 X<>Y
35 DSE X
36 GTO 02
37 ASTO IND 01
38 ISG 01
39 GTO 01
40 RCL 03
41 CLA
42 END
```

A>REG ; Storing a message into contiguous registers.

Example: You want to store the confidential message: "THE GARDEN PEAS ARE SIMMERING IN THE SAUCEPAN" into registers R00 , R01 , ...

STACK	INPUT	OUTPUT
X	bbb	bbb.eee

0 XEQ "A>REG" => the HP-41 displays "1? "

and the alpha keyboard is activated. Now key in:
"THE GARDEN PEAS ARE SIMM"

you hear a TONE since there are now 24 characters in the alpha "register".
Press R/S and the HP-41 displays "2?", then key in:"ERING IN THE SAUCEPAN"

press R/S the HP-41 displays "3?". Simply press R/S without any entry the HP-41 returns the control number 0.007. Your message has been stored into registers R00 thru R07

Program listing.-

```
01*LBL "A>REG"
02 ENTER^
03 AON
04  E
05 ST- Y
06 GTO 02
07*LBL 01
08 ISG Z
09 CLX
10 ASTO IND Z
11 ASHF
12 CLX
13 ALENG
14 X#0?
15 GTO 01
16 SIGN
17 +
18*LBL 02
19 RCLFLAG
20 FIX 0
21 CF 29
22 " "
23 ARCL Y
24 "`?"
25 STOFLAG
26 STOP
27 FS?C 23
28 GTO 01
29 AOFF
30 RDN
31 CLX
32  E3
33 /
34 +
35 CLA
36 END
```

INVREG and **INVRGX**; Registers Inversion *(by F. Altensen)*

This program does basically an inversion of the Text data contained in Data Registers simply by using a character mirror image method. It works on the entire data register range currently configured in the calculator - but only affecting those containing ALPHA data.

Example: With the following strings stored in registers R07 – R09:

Before	After:
R07 = "ROGER1"	R07 = "1REGOR"
R08 = "CHARLY"	R08 = "YLRAHC"
R09 = "OSKAR9"	R09 = "9RAKSO"

A second execution restores the original data in the registers.

The original program by Frank Altensen only used basic HP-41C functions (plain model!), so you have an example of bare-bones ingenuity. The drawbacks were longer execution times and data registers usage {R00 – R06}.

Using **AREV** the execution improves significantly, and without internal usage of data registers. See the modified listing below that also offers an option for a register range instead, defined by the control word ***bbb.eee*** in the X register.

Line	Instruction	Line	Instruction
01	**LBL "INVRG"**	12	X#0?
02	SIZE?	13	GTO 01
03	E	14	ARCL IND Y
04	-	15	**AREV**
05	**E3**	16	ASTO IND Y
06	/	17	LBL 01
07	**LBL "INVRGX"**	18	**RDN**
08	LBL 00	19	ISG X
09	CLA	20	GTO 00
10	RCL IND X	21	**END**
11	SIGN		--

Note how the inversion is only performed on ALPHA data, checking for the result of the SIGN operation.

SHFRG and **USHFRG** ; Registers Shuffling *(by F. Altensen)*

Rather than modifying the data registers contents, these programs base their approach in a modification of the message parts- i.e. shuffling the data registers according to a scheme derived from the user key entered at the beginning of the program. The original message is pierced back together by undoing the shuffling – using the same user key.

Remarks:

- Registers {R00 – R06} are used internally by the program.
- It works on the complete register range as defined by the SIZE of the calculator.
- The user key is the seed for a series of pseudo-random numbers used for the shuffling order
- The user key can have values from 1 to 9999999999, but *this is a very slow process* – which grows exponentially with the length of the supplied key.

The pseudo-random algorithm used here and all throughout the module (including the MCODE function **RAND** itself) is the same one described in the PPC ROM manual, whereby:

RAND(x) = FRC [9821 * x + 0,211327]

Program listing:

```
01*LBL "SHFRG"
02 SF 05
03 GTO 05
04*LBL "USHFRG"
05 CF 05
06*LBL 05
07 "CODE=?"
08 PROMPT
09 "DE"
10 FS? 05
11 CLA
12 "`CODING..."
13 AVIEW
14 STO 01
15 STO 02
16 STO 03
17 CLX
18 STO 00
19 FC? 05
20 GTO 01
21*LBL 00
22 XEQ 10
23 STO 05
24 XEQ 10
25 STO 06
26 RCL IND 05
27 X<> IND 06
28 STO IND 05
29 DSE 02
30 GTO 00
31 ΣREG 01
32 CLΣ
33*LBL 09
34 "READY"
35 PROMPT
36 GTO 05
37*LBL 10
38 RCL 00
39 9821
40 *
41 .211327
42 +
43 FRC
44 STO 00
45 SIZE?
46 7
47 -
48 *
49 7
50 +
51 RTN
52*LBL 01
53 XEQ 10
54 STO 05
55 XEQ 10
56 STO 06
57 DSE 01
58 GTO 01
59 RCL IND 05
60 X<> IND 06
61 STO IND 05
62 CLX
63 STO 00
64 DSE 03
65 ""
66 RCL 03
67 STO 01
68 ISG 01
69 CLX
70 DSE 01
71 GTO 01
72 GTO 09
73 END
```


NUMRIK and **UNUMRK** ; Numeric Conversions.

These programs provide general-purpose data type conversion functionality for all values in the data registers. They basically allow you to change text data into an encrypted equivalent numeric value, which can be decrypted later on to recover the original message.

Because the encrypted values are properly normalized numbers, these programs do not have the limitation discussed before on the **RGNUM** function description, thus you can review the encrypted values without any risk of data loss due to normalization. Note however that the programs use R00 to R10, *the valid register range for the conversion starts with R11*.

Example: Encrypt the text message "ONCE UPON A TIME IN THE WEST", stored in data registers R11 to R15.

First we'll enter the message using **A>REG**, with 11 in X and typing the text ad libitum.

After executing ***NUMRIK*** the registers contain the following values:

R11 = 2,4231214 30
R12 = 2,5242332 32
R13 = 2,9182214 18
R14 = 3,2233229 14
R15 = 3,2323214 29

And running the inverse process with ***UNUMRK*** returns the original data as follows:

R11 = ONCE U
R12 = PON A
R13 = TIME I
R14 = N THE
R15 = WEST

Program listing.-

```
01*LBL "NUMRIK"
02 CLX
03 STO 03
04 FIX 0
05 11.1
06 STO 00
07*LBL 02
08 SF 25
09 RCL IND 00
10 SIGN
11 FC?C 25
12 GTO 05
13 X#0?
14 GTO 11
15 CLA
16 ARCL IND 00
17 AREV
18 10.004
19 STO 01
20*LBL 04
21 ATOX
22 55
23 -
24 X<0?
25 32
26 STO IND 01
27 DSE 01
28 GTO 04
29 5.01
30 STO 01
31 10.00002
32 STO 02
33*LBL 07
34 RCL IND 01
35 RCL 02
36 INT
37 2
38 -
39 10^X
40 *
41 ST+ 03
42 ISG 01
43 ""
44 DSE 02
45 GTO 07
46 RCL 03
47 ENTER^
48 LOG
49 INT
50 10^X
51 /
52 RCL 10
53 10^X
54 *
55 VIEW 00
56 STO IND 00
57*LBL 11
58 CLX
59 STO 03
60 ISG 00
61 GTO 02
62*LBL 05
63 FIX 3
64 "READY"
65 PROMPT
66*LBL "UNUMRK"
67 11.1
68 STO 00
69*LBL 08
70 5.009
71 STO 04
72 10.00002
73 STO 03
74 SF 25
75 RCL IND 00
76 FC?C 25
77 GTO 05
78 X=0?
79 GTO 12
80 STO 01
81 LOG
82 INT
83 55
84 +
85 STO 05
86 CLA
87 RCL 01
88 ENTER^
89 LOG
90 INT
91 10^X
92 /
93  E1
94 /
95 STO 01
96*LBL 09
97 RCL 01
98  E2
99 *
100 STO 06
101 INT
102 32
103 X=Y?
104 GTO 00
105 RDN
106 55
107 +
108*LBL 00
109 XTOA
110 RCL 06
111 FRC
112 STO 01
113 ISG 04
114 ""
115 DSE 03
116 GTO 09
117 RCL 05
118 XTOA
119 VIEW 00
120 ASTO IND 00
121*LBL 12
122 ISG 00
123 GTO 08
124 END
```

CRYPTO; Program Encryption (*w/ the CCD Module*.)

If you happen to have the **CCD Module** around (a sure fact if you own the 41-CL) you may also want to try this user program that encrypts programs (but itself) placed in RAM. The approach is simple a byte inversion of all the program bytes - except the first global label and its END – so it'll be rendered nonsensical even if legible (i.e. non-private). A second execution of the program undoes the encryption, restoring the original code.

CRYPTO expects the name of the user program to be encrypted in the ALPHA register. That program must reside in RAM for obvious reasons. Depending on the program structure some issues may arise, such as the case of multiple global labels that impact the final END. Because of this the program steps may be gibberish if you review them during their encrypted state.

So here you have it, perhaps more of an academic example than a useful application but illustrates the usage of several CCD Functions quite nicely.

As mentioned in the introduction, this is the only program that requires auxiliary modules being plugged into the calculator – not counting the Library#4 itself.

Program listing.-

```
01*LBL 92
02 RDN
03 A-
04 A-
05 DSE Y
06 DSE Y
07 PEEKB
08 240
09 X<=Y?
10 GTO 01
11 RDN
12 RDN
13 GTO 06
14*LBL 01
15 -
16 CHS
17 ST+ Z
18 A+B
19 GTO 06
20*LBL "CRYPTO"
21 CF 00
22 .
23 WSIZE
24 UNS
25 PPLNG
26 PHD
27*LBL 00
28 PEEKB
29 205
30 X>Y?
31 SF 00
32 CLX
33 192
34 X>Y?
35 CF 00
36 RDN
37 FS?C 00
38 GTO 92
39 240
40 X<=Y?
41 SF 00
42 CLX
43 63
44 X>Y?
45 SF 00
46 CLX
47 50
48 X>Y?
49 CF 00
50 CLX
51 16
52 X>Y?
53 SF 00
54 RDN
55 FC?C 00
56 NOT
57 POKEB
58 RDN
59*LBL 06
60 A-
61 DSE Y
62 GTO 00
63 END
```


CYPHER and DECPHR ; All together now.

If you read the application example in the description of **AREV** you'd already guess what these pair of routines do: data registers encryption using a sequential combination of functions to erase our tracks more effectively, and come up with a pretty unbreakable scheme (even if nothing really is!) .

The idea starts by applying **ΣDGT** to the key (call it a scaling, or maybe protecting the key from the user himself?), followed by a consecutive application of **AREV** and **USHFT+** in the coding phase; and the reverse sequence in the decoding phase – **DSHFT+** and **AREV**. The progressive shifting is preferred to the constant one as it provides a harder coding.

Note that the shifting will ignore the NULL characters – which are an integral part of the ASTO and ARCL functions! (ever before wondered why only six Alphabetical characters can be stored in a data register, if there is room for seven?)

Note that these programs use R00 to store the user-provided key. Also that like with the other programs dealing with message encryption, only registers containing Alpha Data will get Encrypted/Decrypted.

Program listing.-

```
01*LBL "CYPHER"
02 SF 00
03 GTO 00
04*LBL "DECPHR"
05 CF 00
06*LBL 00
07 "KEY=?"
08 PROMPT
09 ΣDGT
10 STO 00
11 SIZE?
12  E
13 -
14  E3
15 /
16  E
17 +
18*LBL 02
19 RCL IND X
20 SIGN
21 X#0?
22 GTO 01
23 CLX
24 RCL 00
25 CLA
26 ARCL IND Y
27 FS? 00
28 AREV
29 FS? 00
30 USHFT+
31 FC? 00
32 DSHFT+
33 FC? 00
34 AREV
35 ASTO IND Y
36*LBL 01
37 RDN
38 ISG X
39 GTO 02
40 END
```

4 – Code Theory.

CODING, SHANN and HUFFM ; *(by M. Schilli)*

Leaving the biggest for last… The Crypto ROM includes a couple of programs related to the Code Theory field. Of those the more distinct one is also the biggest of the module: **CODING** and its subroutines to perform Shannon-Fano and Huffman coding of a given coding scheme. They were written by Michael Schilli and published in PRISMA, the German's user club Magazine. You should refer to the original article for description and user instructions.

Coding Theory. (by Michael Schilli)

PRISMA Magazine 1/90, pg. 23

1.- General Overview

The encoding method is used whenever it is necessary to uniquely map individual symbols of a source character set Q (e.g. {A,B, ... G}) to elements of a target character set Z (e.g. {[0,1]).
In this case, as given in the example, the character set Q to be mapped may contain more elements than the target character set Z: In this case, each element of Q is mapped to a combination of multiple cells of the set V()(l Z (for example, as in Fig.l).

A typical application example from practice is the adaptation of a news source which contains "n" different symbols with the probabilities of occurrence p(Q1), p(Q2), ... ,p(Qn) can be emitted. On a binary channel, for example, this can be an electrical line, which is operated with the digital signals "0"
and "1" and can transmit the emitted symbols of the source over a long distance.

Each emitted symbol from Q is now assigned a particular sequence of zeros and ones is now assigned to each emitted symbol from Q and this is then fed into the channel. The manner of this assignment, i.e. the mapping rule, can now be realized very differently: Besides the assignment by numbering with dual numbers (as shown in Fig. 1), there are several relevant methods, of which two (the Shannon-Fano and the Huffman method) will be presented here. Both methods additionally take into account the occurrence probabilities of the source symbols - it is important to know that for an efficient signal transmission it is crucial to keep the average number of bits to be transmitted per transmitted source signal as small as possible.

For this reason, one assigns very short code words to frequently occurring interfering signals, and, only if necessary longer ones. It must be noted, however, that no code word that occurs may coincide with the beginning of another code word, otherwise a downstream decoder would not be able to clearly separate the words sent clearly separate the words sent. The quality of this code is reflected in the value of the relative redundancy "r" (excess, dependency) and the entropy H:

$$r = \frac{E-H}{E} \qquad H = \sum_{i} \left[p(Qi) * ld \frac{1}{p(Qi)} \right]$$

(good coding: r -> 0), where E is the information content of the code, i.e. the average number of transmission bits per transmission symbol: "m") and H represents the entropy of the source (disorder, becomes maximum when the symbols are equally divided) - where ld = digital logarithm base 2.
The occurrence probabilities of the source symbols in our example are given as follows:

P(A) = 0.41
P(B) = 0.16
P(C) = 0.15
P(D) = 0.13

PIE) = 0.11
p(F) = 0.03
p(G) - 0.01

Figure 1 shows the resulting values for a "numerical" coding are listed.

For the coding shown in Fig. 2 the Shannon-Fano method was used. Due to the fact that the dominant symbol A requires only two bits for transmission, the average codeword length "m" decreases significantly and thus the relative redundancy r.

By applying the Huffman method, the best possible code is obtained, i.e. the code with the lowest redundancy (Fig. 3.).

A: 000		A: 00		A: 0	
B: 001		B: 01		B: 111	
C: 010	$\overline{m}$ = 3.00	C: 100	$\overline{m}$ = 2.47	C: 110	$\overline{m}$ = 2.37
D: 011	H = 2.31208	D: 101	H = 2.31208 (const.)	D: 100	H = 2.31208 (const.)
E: 100	r = 22.93 %	E: 110	r = 6.39 %	E: 1011	r = 2.44 %
F: 101		F: 1110		F: 10101	
G: 110		G: 1111		G: 10100	

Abb. 1 ("numerierende" Codierung) *Abb. 2 (Codierung nach Shannon-Fano)* *Abb. 3 (Codierung nach Huffman)*

In the following, the two most common methods for source coding will be explained explicitly.

2. the Shannon-Fano method

1) Order the queue symbols according to falling probability (Fig. 4.0).

2) Subdivision of the resulting symbol series into two contiguous subgroups in such a way that the sums of the symbol probabilities of both subgroups are approximately equal (*): Separation line between AB and CDEFG; p(AB)=0.57 p(CDEFG)=0.43.

3) All symbols which are in the 1st subgroup are assigned the (partial) code "0", those that are in the 2nd subgroup are assigned the symbol "1'.

4) Continuation of points 2) and 3) with all the subgroups thus created, until at the end of each "branch" of the ~partial tree" only two subgroups are left (Fig. 4.2). are left (Figs. 4.2 to 4.4).

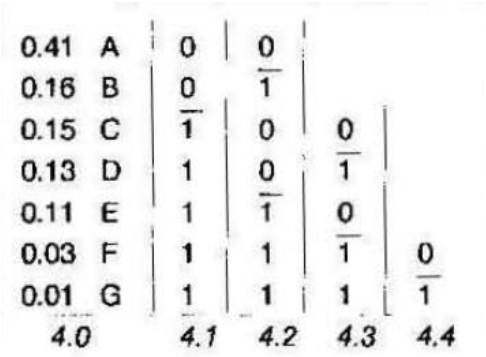


Abb. 4 (Shannon-Fano-Verfahren)

(*) "Approximately coincide" is admittedly a somewhat woolly term, but so present in the definition of the procedure. However, it says nothing other than that the difference of the symbol probability sums of both subgroups should assume a minimum value. Here this procedure is not unique: There are certainly probability distributions conceivable, in which one can be just as legitimately decided in favor of one the other separation of the groups could be chosen just as legitimately.

3.- The Huffman method.

1) Ordering the source symbols by decreasing probability and assigning the labels a1, a2, ... ,an (Fig. 5)

2) The two symbols in the row are under one name (a67 in Fig. 5.1), add the individual probabilities and sort the whole in a new diagram according to falling probabilities (Fig. 5.1).

3) Step 2) is repeated until only two different symbols remain in the (then) last table (Fig. 5.5).
This is followed by the actual coding, starting with the last table:

4) The two lowest symbols are assigned the (partial) codes "1" and "0" are assigned to the two lowest symbols (in square brackets in Fig.5.5O).

5) Repeat step 4) until all tables are processed. New (partial) codes are simply appended to the right to the right of the already existing codes (in bold).

a1	0.41	[0]	a1	0.41	[0]	a1	0.41	[0]
a2	0.16	[111]	a2	0.16	[111]	a2	0.16	[111]
a3	0.15	[110]	a3	0.15	[110]	a3	0.15	[110]
a4	0.13	[100]	a4	0.13	[100]	→ a567	0.15	[101]
a5	0.11	[1011]	a5	0.11	[1011]	a4	0.13	[100]
a6	0.03	[10101]	→ a67	0.04	[1010]			
a7	0.01	[10100]						
	5.0			*5.1*			*5.2*	
a1	0.41	[0]	a1	0.41	[0]	→ a2-7	0.59	[1]
→ a4567	0.28	[10]	a23	0.31	[11]	a1	0.41	[0]
a2	0.16	[111]	→ a4567	0.28	[10]			
a3	0.15	[110]						
	5.3			*5.4*			*5.5*	

Abb. 5 (Huffman-Verfahren)

4. program sequence on the HP-41

It is to be noted that *the symbol probabilities must happen in descending row order*. An additional sorting routine for this is already huge program - also because of speed considerations – so I did not think it would make sense (see sequence 1).

With the input of "0", the input routine is left and asks for the desired encoding - in best menu manner:
XEQ A starts the Shannon-Fano encoding,
XEQ B the Huffman coding.

If, on the other hand, you want to only calculate the source entropy from the entered probabilities, then press key "H" (XEQ "H) to start the corresponding routine.

EINGABE:	ANZEIGE:	KOMMENTAR:
XEQ"CODE"	"P1="	Programmstart
0.41 R/S	"P2="	Eingabe von p1
0.16 R/S	"P3="	Eingabe von p2
0.15 R/S	"P4="	Eingabe von p3
0.13 R/S	"P5="	Eingabe von p4
0.11 R/S	"P6="	Eingabe von p5
0.03 R/S	"P7="	Eingabe von p6
0.01 R/S	"P8="	Eingabe von p7
0 R/S	"SH HUF"	Abbruch der Eingaberoutine

Ablauf 1

If you set the flag 00 before the start, you can follow the individual group divisions in the display (see sequence 2). With another R/S the whole output can be repeated. After this demonstration of the Shannon-Fano routine now to the Huffman procedure, which, like the previous program, can be started at any time - it is sufficient if the probabilities have been entered once (see procedure 3).

SF 00	...	Teilung mitverfolgen -> SF 00
XEQ"A"	...	Start des Shannon-Fano-Verfahrens durch drücken der Taste "A".
	"T: 1-2 3-7"	1.Teilung: AB-CDEFG
	"T: 1-1 2-2"	2.Teilung: A-B (Untergruppe 1.Art)
	"T: 3-4 5-7"	3.Teilung: CD-EFG (Untergruppe 1.Art)
	"T: 3-3 4-4"	4.Teilung: C-D (Untergruppe 2.Art)
	"T: 5-5 6-7"	5.Teilung: E-FG (Untergruppe 2.Art)
	"T: 6-6 7-7"	6.Teilung: F-G (Untergruppe 3.Art)
	"C1=00"	Code von A: 00
R/S	"C2=01"	Code von B: 01
R/S	"C3=100"	Code von C: 100
R/S	"C4=101"	Code von D: 101
R/S	"C5=110"	Code von E: 110
R/S	"C6=1110"	Code von F: 1110
R/S	"C7=1111"	Code von G: 1111
R/S	"M/=2,4700"	$\overline{m}$ = 2,47 (mittlere Codewortlänge)
R/S	"H=2,311208"	H = 2.31208 (Quellenentropie)
R/S	"RR=6,39 %"	r = 6,39 % (relative Redundanz)

Ablauf 2

```
SF 00                                ... wir wollen wieder mitverfolgen.
XEQ"B"          ...                  Start des Huffman-Verfahrens durch
                                     drücken der Taste "B"
                "1.SCHRITT"          1.Teilungsschritt
                "1: a1: 0,41"        (siehe Abb.5.1)
                "2: a2: 0,16"        (      -"-      )
                "3: a3: 0,15"        (      -"-      )
                "4: a4: 0,13"        (      -"-      )
                "5: a5: 0,11"        (      -"-      )
                "6: a67: 0,04"       (      -"-      )

                "2.SCHRITT"          2.Teilungsschritt
                "1: a1: 0,41"        (siehe Abb.5.2)
                "2: a2: 0,16"        (      -"-      )
                "3: a3: 0,15"        (      -"-      )
                "4: a567: 0,15"      (      -"-      )
                "5: a4: 0,13"        (      -"-      )

                "3.SCHRITT"          3.Teilungsschritt
                ...                  (siehe Abb.5.3)

                ...                  ...

                "5.SCHRITT"          5.Teilungsschritt

                "1: a234567: 0,59"
                "2: a1: 0,41"        (siehe Abb.5.5)

R/S             "C1=0"               Code von A: 0
R/S             "C2=111"             Code von B: 111
R/S             "C3=110"             Code von C: 110
R/S             "C4=100"             Code von D: 100
R/S             "C5=1011"            Code von E: 1011
R/S             "C6=10101"           Code von F: 10101
R/S             "C7=10100"           Code von G: 10100

R/S             "M/=2,3700"          m̄ = 2,37     (mittlere Codewortlänge)
R/S             "H=2,31208"          H = 2,31208  (Quellenentropie)
R/S             "RR=2,44 %"          r = 2,44 %   (relative Redundanz)
```

Ablauf 3

Program listing.

```
01*LBL "CODING"
02 20
03 STO 05
04  E
05 STO 10
06 STO 00
07 STO 12
08 -
09 STO 13
10  E1
11 +
12 XROM "*SZ"
13  E3
14 STO 11
15 CF 29
16*LBL 00
17 "P"
18 FIX 0
19 ARCL 00
20 "`=?"
21 PROMPT
22 RCL 10
23 X<Y?
24 GTO 27
25 X<>Y
26 STO 10
27 RCL 13
28 RCL 00
29 +
30 X<>Y
31 STO IND Y
32 RCL 12
33 ST+ 00
34 X<>Y
35 X#0?
36 GTO 00
37*LBL 01
38 RCL 12
39 ST+ X
40 ST- 00
41 RCL 00
42 RCL 13
43 +
44 RCL 11
45 /
46 RCL 05
47 +
48 STO 01
49 SF 27
50 "SHA  HUF"
```

```
51 PROMPT
52*LBL 14
53 RCL 00
54 RCL X
55 RCL 11
56 /
57 +
58 RCL 01
59 +
60 STO 06
61 RCL 12
62*LBL 02
63 STO IND Y
64 ISG Y
65 GTO 02
66 RTN
67*LBL "HUFFM"
68*LBL C
69 RCL 00
70 4
71 *
72 RCL 05
73 +
74 RCL 12
75 -
76 XROM "*SZ"
77 3
78 STO 19
79 2
80 STO 18
81 XEQ 14
82 X<>Y
83 INT
84 RCL X
85 RCL 00
86 +
87 RCL 12
88 -
89 RCL 11
90 /
91 +
92 STO 03
93 STO 02
94 RCL 00
95 RCL X
96 RCL 11
97 /
98 +
99 +
100 STO 04
101 RCL 12
102*LBL 15
103 STO IND 02
104 ST+ X
105 ISG 02
106 GTO 15
107 RCL 01
108 STO 02
109 RCL 00
110 RCL 19
111 *
112 STO 07
113*LBL 16
114 RCL 02
115 RCL 07
116 +
117 RCL IND 02
118 STO IND Y
119 ISG 02
120 GTO 16
121 RCL 00
122 STO 13
123*LBL 20
124 RCL 13
125 RCL 04
126 INT
127 +
128 RCL 12
129 -
130 RCL X
131 DSE X
132 RCL IND X
133 RCL IND Z
134 +
135 STO 14
136 RDN
137 RCL 00
138 ST- Y
139 ST- Z
140 RDN
141 RCL IND X
142 STO 16
143 RCL IND Z
144 STO 17
145 +
146 STO 15
147 RCL 04
148 RCL 00
149 RCL 13
150 -
151 RCL 18
152 +
153 RCL 11
154 /
155 -
156 STO 02
157*LBL 17
158 RCL 14
159 RCL IND 02
160 X<Y?
161 GTO 18
162 ISG 02
163 GTO 17
164*LBL 18
165 RCL 14
166 RCL 15
167*LBL 19
168 RCL 02
169 RCL 00
170 -
171 X<>Y
172 X<> IND Y
173 X<> Z
174 X<> IND 02
175 RCL Z
176 ISG 02
177 GTO 19
178 RCL 02
179 RCL 00
180 -
181 X<>Y
182 STO IND Y
183 X<> Z
184 STO IND 02
185 RCL 12
186 ST- 13
187 FS? 00
188 XEQ 29
189 SF 05
190 RCL 16
191 XEQ 21
192 RCL 17
193 CF 05
194 XEQ 21
195 RCL 13
196 RCL 12
197 X#Y?
198 GTO 20
199 SF 06
200 GTO 10
201*LBL 21
202 RCL 06
203 STO 10
```

```
204 X<>Y
205 RCL 18
206*LBL 22
207 RCL Y
208 X<>Y
209 MOD
210 ST- Y
211 X<>Y
212 LASTX
213 /
214 LASTX
215 RCL Z
216 X=0?
217 GTO 23
218 X<>Y
219 ST* IND 10
220 FS? 05
221 ISG IND 10
222 ""
223 X<>Y
224*LBL 23
225 X<> Z
226 X<>Y
227 ISG 10
228 GTO 22
229 RTN
230*LBL 29
231 RCL 13
232 RCL 12
233 X=Y?
234 RTN
235 RCL 00
236 RCL 11
237 /
238 RCL 12
239 +
240 STO 14
241 2
242 STO 08
243 RCL 00
244 RCL 13
245 -
246 "STEP "
247 FIX 0
248 ARCL X
249 "`:"
250 AVIEW
251 RCL 03
252 STO 09
253 RCL 13
254 RCL 11
255 /
256 RCL 12
257 +
258 STO 02
259*LBL 30
260 CLA
261 ARCL 02
262 "`: a"
263 RCL 14
264 STO 07
265 RCL IND 09
266 RCL 08
267*LBL 31
268 RCL Y
269 X<>Y
270 MOD
271 ST- Y
272 X<>Y
273 LASTX
274 /
275 X<>Y
276 X#0?
277 ARCL 07
278 X<>Y
279 X=0?
280 GTO 32
281 LASTX
282 ISG 07
283 GTO 31
284*LBL 32
285 "`: "
286 FIX 2
287 RCL 09
288 RCL 00
289 +
290 RCL IND X
291 RND
292 X=0?
293 FIX 5
294 ARCL IND Y
295 AVIEW
296 ISG 09
297 ""
298 FIX 0
299 ISG 02
300 GTO 30
301 RTN
302*LBL "SHANN"
303*LBL A
304 RCL 05
305 RCL 00
306 2.5
307 *
308 +
309 INT
310 XROM "*SZ"
311 CF 06
312 RCL 05
313 RCL 12
314 -
315 STO 13
316 XEQ 14
317 X<>Y
318 STO 17
319 CLX
320 STO 16
321 RCL 00
322 RCL 11
323 /
324 RCL 12
325 +
326 XROM "PUSH"
327*LBL 03
328 XROM "POP"
329 X=0?
330 GTO 10
331 INT
332 LASTX
333 FRC
334 RCL 11
335 *
336 STO 04
337 X<>Y
338 STO 03
339 X<>Y
340 RCL 13
341 +
342 RCL 11
343 /
344 +
345 RCL 13
346 +
347 STO 02
348 STO 15
349 CLX
350 STO 07
351 STO 08
352*LBL 06
353 RCL IND 02
354 ST+ 08
355 ISG 02
356 GTO 06
```

```
357 RCL 15
358 STO 02
359 RCL 12
360 STO 14
361*LBL 07
362 RCL IND 02
363 STO 09
364 ST- 08
365 ST+ 07
366 RCL 08
367 RCL 07
368 -
369 ABS
370 RCL 14
371 X<Y?
372 GTO 08
373 X<>Y
374 STO 14
375 ISG 02
376 GTO 07
377*LBL 08
378 RCL 02
379 INT
380 RCL 05
381 -
382 RCL X
383 RCL 11
384 /
385 RCL 03
386 +
387 X<>Y
388 RCL 04
389 RCL 11
390 /
391 RCL 12
392 +
393 +
394 X<>Y
395 STO 03
396 X<>Y
397 STO 04
398 FC? 00
399 GTO 28
400 X<>Y
401 INT
402 LASTX
403 FRC
404 RCL 11
405 *
406 "T: "
407 FIX 0
408 ARCL Y
409 "`-"
410 ARCL X
411 RCL 04
412 INT
413 LASTX
414 FRC
415 RCL 11
416 *
417 "` "
418 ARCL Y
419 "`-"
420 ARCL X
421 AVIEW
422*LBL 28
423 RCL 04
424 RCL 06
425 INT
426 RCL 12
427 -
428 RCL X
429 RCL 11
430 /
431 +
432 +
433 STO 02
434 2
435 STO 09
436*LBL 04
437 RCL IND 02
438 RCL 09
439 *
440 RCL 12
441 +
442 STO IND 02
443 ISG 02
444 GTO 04
445 RCL 04
446 ISG 04
447 XROM "PUSH"
448 RCL 03
449 RCL 06
450 INT
451 RCL 12
452 -
453 RCL X
454 RCL 11
455 /
456 +
457 +
458 STO 02
459*LBL 05
460 RCL IND 02
461 ST+ IND 02
462 ISG 02
463 GTO 05
464 RCL 03
465 ISG 03
466 XROM "PUSH"
467 GTO 03
468*LBL "PUSH"
469 RCL 12
470 ST+ 16
471 CLX
472 RCL 16
473 RCL 17
474 +
475 X<>Y
476 STO IND Y
477 RTN
478*LBL "POP"
479 RCL 16
480 X=0?
481 RTN
482 RCL 16
483 RCL 17
484 +
485 RCL IND X
486 RCL 12
487 ST- 16
488 X<>Y
489 RTN
490*LBL 10
491 "0"
492 ASTO 18
493 "1"
494 ASTO 19
495 RCL 06
496 STO 02
497 2
498 STO 08
499 LN
500 STO 09
501*LBL 09
502 "C"
503 RCL 02
504 RCL 06
505 INT
506 -
507 RCL 12
508 +
509 FIX 0
```

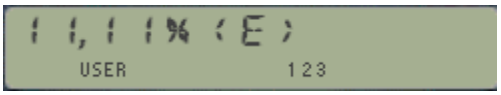
```
510 ARCL X
511 "`="
512 FS? 06
513 GTO 24
514 RCL IND 02
515 STO 03
516 XEQ 55
517 RCL 08
518 X<>Y
519 Y^X
520 STO 04
521 ST- 03
522*LBL 11
523 RCL 08
524 ST/ 04
525 RCL 03
526 RCL 04
527 X<=Y?
528 ST- 03
529 X<=Y?
530 ARCL 19
531 X>Y?
532 ARCL 18
533 RCL 12
534 RCL 04
535 X>Y?
536 GTO 11
537 GTO 25
538*LBL 24
539 RCL 08
540 RCL IND 02
541*LBL 26
542 RCL 12
543 X=Y?
544 GTO 25
545 RDN
546 X<>Y
547 RCL Y
548 X<>Y
549 MOD
550 X=0?
551 ARCL 18
552 X#0?
553 ARCL 19
554 ST- Y
555 X<>Y
556 LASTX
557 /
558 LASTX
559 X<>Y
560 GTO 26
561*LBL 25
562 PROMPT
563 ISG 02
564 GTO 09
565 RCL 06
566 STO 02
567 CLX
568 STO 18
569*LBL 12
570 RCL IND 02
571 XEQ 55
572 RCL 02
573 RCL 00
574 -
575 X<>Y
576 RCL IND Y
577 *
578 ST+ 18
579 ISG 02
580 GTO 12
581 "M/="
582 FIX 4
583 ARCL 18
584 PROMPT
585*LBL H
586 RCL 01
587 STO 02
588 CLX
589 STO 19
590*LBL 13
591 RCL IND 02
592 1/X
593 2LD
594 RCL IND 02
595 *
596 ST+ 19
597 ISG 02
598 GTO 13
599 "H="
600 FIX 5
601 ARCL 19
602 PROMPT
603 RCL 18
604 RCL 19
605 -
606 RCL 18
607 /
608 "RR="
609  E2
610 *
611 FIX 2
612 ARCL X
613 "` %"
614 PROMPT
615 GTO 10
616*LBL "*SZ"
617 SF 25
618 RCL IND X
619 FS? 25
620 X<>Y
621 RCL 12
622 +
623 FC?C 25
624 PSIZE
625 RTN
626*LBL 27
627 "ERROR: P"
628 FIX 0
629 RCL 00
630 RCL 12
631 +
632 ARCL X
633 "`P"
634 ARCL 00
635 PROMPT
636 RTN
637*LBL 55
638 RCL 08
639 RCL Y
640 2LD
641 INT
642 RCL 12
643 +
644 Y^X
645 X>Y?
646 DSE L
647 LASTX
648 END
```

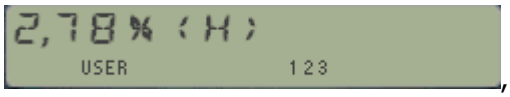
SEMIO ; Semiotic Analysis.

To complete the collection here is the simplest application of semiotic analysis: calculate the frequency of appearance of each character in a given text. The program will allow you to enter a very long text in segments of up to 24 characters each, storing them in data registers as you go. You can run the analysis at any intermediate stage or at the end of the process, as offered by the menu-driven choices:

- Press the [**A**] key to continue adding more text, and
- Press the [**C**] key to run the analysis of the partial (or final) text.

Each time you request the analysis it'll show you the total number of characters and the relative frequency of each character's appearance, until the list is complete (and the sum equals 100%).



, 

, etc…

Program listing:

```
01*LBL "SEMIO"
02 CLRG
03 SIZE?
04 30
05 X>Y?
06 PSIZE
07*LBL J
08 "ADD SHOW"
09 PROMPT
10 GTO J
11*LBL A
12 CLA
13 PMTA
14*LBL 00
15 ATOX
16 X=0?
17 GTO 02
18 62
19 -
20 ISG IND X
21 ""
22 GTO 00
23*LBL 02
24 STO 02
25 GTO J
26*LBL C
27 3.03
28 STO 00
29*LBL 10
30 RCL IND 00
31 ST+ 02
32 ISG 00
33 GTO 10
34 "CHR: "
35 ARCL 02
36 PROMPT
37 CLX
38 STO 01
39 3.03
40 STO 00
41*LBL 05
42 RCL IND 00
43 X#0?
44 XEQ 09
45 ISG 00
46 GTO 05
47 GTO J
48*LBL 09
49 RCL 02
50 RCL IND 00
51 %CH
52  E2
53 +
54 CLA
55 ARCL X
56 "`%("
57 RCL 00
58 62
59 +
60 XTOA
61 "`)"
62 PROMPT
63 END
```