Notes on the back story of this letter:

I sent this *9-page* letter to ***John McGechie*** on Sept, 27 (1980) to thank him and the members of the *PPC Melbourne Chapter* for their extremely warm welcome and enthusiastic reception of my materials, which I began to submit to them as well after growing fed up with *Richard Nelson*'s continuous disregard and complete silence. In stark contrast, the Australian people were warm and appreciative and, most importantly, they *did* communicate with me very extensively and on a regular basis, while *Mr. Nelson* never *ever* sent a single word on *anything*, be they my submittals, questions, comments, proposals, news, requests, whatever. *Melbourne's PPC* was *heaven* in comparison.

The letter continues with comments re the *PPC Jul/Aug* issue's materials (which included some *Mr. Nelson* grabbed directly from *Melbourne's Technical Notes*) and on his alleged disapproval of *TN* publishing materials previously submitted to him (even if he chose *not* to publish them, talk about *dog in the manger* !), as well as comments on the materials selected (or not) for the *PPC ROM*. Additionally, I also comment on the article I sent to *PPC* which was censored because of **`PRIVATE`**, and possible "culprit party" for that censorship.

After some comments on two *Solutions Books* written by me and my friend *Fernando del Rey* which *HP Spain* intended to publish locally, I include and detail my own attempt to improve on John's ***b2*** routine (using my approach discussed in a previous letter) which indeed did result in working code but John's original approach was still better.

Finally, I include the following materials: *(1)* ***RP***, an **HP-41C** program to automatically find *all roots of a polynomial equation* of arbitrary degree (limited only by available memory) with real and/or complex coefficients, in a completely global fashion (i.e., no initial approximations required) and callable as a subroutine from other programs (I had previously sent *RP* to *Mr. Nelson* for publication in *PPC CJ* but just in case he would silently ignore it as usual, I didn't want to have its publication indefinitely delayed); *(2)* an **HP-41C** fast implementation of an *N-level RPN stack* including all the usual functions, with *N* limited only by available memory, and *(3)* an autolearning **HP-41C** game, *"Even Wins"*, which plays better and better the more you play against it.

Last, I mention my programs to be submitted next to the *Melbourne Chapter*, including least-squares *N-variable linear regression* or *Nth-degree polynomial regression*, *NxN-matrix 2-level RPN stack* with the usual matrix operations, plus *Checkers* and *Chess 5x5* coming soon.

Valentin Albillo, 14-02-2022

John McGechie
Philosophy Department
Monash University
Clayton, Victoria
AUSTRALIA , 3168

Valentin Albillo
Padre Rubio, 61- 2º C
Madrid 29
SPAIN

Sept, 27

Dear John:

First, notice I follow your suggestions: a small envelope, and the letter is entirely written as if it were to be submitted to PPC: one side printing, 14 cm wide (the edges have been chopped off, to reduce weight). Your letter dated 17th arrived just today: I was beginning to believe you had forgotten me, you see. Glad to hear all my letters arrived: by the way, the letter containing PPC Technical Notes did also - arrived in an envelope opened along one edge, it should be a - new fashion ! Very happy to hear my material is of any use to- some of you. I whishto thank you, each and everyone of you,for accepting me as a member of your Chapter, I'll try to do my best at this distance (over 20,000 Km!) to deserve such an honor. I specially whish to thank Ernie Gibbs and Richard Kuhoutek (right spelling?) for typing some of my programs: from now on, I will either type the programs in this format 14 cm wide, or send the best quality photocopies I could afford, so as to avoid any subsequent retyping.

Some comments about Jul/Aug issue of PPC: I notice the pages full of reports of the Melbourne Chapter, it is wonderful to see those all good news being shared with the whole membership. In fact, Richard took some material from PPC TN,the AN routine for example. I don't see why you suspect Richard does not approve your publishing material previously sent to him, he seems to publish even things not submitted to him ! On the other hand, Richard should be happy of having so much material contributed: you should know by now that every number of PPC journal reflects the knowledges and discoveries of several months ago, so quite a lot of published material is obsolete by the time of its publication. PPC TN seems to have avoided this problem, publishing the hottest news, so I don't see why Richard would have nothing to object: for instance, I sent him the Othello program several months in advance, about two months before I sent it to you, however, it seems that it will be published in PPC TN and not in PPC CJ; if Richard has the material, but does not make use of it, why would he object others publishing it ?

Another comment: The column ROM PROGRESS in V7 N6 P32 (jul/Aug iss) features a proposal for the contents of - the Synthetic Programming Group for the Custom ROM. It includes 3 of my routines: the Byte Counter, Display Test, and Flag reset. I feel two others should be included as well, namely the Clear Assignments (it is shorter than the versions published , although I have yet to test overall performance of those other versions),and the sigma finder, (also curtain finder, etc),but this is irrelevant. What is certainly inadmissible is that your routines AN & little b2 were not included in the list !!!

Now, I feel that you must do something ! Those routines are most useful. In fact, after having your AN, I have never used DECODE: your routine is so much shorter and faster, that it should be preferred. I don't object to DECODE being included in the ROM, it has to, but your routine AN is a must. The same is valid for your b2: it is far more convenient than the B2 routine from Wickes included in that list. Your b2 is faster (B2 uses DECODE as subroutine) and more convenient to use, because it does not require the decimal address of the register in which the NNN is to be stored, but rather a simple program pointer address, easily obtained via RCL b. I've tried to program a little b2 in the other way I told you in a previous letter, but to no purpose, yours is better. See my attempt later.

Resuming, please, do something ! Your routines are so useful that the ROM will be enhanced if they are included. I'll write to PPC asking for them. Also, another miss is the B3 routine published in (I think) the May issue of PPC, or a similar one. It seems obvious that a routine to set or clear any flag is needed, and I can't see why the B3 is not in the list.

Another comment: my "Universal Byte jumper" was finally published in the Jul/Aug issue, and the point (5) was deleted, as expected (the method of breaking PRIVATE using just STO b, RCL b). However, Richard forgot (?) to also suppress the header, in which I mentioned the possibility of breaking PRIVATE that way, and several PPC members got my address (I can't figure out how!) and wrote me asking for the method. Most noticeably, Mr. A.C. Collinson, who works for HP in the United Kingdom. I was told by some HP people that Richard and HP made an agreement in the sense that PPC would not publish any information concerning PRIVATE. I really don't believe it, but , do you think that it is possible ? Anyway, I think the easiest method to break PRIVATE is using the Wickes' Byte jumper, though I've not tried it yet. Are you going to publish methods to break PRIVATE in PPC TN ?

Concerning the books "HP-34C Advanced Math" & "HP-41c Advanced Math", I'll send you a copy as soon as I can get them. I have a copy of each, but it is easy to get HP to give me another two. They are the same format as the rest of the Solution Books for the 41c. Remember they are written in Spanish. The book for the 41c will include bar codes (my copy does not include bar codes, but it is a pre-production copy: future copies <u>will</u> include bar codes, possibly printed in Spain ! I'm working in a program for the HP-85 to print barcodes via a peripheral plotter, and it works). Can you suggest a way to send them safely, so they will arrive to your hands in a good condition ? Large envelope ? flat box ? Any ideas ?

Here is my attempt to improve your b2:

```
LBL"b2"      X()Y      -             LBL 01     INT
CLA          XEQ 01    "天図ν¯図"    STO M      HMS
X() M        16        RCL c         R down     16
"├123456"    MOD       R up          "├----天"  *
X() N        LASTX     ASTO c        RCL M      RCL M
"├7"         x²        STO IND Z     FRC        +
Ø            *         X()Y          10         END
X() N        +         STO c         *
XEQ 01       192       RTN           X() M
```

it works the same as yours, NNN in Y and Rb in X, but it is 90 bytes, 44 lines, while yours is 36 lines, 89 bytes. Besides, yours is faster. So I was wrong, and your method is better. First text is F501690C00BF. The 2nd is F67F0000000001.

I have been working in several programs. I've put aside synthetics (momentarily, don't fear!), and set the task of writing some programs. Of course, those programs could use synthetic functions, but, as they are written to be published later by HP, synthetics are out of the question. Here included are several programs:

1) <u>Roots of polynomial equations</u> : this program solves an equation of any degree, with real and/or complex coefficients. The program finds all its n roots, real or complex. <u>No</u> initial approximations are required: only input are the coefficients

and the degree. Output are all n roots. The program can solve equations of degrees 1 thru 132. It fits and runs in a basic machine (up to 5th degree without modules, up to 36th degree using just one module).The roots are also stored, and are output once all of them have been computed. As they are stored, it may be - used as a subroutine, simple delete the input-output part. An - example of a program requiring root finding of polynomials is a filter design program. This one is optmized to run as fast as - possible and not take more than 300 bytes (298, in fact). Hope you (and other members as well) will like it. It is typed 14 cm wide, so it must be easy to use without retyping. Notice the - LBL 03 subroutine: it performs the multiplication of two complex numbers using just the stack, and no P-R or R-P conversion, to save time; it should be useful in any program requiring multi - plication of complex numbers.

2) n-th level RPN stack : this program simulates a RPN stack of n registers, where n is chosen by the user. It includes CL, +,-,x,/,ENTER,LASTX,PI,RCL,Rdown,X()Y,etc. (of course, STO is the normal STO). The stack may be of any number of levels, not just the 4-level stack built-in. It should be good for all those people who want 5-level stack, etc. This one is - the answer to their problems, isn't it ?

3) Even wins , a game: it is a much optimized version of a game I saw written in BASIC. It seemed good, so I took the work of translating it to RPN. See the rules in the enclosed listing. The game is most remarkable, because the computer (41c) starts knowing just the rules of the game, but it learns as it plays, and soon plays better than the user (similar to hexapawn programs published here and there, only this game is much more complex, and its strategy is not at all trivial). After 20 games in a row, the program is very hard to defeat. The better you play, the faster it learns to play well. The learning strategy is simple, but quite good.

This would suffice by now: it it already too much for such a - small envelope ! It almost does not fit !

By the way, here are some programs I am working on just now:

N-variables least-squares linear regressión: given any number of data points, finds the coefficients of the least-squares linear regressión of n variables. Can also find the n-th degree least-squares polynomial to fit the data. This is, not just linear regressión, or cuadratic regression, but n-th degree regression! Any member interested in Statistics will love this one. The program is already written, I'll send it in my next letter.

NxN matrix operations: simulates a 2-level RPN stack of NxN matrices, so allowing chaining of operations. Includes input,output,+,-,x,inverse, etc. Allows easy handling of NxN matrices. Such things as $3M^3-2M^2+8I$ where M is a given 10x10 (say) matrix or $P^{-1}.Q.P$, where P,Q are 8x8 matrices, are trivial. Almost finished, only a little debugging is still needed.

I'm also writing a program that plays checkers against the user, and still another that plays chess in a 5x5 board (Each side has the king, a queen, a rock, a bishop,a knight and 5 pawns) and includes all standard chess rules, except castling and capture "en passant". I already have the complete flowchart, and am in the phase of actually writing 41c code for it.

That's all. Please, sent me information about the "42-c" and the TI machine, and, if possible, a brochure of the Sharp handheld computer. Waiting for your news:

Yours

ROOTS OF POLYNOMIAL EQUATIONS - R/C COEFFICIENTS

This program finds all n roots, real and/or complex, of any given equation of degree n:

$$P(z) = c_n z^n + c_{n-1} z^{n-1} + \dots + c_2 z^2 + c_1 z + c_o = 0$$

where the coefficients, c_i are of the general form: $c_i = a_i + b_i i$ this is, they are complex coefficients. Of course, the particular case of real coefficients is included, simply all b_i are 0.

The program finds automatically all n roots of the equation. The roots are of the general form: $z_i = u_i + v_i i$ if the root is real, $v = 0$.

No initial approximations are needed, simply enter the coefficients and go have a cup of coffee. All roots will be computed to 10-digit accuracy, and stored as well. The roots are displayed after you press R/S, so you'll have all needed time to write them down.

CHARACTERISTICS

The program is 177 lines, 298 bytes long. It requires a minimum size 2n+11 to solve an equation of degree n. If you have no modules, you can solve up to a 4th degree equation (if you use the .END., and supress the alpha label, up to 5th degree is posible). Having modules, the following applies:

1 module - up to 36th degree
n modules - up to (32n+4) deg.

so, the range is 1(=n(=132 . Roots are stored, so this program may be used as a subroutine of another main program requiring the zeros of a polynomial (filters, perhaps) by simply supressing the input-output routines. See listing for details.

The execution time is quite fast, but for large n, it should be long. Each time flag 0 is set (watch indicator), a root has been found, and the search for another root beguins. The program first calculates the n-th root, then the (n-1)th one, up to the 1st one. If you want to review which root is being computed at a given moment, simply R/S, VIEW 00, will display the number of the root being calculated. Then R/S to resume the computation.

The program uses Newton's method to find each root, starting from a program's selected initial approximation:

$z_{n+1} = z_n - P(z_n)/P'(z_n)$, where the subscripts denote the next approximation,

$$P(z) = c_n z^n + c_{n-1} z^{n-1} + \dots + c_1 z + c_o$$

$$P'(z) = n c_n z^{n-1} + (n-1) c_{n-1} z^{n-2} + \dots + 2 c_2 z + c_1$$

Once a root has been found, deflation is used (by means of Horner's scheme) to remove the root from the equation, so it is reduced by one degree, and the search for another root begins. As the degree decreases by one (or two if coefficients are real and the root is complex) every time a root has been found, the following root takes usually less time to compute.

Every iteration includes about n+2 complex multiplications and 1 complex division (not to mention +,-). LBL 03 performs the multiplication of two complex numbers c_1, c_2, leaving the result in X,Y, and uses only the stack. It does not use R-P or P-R, so as to be as fast as possible.

HOW TO USE : the equation is $c_n z^n + c_{n-1} z^{n-1} + \dots + c_1 z + c_o = 0$
where $c_k = a_k + b_k i$

-set SIZE 2n+11 minimum, where n is the degree, of course.
-XEQ "RP" → N? , key in the degree of the equation
n R/S → An=? , key in An (real part of c_n)
a_n R/S → Bn=? , key in Bn (imag.part of c_n)

b_n R/S → An-1=? , keep on introducing all coefficients...
... → BØ=? , enter the last coefficient
b_0 R/S → the computation begins , every time a root is found, you'll see the 0 indicator turn on. After a while, all roots are computed and stored, the output takes place:

(beep) → ROOT 1 → U=real part of z_1
R/S → V=imag part of z_1
R/S → ROOT 2 → U=real part of z_2
R/S → V=imag part of z_2
... ...
→ ROOT n → U=real part of z_n
R/S → V=imag part of z_n
R/S → 0.00n-1

-for another equation, go back to the beginning.
-remember, $z_k = u_k + v_k i$. If the root is real, v_k is either 0 or very close to 0, say 2E-9 or so.

if your equation has only real coefficients, enter all b_i as 0

<u>WARNINGS</u> : -convergence is not guaranteed. It may be possible for the program to never find a root. However, I have been unable to find such a case: all tested cases up to date were solved successfully. Convergence is quadratic: once a good approximation is found, the number of exact digits doubles on every iteration.

-multiple roots will take much longer to compute, and the accuracy will get worse. For instance:

$x^2 + 4x + 4 = 0$ gives (2 min.26 sec), $z_1 = -2.0000005-0.0000004i$
(double root, $z_1=z_2=-2$) $z_2 = -1.9999995+0.0000004i$

$x^3+3x^2+3x+1=0$ (7 min.48 sec) , $z_1= -1.0004717-3.9996900E-8\ i$
$z_2= -0.9997642+0.0004079\ i$
(triple root, $z_1=z_2=z_3=-1$) $z_3= -0.9997641-0.0004079\ i$

<u>EXAMPLES</u> : 1) Find all roots of the following equation:

$(2+8i)z^6+(3+0i)z^5+(-1+2i)z^4+(0+2i)z^3+(-3-3i)z^2+(1+2i)z+(-2+3i)=0$

→the degree is 6, so SIZE 23
→ XEQ "RP" → N? , 6 R/S → A6=? , 2 R/S → B6=? , 8 R/S → A5=?
3 R/S → B5=? , 0 R/S → A4=? , -1 R/S → B4=? , 2 R/S → A3=?
0 R/S → B3=? , 2 R/S → A2=? , -3 R/S → B2=? , -3 R/S → A1=?
1 R/S → B1=? , 2 R/S → AØ=? , -2 R/S → BØ=? , 3 R/S →
computation takes place. After 8 min. you get:

→ ROOT 1 → U=-0.9724260 , R/S → V= 0.3032192 , R/S →
→ ROOT 2 → U=-0.0715576 , R/S → V= 1.1235559 , R/S →
→ ROOT 3 → U= 0.0323977 , R/S → V=-0.8883400 , R/S →
→ ROOT 4 → U= 0.5688927 , R/S → V= 0.5464170 , R/S →
→ ROOT 5 → U= 0.8266036 , R/S → V=-0.3541840 , R/S →
→ ROOT 6 → U=-0.4721457 , R/S → V=-0.3777269 , R/S → 0.0050000

2) Solve $5x^6 -4x^5 -3x^4 +8x^3 +8x^2 -2x +7 = 0$

→degree 6, SIZE 23, as before
→ XEQ "RP" → N? , 6 R/S → A6=? , 5 R/S → B6=? , 0 R/S → A5=?
-4 R/S → B5=? , 0 R/S → A4=? , -3 R/S → B4=? , 0 R/S → A3=?
8 R/S → B3=? , 0 R/S → A2=? , 8 R/S → B2=? , 0 R/S → A1=?
-2 R/S → B1=? , 0 R/S → AØ=? , 7 R/S → BØ=? , 0 R/S →
after just 5 minutes, you get:

→ ROOT 1 → U= 1.1936146 , R/S → V=-0.8739372 , R/S →
→ ROOT 2 → U= 1.1936146 , R/S → V= 0.8739372 , R/S →
→ ROOT 3 → U= 0.1940332 , R/S → V=-0.6858876 , R/S →
→ ROOT 4 → U= 0.1940332 , R/S → V= 0.6858876 , R/S →
→ ROOT 5 → U=-0.9876477 , R/S → V=-0.5325453 , R/S →
→ ROOT 6 → U=-0.9876477 , R/S → V= 0.5325453 , R/S → 0.0050000

Happy programming, folks !

VALENTIN ALBILLO (4747)

ROOTS OF POLYNOMIAL EQUATIONS - R/C COEFFICIENTS

```
01 LBL"RP"
02 FIX 0
03 CF 29
04 "N?"
05 PROMPT
06 STO 00
07 STO 03
08 9.008
09 +
10 STO 01
11 STO 05
12 RCL 00
13 ST+ X
14 10
15 +
16 STO 02
17 STO 06
18 LBL 05
19 "A"
20 ARCL 03
21 "⊢=?"
22 PROMPT
23 STO IND 05
24 "B"
25 ARCL 03
26 "⊢=?"
27 PROMPT
28 STO IND 06
29 DSE 03
30 X()Y
31 DSE 06
32 DSE 05
33 GTO 05
34 RCL 03
35 LBL 06
36 CF 00
37 CHS
38 STO 04
39 FIX 2
40 RND
41 FIX 6
42 X≠0?
43 GTO 01
44 SIGN
45 STO 04
46 LBL 01
47 RCL 00
48 STO 08
49 SF 01
50 XEQ 11
51 R-P
52 1/X
53 STO 07
54 X()Y
55 CHS
56 STO 08
57 CF 01
58 XEQ 11
59 RCL 08
60 RCL 07
61 P-R
62 XEQ 03
63 ST- 03
64 X()Y
65 ST- 04
66 RND
67 X≠0?
68 GTO 01
69 X()Y
70 RND
71 X≠0?
72 GTO 01
73 SF 00
74 XEQ 11
75 1
76 ST+ 05
77 ST+ 06
78 1 E-3
79 ST+ 01
80 RCL 03
81 STO IND 05
82 RCL 04
83 STO IND 06
84 DSE 00
85 GTO 06
86 BEEP
87 RCL 01
88 INT
89 10
90 -
91 1 E3
92 /
93 ST- 05
94 LBL 10
95 ISG 00
96 LBL 14
97 "ROOT "
98 FIX 0
99 ARCL 00
100 AVIEW
101 PSE
102 "U="
103 FIX 7
104 ARCL IND 05
105 PROMPT
106 "V="
107 ARCL IND 06
108 PROMPT
109 DSE 06
110 DSE 05
111 GTO 10
112 RTN
113 LBL 11
114 RCL 01
115 STO 05
116 RCL 02
117 STO 06
118 FC? 01
119 GTO 13
120 1 E-3
121 ST+ 05
122 LBL 13
123 RCL IND 06
124 RCL IND 05
125 FC? 01
126 GTO 02
127 RCL 08
128 ST* Z
129 *
130 DSE 08
131 GTO 02
132 RTN
133 LBL 00
134 RCL 04
135 RCL 03
136 XEQ 03
137 RCL IND 05
138 FS? 01
139 RCL 08
140 FS? 01
141 *
142 +
143 FS? 00
144 STO IND 05
145 X()Y
146 RCL IND 06
147 FS? 01
148 RCL 08
149 FS? 01
150 *
151 +
152 FS? 00
153 STO IND 06
154 X()Y
155 FS? 01
156 DSE 08
157 LBL 02
158 DSE 06
159 DSE 05
160 GTO 00
161 RTN
162 LBL 03
163 STO L
164 R↑
165 ST* Y
166 X() Z
167 ST* Z
168 R↑
169 ST* Y
170 ST* L
171 X() L
172 R↑
173 -
174 RDN
175 +
176 R↑
177 END
```

177 lines
298 bytes
SIZE 2n+11

registers:

```
00=n                09=a0    (un)       real parts
01=add. an          10=a1    (un-1)     of coeff.
02=add. bn  n       ... ... ...         (& roots)
03=r.p. of z        n+9=an   (an)
04=i.p. of z        n+10=b0  (vn)
05=aux.(an)         n+11=b1  (vn-1)     imag. parts
06=aux.(bn)         ... ... ...         of coeff.
07=auxiliar         2n+10=bn (bn)       (& roots)
08=auxiliar         ==============
```

remarks : typical running times: n=6 , 5 min (R) ; 8 min (C)
n=10, 14 min (R) ; 22 min (C)

VALENTIN ALBILLO (4747)

RPN STACK OF N LEVELS (by Valentin Albillo) (4747)

```
01 LBL"STKN"
02 "N=?"
03 PROMPT
04 11
05 +
06 1 E3
07 /
08 13
09 +
10 STO 11
11 13.012
12 STO 12
13 XEQ"CLN"
14 "READY"
15 PROMPT
16 LBL"XY"
17 FS?C 04
18 CF 22
19 FS?C 22
20 GTO 10
21 X() IND 11
22 RTN
23 LBL 10
24 XEQ 06
25 X()Y
26 RTN
27 LBL"RD"
28 XEQ"XY"
29 DSE 12
30 DSE 11
31 GTO 01
32 RTN
33 LBL 01
34 LASTX
35 X() 11
36 FRC
37 STO 12
38 1 E3
39 *
40 X() 12
41 RCL 12
42 +
43 X() 11
44 STO L
45 RDN
46 .012
47 ST+ 12
48 RDN
49 RTN
50 LBL 07
51 FS?C 04
52 CF 22
53 FC?C 22
54 GTO 06
55 X()Y
56 XEQ 06
57 X()Y
58 LBL 06
59 ISG 11
60 ISG 12
61 GTO 02
62 STO IND 11
63 RTN
64 LBL 02
65 13.012
66 STO 12
67 RDN
68 LASTX
69 X() 11
70 FRC
71 13
72 +
73 X() 11
74 STO L
75 RDN
76 STO IND 11
77 RTN
78 LBL"ET"
79 XEQ 07
80 SF 04
81 RTN
82 LBL 03
83 FS?C 04
84 CF 22
85 FS?C 22
86 RTN
87 ISG 11
88 GTO 10
89 RCL 11
90 FRC
91 13
92 +
93 STO 11
94 RDN
95 LBL 10
96 RCL IND 11
97 X() IND 12
98 RCL 11
99 FRC
100 RCL 12
101 INT
102 +
103 STO 11
104 RDN
105 X()Y
106 DSE 12
107 DSE 11
108 GTO 01
109 RTN
110 LBL"+N"
111 XEQ 03
112 +
113 RTN
114 LBL"-N"
115 XEQ 03
116 -
117 RTN
118 LBL"*N"
119 XEQ 03
120 *
121 RTN
122 LBL"/N"
123 XEQ 03
124 /
125 RTN
126 LBL"↑N"
127 XEQ 03
128 Y↑X
129 RTN
130 LBL"LX"
131 XEQ 07
132 LASTX
133 RTN
134 LBL"PI"
135 XEQ 07
136 PI
137 RTN
138 LBL"CLN"
139 XEQ 01
140 CLST
141 CF 04
142 CF 22
143 LBL 05
144 STO IND 11
145 DSE 12
146 DSE 11
147 ISG 12
148 GTO 05
149 RTN
150 LBL"RCLN"
151 XEQ 07
152 "RCL __ "
153 AVIEW
154 LBL 04
155 PSE
156 FC?C 22
157 GTO 04
158 RCL IND X
159 END
```

++

Characteristics .- This program simulates a n-level RPN stack, this is, a stack with n registers (not just the 4 registers of the standard, built-in, 4-level stack). The value, n, is chosen by the user, and is limited only by available memory. Several functions are provided: ENTER,X()Y,RDN,CLST,+,-,*,/,y^x,LASTX, PI,and RCL. The rest of the functions are the built-in functions, for instance, STO is the built-in STO, SQRT ,SIN,etc.

The program is 159 lines, 343 bytes. It requires SIZE n+12 for a n-level stack. All operations are very fast, even for large n, so the program may be used as easily as if it were the standard 4-level stack. All functions are supposed to be asigned to keys for its execution in USER mode: ET (ENTER) is assigned to 41 (ENTER), RD (RDN) to 22 (RDN) +N (+) to 61 (+), -N(-) to 51 (-), *N(*) to 71 (*), /N (/) to 81 (/), PI to -82 (PI), CLN (CLST) to -21 (CLΣ), RCLN (RCL) to 34 (RCL), XY (X()Y) to 21 (X()Y),↑N to -12(y^x)

The stack behaves exactly like the original one: it lifts and performs the same, register duplication, etc, but for a minor detail:RCL after ENTER does not overwrite the number in X, but the stack is lifted. This has been done intentionally, but can be changed to the overwrite mode easily.Except for this sequence, all other functions performs as you -

would expect, the upper register replicates each time the stack drops because of a two-number operation, etc.

RCLN, when executed, prompts for an argument with the standard RCL _ _, and the program stays in a PSE loop, waiting for you to enter the argument for the desired register. This can be 0 thru 10 (both included) and n+12 upwards, where n is the number of levels of your stack. So, when using STO, remember that you have registers 00 thru 10 and n+12 upwards for your use. R11,R12 are used as scratch, and R13 thru R(n+11) are used to store part of the stack.

Instructions.- make all the neccessary assignments, set USER mode

-use the stack as normal: first, XEQ "STKN" → N=?
-enter the desired number of levels: n R/S → READY
-from now on, think of the 41c as a n-level stack machine, and - execute desired functions accordingly. Take into account that STO should be used only with addresses 00 thru 10 and n+12 up, and the same is true for RCL. The argument for RCL is entered during a pause. RCL after ENTER does not overwrite X, but lifts the stack first.

EXAMPLES : We want a 5-level RPN stack: set USER, FIX 2

XEQ "STKN" → N=? , 5 R/S → READY
1 ENTER 2 ENTER 3 ENTER 4 ENTER 5 RDN → 4.00, RDN → 3.00, RDN → 2.00 , RDN → 1.00, RDN → 5.00 (the 5 levels have been - shown) , * → 20.00, + → 23.00 , X()Y → 2.00, X()Y → 23.00, RDN → 2.00, RDN → 1.00, RDN → 1.00, RDN → 1.00, RDN → 23.00 ,

(the upper register has replicated as the stack dropped),

LASTX → 20.00, / → 1.15, STO 03 → 1.15, * → 2.30, PI → 3.14, + → 5.44, RCL → RCL _ _ , 3 → 1.15, y^x → 7.02, CLΣ → 0.00

so, you see, it is as easy to use as if it were the normal stack. Now, let´s compute an example taken from TI adds:

Compute $1 + 2 * 2.5^{(3/7)} = ?$

-if we want to key in the problem left-to-right, we need a 5-level stack (minimum):

XEQ"STKN" → N=? , 5 R/S → READY

1 ENTER 2 ENTER 2.5 ENTER 3 ENTER 7 , / → 0.43 , y^x → 1.48 , * → 2.96 , + → 3.96 , FIX 9 → 3.961936296

so, the problem was keyed in left-to-right. This is a very good advantage of a n-level stack, you can hold up to n-1 pending operations. Using the standard 4-level stack, up to 3 operations may be left pending, and problems requiring more pending operations cannot be keyed left-to-right, and have to be rearranged.

But, using a, say, 15-level stack, you can hold as many as 14 pending operations, and thus, you can confidently key in any - problem left to right, without rearranging anything. That´s the usefulness of the program. You can also use it when leaving someone your 41c, and that person is not very used to RPN: show him how to use ENTER,RDN,and X()Y, and let the 15 (say) level stack do the rest !

VALENTIN ALBILLO (4747)

EVEN WINS (by Valentin Albillo , #4747)

```
LBL"EVEN"
FIX 0
STO 00
CLX
STO 01
STO 03
9.02
CF 29
4
LBL 00
STO IND Y
ISG Y
GTO 00
LBL 01
CLX
STO 05
STO 06
RCL 00
R-D
FRC
STO 00
13
x
9
+
2
/
INT
ST+ X
1
+
STO 07
LBL 02
"THERE ARE "
ARCL 07
AVIEW
PSE
RCL 03
STO 04
RCL 01
STO 02
RCL 05
2
MOD
CF 02
X=0?
SF 02
STO 03
RCL 07
6
MOD
STO 01
RCL 03
XEQ 04
RCL 07
X(=Y?
GTO 06
X()Y
STO 08
X(=0?
GTO 09
ST- 07
ST+ 06
"I TAKE "
ARCL 08
AVIEW
TONE 9
PSE
"LEAVE "
ARCL 07
AVIEW
PSE
"YOU?"
PROMPT
STO 08
RCL 07
X=Y?
GTO 08
RCL 08
ST- 07
ST+ 05
GTO 02
LBL 06
"I TAKE "
ARCL 07
AVIEW
TONE 9
PSE
RCL 07
STO IND T
ST+ 06
LBL 08
RCL 06
2
MOD
X=0?
GTO 10
LBL 09
"YOU WIN"
AVIEW
BEEP
PSE
RCL 01
RCL 03
XEQ 04
1
X=Y?
GTO 13
GTO 07
LBL 10
"I WIN"
AVIEW
BEEP
PSE
GTO 01
LBL 13
RCL 02
RCL 04
XEQ 04
1
X=Y?
GTO 01
LBL 07
-
STO IND Y
GTO 01
LBL 04
6
x
+
9
+
RCL IND X
END
```

134 lines

SIZE 021

Description .- At the beginning of the game, a random number of chips are placed on the board. On each turn, a player must take 1,2,3 or 4 chips. The winner is the player who finishes with a total number of chips that is even. The game runs continually, as soon as one finishes, another starts, but you may quit at any moment by inputting 0 as your move.

What is really remarkable is that the computer starts out knowing only the rules of the game, but a learning mechanism allows it to learn from its mistakes, playing gradually better and better, until it is extremely difficult to beat. In fact, after 20 games in a row, it is almost unbeatable. Thus, this is a learning program: the better you play against it, the faster it learns to play well.

It could be interesting for those members desiring to implement playing strategies which learn more and more as they play.

Warnings : your move is not tested for legality: you must take 1,2,3 or 4 chips. HP moves first. If, at any moment, flag 02 is set (see annunciator), your total is even. If flag 02 is clear, your current total is odd.

Instructions .- to begin a session: key in a seed (betw. 0 & 1) seed , XEQ"EVEN" → THERE ARE nn → I TAKE m → LEAVE nn → YOU? enter your move (take 1,2,3 or 4): n R/S →
→ THERE ARE nn → I TAKE m → LEAVE nn → YOU?
continue the game. When the last chip is taken off the board, either YOU WIN or I WIN appears (the winner is the one with a total number of chips that is even) and another game starts.