Notes on the back story of this letter:

I sent this *10-page* letter to ***Richard Nelson*** on Sept, 27 (1980), including within the following materials for their publication in the *PPC Calculator Journal*, namely:

(1) ***BASIC Software for the HP-41C****,* an article describing some guidelines to convert an arbitrary *BASIC* program to the **HP-41C**'s *RPN* version in a rather automatic way, so as to create an *RPN* program that worked like the original *BASIC* program without knowing anything about the algorithms used, the implementation details or even the program's purpose. An enclosed example demonstrated in full the complete translation process applied to an autolearning *BASIC* game, *"Even Wins"*, which plays better and better the more you play against it.

(2) ***RP (Roots of Polynomials)***, an **HP-41C** program to automatically find *all real and/or complex roots of a polynomial equation* of arbitrary degree (up to 132, 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.

Finally, I commented on the proposed publication of the *PPC Barcode Book* and the wand itself, some notes on the memory required to call **`XROM`** functions with long names, and the idea of asking members to rate programs published in the *PPC CJ*.

.

Valentin Albillo, 14-02-2022

Richard Nelson
Editor, PPC CJ
2541 W. Camden Place
U.S.A.

Sept, 25 - 80

Valentin Albillo (4747)
Padre Rubio, 61 - 2º C
Madrid 29
SPAIN

Hi, Richard:

how are you ? As busy as always, I guess. I have let two months pass by before submitting material, because of the large amounts of materials already sub - mitted, and yet waiting for its publication. Now the stack seems to have lowered a little, so here are two more contributions for PPC CJ: (Calculator Journal, of course!)

a) BASIC SOFTWARE FOR THE 41c , an article describing some basic guidelines to convert a program written in BASIC language to the 41c, in a rather automatic way. This allows 41c user's to gain access to a vast amount of material and books, plenty of programs in BASIC for microcomputers. The article describes ideas and some rules to successfully carry out the translation to RPN. An enclosed example shows how to translate a game written in BASIC to the 41c, so as to create a program written using conventional 41c functions that does exactly the same as the BASIC program. The translation is performed "blindly", this is, the user does not have to know the algorithms of the original program to create the translation, only things needed are a general knowledge of BASIC language, and the BASIC program listing.

b) ROOTS OF POLYNOMIAL EQUATIONS , a program for the 41c, which solves the general equation of degree n, with real or complex coefficients, finding all its n roots, whether real or complex, to 10-digit accuracy. The degree, n, ranges from 1 to 132. Only input required are the (n+1) complex (or real) coefficients. Output are all n roots. Roots are stored as well, so the program is usable as subroutine. Program is optimized to be short (177 lines, 298 bytes) and fast (indirect addresses duplicated to increase speed). It is the most general version possible for polynomial root finders. Hope you'll like it.

Most unfortunately, I cannot send you a card with the program recorded on it, as my card reader is (once more!) being repared at HP. However, the enclosed listing has been exhaustively checked to ensure it being correct, and error-free.

Some comments: -I agree with the publication of the PPC Barcode Book. I do not have a wand yet, but I had a demo unit at home to test several weeks, and it worked nicely. It read absolutely all barcodes in V7 N5, including all programs and sparse barcodes. My wand was an 1D type. It took very little power from the batteries. In fact, the wand continued to read even with the BAT annunciator on (and my calculator has a very low BAT level; 1 hour after BAT appears, my calculator stops functioning properly)

-I also agree with Steve Flarity (5154) (see V7 N6 P13c): labels in the ROM may be as large as desired, because, in user's RAM, XROM"DECODE" is only 2 bytes ! The same is true regardless of being XROM"DD" or XROM"ABCDEFG", any XROM takes just 2 bytes. You didn't seem to understand it ! Be aware that any XROM call is 2 bytes, regardless of the length of the function's name being called. I point this out, because you mentioned: "... if the users RAM ... could call the ROM subroutines by their XROM number, a byte would be saved... ". You're wrong: XROM 29,01 is 2 bytes, as is XROM"1234567".

Well, that's all. Thank you, for you work so hard to keep PPC going on smoothly. Till the next letter:

Valentin Albillo (4747)

Final note : I also agree with Werner Frangen (2468)(see V7 N6 P7c) about the idea of asking members to evaluate programs published in PPC, and further publish the results, so that every author should have a feedback about how well his program was considered by the membership. It would be funny to see wonderful programs about synthetics being rated 0, and financial programs rated 10 !

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                                                        +
+  (((((((( BASIC  SOFTWARE  FOR  THE  41C ))))))))      +
+                                                        +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Let's be positive: whether you believe it or not ,the HP 41c is the heart of a true microcomputing system, not just a mere handheld programmable calculator, but a handheld (or briefcase) computer, with over 2.2 K of RAM, up to 32 K of external ROM, peripheral printer, mass storage devices, and last but not least, analog-digital interface via the wand. It has interface capabilities thru its 4 ports, and may handle any task using appropriate interface modules. It is quite possible to have (in a future) some data cartridge drive available for it, to store, retrieve, even chain programs or data, at a tremendous speed, and capable of storing as much as 200 K of programs in a single tape. Almost any desired input/output may be done if the proper peripheral is available. Now, you'll agree with me that this sounds more to microcomputer than to calculators, isn't it ?

Once we agree that the 41c is a microcomputer, we may expect it to be capable of running the programs a microcomputer will. However, a great deal of good programs for micros is written using BASIC (or FORTRAN, or PASCAL, or ...) language , and it seems to be inaccesible for the 41c, unless the algorithms used by the BASIC program are fully documented and understood,so that they can be rewritten using RPN. But, what if the program is not documented (except for the user's instructions), and only the listing is available ? Should we discourage ? Of course, if you have a pretty good patience, you may try to follow what the program does, in order to fully understand its function and all handle of data to arrive at the output. Once you understand the algorithms, it is relatively simple to translate them to RPN.

However, yo do not need to understand what the BASIC program does in order to write a translation for the 41c that will give the same outputs for the same inputs.

This article tries to stablish some guidelines to convert a BASIC program to the 41c, so that software in BASIC may be used in the 41c, even if the internal algorithms of the original program are not known or understood. The concept may be easily used for other languages.

This allows you to use programs written in BASIC, so gaining access to a vast amount of books and pacs full of programs in BASIC.

An included example shows how to translate a game in BASIC to the 41c, step by step, automatically.

This is not an exhaustive article on the subject, and not all BASIC programs may be translated for the 41c. It attempts to give you useful ideas, and remove your fear to BASIC software. You must be able to implement your own algorithms, and create new ones not explicitly pointed out here, like READ-DATA statements.

BASIC GUIDELINES

-first of all, a general knowledge of BASIC language is assumed, of course.

(1) Scan the BASIC listing. Delete all REMarks statements, all comments and titles. Mark with a * all lines referenced by GO TO, or IF ... THEN , or any other jumping statement. Also, mark with a * all statements FOR .

(2) Scan the listing and make a separate record of all the variables either simple (A,J,W5), or subscripted (R(I,J), A(7)) assigning a register for every element. This is, R05 is assigned to , say, P , R06 to A7, R07 thru R18 to R(0,0) thru R(1,5) (12 elements in total), be aware that the number of elements of a unidimensional array is given by a statement DIM in the listing. For instance, DIM A7(8) means that you

need to reserve 8+1 = 9 registers to store A7(0),A7(1),..., A7(8), and DIM Q(2,3) means you must reserve (2+1).(3+1), this is, 12 registers, to store Q(0,0),Q(0,1),...,Q(0,3), - Q(1,0),...,Q(2,3).

(3) If the program listing includes generation of random numbers reserve a storage register to store a seed. This is, if you find statements inclusing RND(number) or RANDOMIZE, reserve a register, to be used for the seed.

(4) Every time you find a FOR, reserve a register to be used as an index. It may be not needed if the stack is available, but reserve it anyway. If another cycle FOR starts within the first, save another register for its index. If another cycle FOR starts out of the first, it may use its register for indexing, so another register is not neccessary.

(5) Input statements should be translated as alpha prompts. For instance, INPUT W5 should be regarded as "W5 ?", PROMPT, or, if W5 is your move, as "MOVE ?" , PROMPT. Outputs are performed using "text" , ARCL nn, AVIEW , PSE , where nn is the register assigned to the variable being output, and "text" is some text describing the variable.

(6) Now, let's begin the translation. Scan the listing:

(7) every time you find an assignment statement, this is, an expresion of the form LET variable = ... , or : variable = .. compute the value using RPN for evaluation, remembering that each variable is a RCL nn, where nn is the number of its - assigned register. If the variable is a suscripted one (unidimensional or two-dimensional), it is equivalent to:

(1 dim)	index XEQ 99 RCL IND X	or	index 1 index 2 XEQ 99 RCL IND X	(2 dim)

where LBL 99 is a routine that computes the assigned register for the suscripted element given its index. Each suscripted array needs its own "computer" routine, so one may be LBL 99, another LBL 98, and so on. This routines are written later, so simply consider the sequence given before.

Once the value of the variable is computed, use STO nn to assign it to the variable. If the variable whose value has been computed is suscripted, use the same sequences as before, but using STO IND Y instead of RCL IND X. It is assumed that the value of the variable is previously stored in a scratch register before calling LBL 99 (say), then it is recalled, then STO IND Y will do the job.

(8) every time you find a line marked with * , inmediately put a label, starting from LBL 00 , and, enclosed within brackets, the number of the BASIC line.

For instance, * 130 LET A ... , inmediately generates the sequence LBL 00 (130)

As soon as you find a reference to a line, write down a GTO and, enclosed within brackets, the number of the basic line. For instance,

150 GO TO 280 should generate GTO (280)
230 IF ... THEN 1040 , must be ... GTO (1040)

also, if you find a NEXT statement, do not forget a GTO to the label of its corresponding FOR statement. For instance:

128 FOR ...
... ...
167 NEXT , must be , ... GTO (128)

(9) a FOR-NEXT cycle must include a register to hold the index.

If the index is a computed value, proceed to the computation, then assign the value to the index (STO in the index register) If the step is going to be such as to increment the index value in each cycle, use ISG , otherwise, DSE. For instance,

```
130 FOR I = 2 TO 30 STEP 3   (label of FOR , LBL 28, say)
... ... ... ...
240 NEXT I
```

should be translated as (I register is 17, say) :

```
2.03003        ... set-up of the index
STO 17         ... I is assigned a value
LBL 28         ... the cycle begins
... ...            if I is used, you should take INT(I)
ISG 17         ... the NEXT. First, get next value of I, then
GTO 28             test to see if the cycle is done. If not,
                   return to FOR, otherwise, continue.
```

(10) Conditionals are translated as conditionals in the calculator. For instance, the sentence:

```
250 IF R(E,L)>= P THEN 320
```

is translated as follows: assume E is assigned to R17, L to R01, P to R16, and LBL 99 computes the address of R(x,y):

```
RCL 17       ... get E , first index   (or suscript)
RCL 01       ... get L , second index  (or subscript)
XEQ 99       ... get the address of R(E,L)
RCL IND X    ... get R(E,L)
RCL 16       ... get P
X<=Y?        ... perform the test
GTO (320)    ... if the test is met, perform the jump.
```

(11) Once the whole program is written down as a sequence of 41c instructions, edit the program:

-write the routines to compute the address of a subscripted variable, assuming the index(es) are in X and Y , and the address should be returned to X. For instance:

if R(0,0) thru R(1,5) are assigned to R03 thru R14, and R is dimensioned 1,5 , then if I is in Y and J is in X, the address for R(I,J) is given by

```
LBL 99 , 3 , + , X()Y , 6 , * , + , RTN
```

for instance, R(1,3) is in R12 (3+3+6*1 = 12)

-scan all GTO (...) , and change the contents of the brackets for the label corresponding to that line. This is:

```
LBL 07 (340)
...
GTO (340)    should be changed to GTO 07
```

-set the initial conditions: prompt for a random seed, set the display to FIX 0, CF 29, insert BEEP, etc.

Resuming, if all goes well, you'll have a program for the 41c - which does what the original BASIC program did. The program should work, but if you feel capable, optimize it. Once you know the program works (run an example), put aside the BASIC program, and put all your knowledge to the task of optimizing your RPN program: save registers, use stack for indexing, suppress innecesary printing

Now, let's see an example of a translation. The following program in BASIC is given. All remarks have been previously supressed, and the lines referenced by GO TO's, conditionals and NEXT are marked:

Original BASIC program

```
  20 DIM R(1,5)
  25 L=Ø : E=Ø
* 30 FOR I = Ø TO 5
  40 R(1,I) = 4 : R(0,I) = 4
  60 NEXT I
* 70 A = Ø : B = Ø
  90 P = INT((13*RND(1)+9)/2)*2+1
*100 IF P = 1 THEN 530
 110 PRINT "THERE ARE";P;"CHIPS ON THE BOARD"
*120 E1 = E : L1 = L
 140 E = (A/2 - INT(A/2))*2
 150 L = INT((P/6 - INT(P/6))*6 + .5)
 160 IF R(E,L)>= P  THEN 320
 170 M = R(E,L)
 180 IF M<=Ø THEN 370
 190 P = P-M
 200 IF M=1 THEN 510
 210 PRINT "I TAKE";M;"CHIPS LEAVING";P;"  YOUR MOVE"
*220 B = B+M
*230 INPUT M : M = INT(M)
 250 IF M<1 THEN 450 : IF M>4 THEN 460
 270 IF M>P THEN 460 : IF M=P THEN 360
 290 P = P-M : A = A + M : GO TO 100
*320 IF P=1 THEN 550
 330 PRINT "I TAKE";P;"CHIPS"
*340 R(E,L) = P : B = B + P
*360 IF B/2 = INT(B/2) THEN 420
*370 PRINT "YOU WIN"
 390 IF R(E,L) = 1 THEN 480
 400 R(E,L) = R(E,L) - 1  : GOTO 70
*420 PRINT "I WIN" : GOTO 70
*450 IF M=Ø THEN 570
*460 PRINT "ILLEGAL  ,   YOUR MOVE" : GOTO 230
*480 IF R(E1,L1) = 1 THEN 70
 490 R(E1,L1) = R(E1,L1) - 1  : GOTO 70
*510 PRINT "I TAKE 1 CHIP LEAVING";P;"  YOUR MOVE" : GOTO 220
*530 PRINT "THERE IS 1 CHIP ON THE BOARD" : GOTO 120
*550 "I TAKE 1 CHIP" : GOTO 340
*570 END
```

This is the BASIC listing of a game called "Even wins" taken from "BASIC COMPUTER GAMES" by David H. Ahl , edited by - Creative computing press. The game is played as follows: 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 one finishes, other starts.

What is really remarkable is that the computer starts out knowing only the rules of the game, and a learning strategy allows it to play gradually better and better, until it is extremely difficult to beat. Thus, this is a learning program: the better you play against it, the faster it learns to play well. After 20 games in a row, it is almost unbeatable.

We are told nothing about the internal algorithms. Can we make a translation for the 41c? YES !!! We <u>can</u> . Let's start:

(1) REMarks and comments have been already deleted. The lines referenced by GTO's, conditionals and NEXT are marked, too.

(2) There are 8 simple variables: L,B,A,P,E,E1,L1,M, and one 2-dimensional variable R(I,J). Its DIM statement at line 20 tells us it is a 2x6 matrix, 12 elements total. We assign registers as follows: (there is also a RND statement at line 90, a seed in needed): 00=seed , 01=L, 02=B , 03 thru 14= R(0,0) thru R(1,5), 15=A , 16 = P , 17 = E, 18=E1 , 19=L1, 20=M
   - due to the RND (random) statement, you should input a seed (any number between 0 and 1): at the beginning simply, key in seed, XEQ"EVEN", and the rest is

-this is most useful in tabular form:

00 = seed	11 = R(1,2)
01 = L	12 = R(1,3)
02 = B	13 = R(1,4)
03 = R(0,0)	14 = R(1,5)
04 = R(0,1)	15 = A
05 = R(0,2)	16 = P
06 = R(0,3)	17 = E
07 = R(0,4)	18 = E1
08 = R(0,5)	19 = L1
09 = R(1,0)	20 = M
10 = R(1,1)	

-so, all variables have an assigned 41c register, and thus, an address. The auxiliar routine to compute the address of each element in the 2-dimensional array R(I,J) is :

```
LBL 99     6     where I is in Y,
 3         *           J is in X,
 +         +     the address is
X()Y      RTN    in X at RTN.
```

```
the program is given a name:  LBL "EVEN"
the seed is stored         :  STO 00
Ø            ... line 25 is translated: L & E get a value of Ø
STO 01           L is assigned to R01 and E to R 17
STO 17
3.014        ... lines 30 thru 60: a loop is set to assign the
4                value 4 to all R(I,J). After the loop is execu-
LBL 00           ted , R03 thru R14 all contain 4's. This is just
STO IND Y        what the original loop does, assign 4 to all
ISG Y            values of R(I,J)
GTO 00
LBL 01(70) ... line 70 is marked, so we set up a LBL. A & B
Ø                get a value of Ø . A is R15, B is R02
STO 15
STO 02
RCL 00 , R-D, FRC , STO 00, ... line 90: this generates a ran -
13 , * , 9 , + , 2 , / ,        dom number, and computes the va
INT , ST+X , 1 , + ,            lue of P using RPN. Then P,which
STO 16                          is R16, is assigned that value.
LBL 02(100)   ... line 100: P (R16) is tested against 1. If the
RCL 16              test is met, jump to 530
1                   Otherwise, go on with next line
X=Y?
GTO(530)
"THERE ARE ", ARCL 16 , "├ CHIPS" ... line 110, the PRINT state
AVIEW , PSE                           ment. R16 is P
LBL 03(120)
RCL 17, STO 18, RCL 01, STO 19   ... line 120, E1=E, L1=L
RCL 15, 2 , / , FRC, ST+X ,    ... line 140: A/2-INT(A/2) is the
STO 17                             same as FRC(A/2). R17 is E
RCL 16, 6 , / , FRC, 6 , * ,   ... line 150: L (R01) gets its
.5 , + , INT , STO 01              computed value
RCL 17, RCL 01, XEQ 99       ... line 160: get address of R(E,L)
RCL IND X                        get R(E,L) itself
RCL 16 , X<=Y?, GTO(320)         test against P (R16)
X()Y , STO 20     ... line 170: M was in Y. R20 is assigned to M
X<=0? , GTO (370)   ... line 180: M is tested (& remains in X)
ST-16               ... line 190: M is substracted from P
1 , X=Y? , GTO(510)  ... line 200: M (still in X)is tested
"I TAKE " , ARCL 20 , AVIEW , BEEP, ... line 210: M is R20, P
PSE, "LEAVE ", ARCL 16 , AVIEW ,          is R16
PSE , "YOUR MOVE" , AVIEW, PSE
LBL 04(220)         ... line 220: M , which is in R20, is
RCL 20 , ST+ 02         added to the previous value of B (R02)
LBL 05(230)                  ... line 230: M is requested, and
"M=?" , PROMPT, INT , STO 20      its INT is assigned to R20
1, X()Y, X<Y?, GTO(450) ,  ... line 250: M (was in X), is tes-
4, X < Y? , GTO(460)           ted twice
RDN, RCL 16, X<Y?,GTO(460) ... line 270: M (was in Y) is tested
X=Y? , GTO(360)                against P (R16) twice
RCL 20, ST-16, ST+15, GTO(100) ... line 290: P=P-M, A=A+M
LBL 06(320)        ... line 320: P (R16) is tested
RCL 16, 1 , X=Y? , GTO(550)
"I TAKE ", ARCL 16, AVIEW, PSE, BEEP ... line 330
LBL 07(340)     ... line 340:
```

```
RCL 16 , STO IND Y , ST+ 02  ... assign P to R(E,L) & add to B
LBL 08(360)                        ... line 360: test B to see if
RCL 02,2,/,FRC,X=0? , GTO(420)         it is even (FRC(B/2)=Ø)
LBL 09(370)                        ... line 370 : output the messa-
"YOU WIN" , AVIEW, BEEP , PSE                    ge
RCL 17, RCL 01, XEQ 99                 ... line 390: get addr. R(E,L)
RCL IND X , 1 , X=Y ?, GTO(480)     get R(E,L) & test
- , STO IND Y, GTO(70)          ... line 400: 1 substract.from R(,)
LBL 10(420)                            ...line 420: output the
"I WIN" , AVIEW, BEEP, PSE, GTO(70)      message and jump
LBL 11(450),RCL 20,X=0?,GTO(570)  ...line 450: M (R20) is tested
LBL 12(460),"ILLEGAL",AVIEW,PSE,GTO(230)  ... Illegal message
LBL 13(480)                         ... line 480: R(E1,L1) is first
RCL 18,RCL 19,XEQ 99,RCL IND X         recalled ,
1, X=Y?, GTO(70)                       then tested
-, STO IND Y, GTO(70)           ... line 490: 1 is substracted
LBL 14(510)                          ... line 510: all messages are
"I TAKE 1",AVIEW,BEEP,PSE,              output. R16 is P
"LEAVE ",ARCL 16, AVIEW, PSE
"YOUR MOVE", AVIEW, PSE,GTO(220)        after output, jump to 220
LBL 15(530)
"THERE IS 1 CHIP",AVIEW,PSE,GTO(120)  output message & jump
LBL 16(550)
"I TAKE 1",AVIEW,BEEP,PSE,GTO(340)   output message & jump
LBL 17(570) , END                      end of program
```

So, the work is done !!! Now, simply compile all the GTO's (This is, change, say, GTO(70) by GTO 01, because LBL 01 includes 70 within brackets, and so on). The resulting HP-41C program is as follows: (in condensed form)(to save space!)

```
01 LBL"EVEN"
STO 00, Ø, STO 01, STO 17, 3.014, 4, LBL 00, STO IND Y, ISG Y,
GTO 00, LBL 01, Ø, STO 15, STO 02, RCL 00, R-D, FRC, STO 00, 13,
*, 9, +, 2, /, INT, ST+X, 1, +, STO 16, LBL 02, RCL 16, 1, X=Y?,
GTO 15, "THERE ARE ", ARCL 16, "├CHIPS", AVIEW, PSE,LBL 03
RCL 17, STO 18, RCL 01, STO 19, RCL 15, 2, /, FRC, ST+X, STO 17,
RCL 16, 6, /, FRC, 6, *, .5, +, INT, STO 01, RCL 17, RCL 01,
XEQ 99, RCL IND X, RCL 16, X<=Y?, GTO 06, X()Y, STO 20, X<=0?,
GTO 09, ST-16, 1, X=Y?, GTO 14, "I TAKE ", ARCL 20, AVIEW, BEEP,
PSE, "LEAVE ", ARCL 16, AVIEW, PSE, "YOUR MOVE", AVIEW, PSE ,
LBL 04
RCL 20, ST+ 02, LBL 05, "M=?", PROMPT, INT, STO 20, 1, X()Y,
X<Y?, GTO 11, 4, X<Y?, GTO 12, RDN, RCL 16, X<Y?,GTO 12, X=Y? ,
GTO 08, RCL 20, ST- 16, ST+ 15, GTO 02, LBL 06, RCL 16, 1, X=Y?,
GTO 16, "I TAKE ", ARCL 16, AVIEW, PSE, BEEP, LBL 07
RCL 17, RCL 01, XEQ 99, RCL 16, STO IND Y, ST+ 02, LBL 08, RCL 02
2, /, FRC, X=0?, GTO 10, LBL 09, "YOU WIN", AVIEW, BEEP, PSE,
RCL 17, RCL 01, XEQ 99, RCL IND X, 1, X=Y?, GTO 13, -, STO IND Y,
GTO 01, LBL 10, "I WIN", AVIEW, BEEP, PSE, GTO 01, LBL 11
RCL 20, X=0?, GTO 17, LBL 12, "ILLEGAL", AVIEW, PSE, GTO 05,
LBL 13, RCL 18, RCL 19, XEQ 99,RCL IND X, 1, X=Y?, GTO 01, -,
STO IND Y, GTO 01, LBL 14, "I TAKE 1", AVIEW, BEEP, PSE, "LEAVE "
ARCL 16, AVIEW, PSE, "YOUR MOVE", AVIEW, PSE, GTO 04, LBL 15
"THERE IS 1 CHIP", AVIEW, PSE, GTO 03, LBL 16 , "I TAKE 1",
AVIEW, BEEP, PSE, GTO 07, LBL 17, END
```

of course, if you want to get a good display of messages, insert FIX 0, CF 29 after 01 LBL"EVEN", and the auxiliar routine, LBL 99, 3, +, X()Y, 6, *, +, RTN, must be inserted somewhere, for instance, add a RTN after LBL 17, then this auxiliar routine.

You now have a program which is an exact translation of the original BASIC program, and performs exactly the same function. Now, you can set the task of optimize it, to reduce space or run time, or use it as it is now, as well.

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,

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

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

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_o 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 z1
           R/S    → V=imag part of z1
    R/S  → ROOT 2 → U=real part of z2
           R/S    → V=imag part of z2
         ... ... ... ...... ... ...
         → ROOT n → U=real part of zn
           R/S    → V=imag part of zn
           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 $\underline{5x^6 -4x^5 -3x^4 +8x^3 +8x^2 -2x +7 = 0}$

→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=a_0 (u_n)	real parts of coeff. (& roots)
01=add. a_n	10=a_1 (u_{n-1})	
02=add. b_n n	... ...	
03=r.p. of z	n+9=a_n (a_n)	
04=i.p. of z	n+10=b_0 (v_n)	imag. parts of coeff. (& roots)
05=aux.(a_n)	n+11=b_1 (v_{n-1})	
06=aux.(b_n)	... ...	
07=auxiliar	2n+10=b_n (b_n)	
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)