PPX Exchange

Vol. 5 Number 4 Copyright 1981 July/August 1981

Advanced Memory Operations

By Don O'Grady

In a continued effort to help TI-59 programmers get more computing power from their programmable calculators, this article will describe the techniques applied in treating program memory as data through the use of STO and RCL. There are many opportunities to apply such a process and even a novice programmer should require only a few moments to come up with several examples.

One of the more obvious uses of this technique is to load data registers by entering the appropriate information in the form of program instructions and then converting this information to data memory through repartitioning. The information may then be retrieved as though it were stored in the data register by conventional means. This operation is especially useful when numbers involving guard digits must be entered into a data register as in the following example.

continued on page 2

Inverse Days Between Dates

By Jay Claborn

Program 20 of the Master Library module allows one to calculate the number of days between two dates. Several PPX members have inquired if the inverse function (given a number of days and a date, find the second date) is also available. A quick check of the PPX Catalog revealed that we did not have such a program. The task seemed easy enough, so I decided to write the program myself. The purpose of this article is not only to provide you with a useful piece of software, but also to give you a guided tour of the development of this or any other program.

As with all problems, the first step was to **CLEARLY DEFINE THE PROBLEM.** Here, the task was to produce a TI-59 calculator program that would allow the user to enter a date and a number of days from that date and have the calculator find the resulting date.

The second step was to **REVIEW AVAILABLE RESOURCES.** In this case I read the Master Library manual section which describes the method of ML-20. I also downloaded ML-20 into main memory and listed it on the printer. The ML-20 documentation revealed that for any date a unique "factor" could be calculated and that the number of

continued on page 8

TI-59 Fast Mode

By Palmer O. Hanson, Jr.

Editor's Note: Through experimenting with unusual key sequences, some TI-59 owners have found that their calculators will respond to certain illegal sequences in ways not intended by the designers. A few of the byproducts of these experiments include fractured display graphics, access to hardwired functions, indirect hierarchy operations, printer interrupts, and the subject of this article — a fast mode.

Due to algorithm requirements, internal operations are normally executed in scientific notation. In order to format the trace and SST output properly, programs run in main memory convert to floating point format and back again between each step. As these functions are not provided for module programs, this conversion is not necessary, making programs that run in a module execute faster than programs that execute in main memory.

The sequence described in this article supresses the floating point conversion, which increases speed. Other known results outlined in this article are the discoveries of Mr. Hanson and his peers. This sequence is not a TI designed feature, but rather a machine "quirk"; therefore, there may be additional unpredictable aspects of its behavior.

The user should take extreme care to observe the restrictions outlined below. The consequences of failing to do so can be as severe as a "system crash" from which control can only be regained by turning the calculator off and on. Use of the fast mode may have unexpected consequences and cannot be endorsed by TI. Nor will TI be responsible for software damage caused thereby. This item is presented solely as an item of interest to TI-59 owners.

Soon after the TI-59 became available users found that code executed from a library module would run faster than the same code when downloaded into user memory. As early as December 1977, Tom Ferguson reported the effect in Volume 2 Number 12 of the "52 Notes." He found that the routine C part of ML-16 would take approximately twice as long to find 69 factorial with the code in user memory than the same code when called from the Solid State Software™ module. Others found a much smaller difference in speed when executing the code for the module diagnostic (Pgm 01 SBR =). More recently in a Letter to the Editor in Volume 4 Number 6 of the "Exchange" Herman Burstein reported a two to one speed differential when using program 20 of the Statistics module. He asked, "Is there by any chance a way to obtain comparably fast running time from a program in (user) memory?" The answer is YES! The technique has

continued on page 4

Memory Operations *(continued)*

Here, the objective is to place the value -8.124436835231 . x 10^{45} in data register 47. This data register corresponds to program memory locations 576 through 583. The first step in the procedure is to partition your calculator by pressing 4 2nd OP 17. Then key in EE D lnx 1/X 2nd Nop RCL 24 RST starting at step 576. The explanation of how this works is presented below; but for now, verify that the correct number has been entered by repartitioning your calculator to at least fifty data registers and recalling register 47.

This technique will generate some strange looking sequences; however, it can be extremely useful in storing frequently used constants. In this case, the technique is especially useful as the value in question requires greater precision than the display can handle. For example, when working with a TI-58C where programs and data cannot be permanently recorded on magnetic cards, entering such constants as part of the program and then repartitioning at the appropriate time can speed program entry.

Other examples include modifying and relocating existing program instructions. Consider a situation in which a long sequence can be modified to handle multiple purposes by simply changing a few instructions as switching a "+" instruction to a "−". Or, what about the possibility of moving blocks of code to make room for new program instructions? And, of course, all of this and much more can be handled from either the keyboard or under program control.

STRUCTURE

In order to understand how to apply this technique, it is first necessary to obtain an in-depth knowledge of how the calculator's memory is structured. As most TI-59 programmers are already familiar with the eight program step per data register trade-off, this subject will not be covered here. If you need a review, see page V-42 of "Personal Programming." Before proceeding any further it would be advantageous to assign each position within a register a name. In order to avoid confusion with numeric data, alphabetic names are used. The names are assigned in relation to the position of the two-digit program codes seen in program memory and are repeated for each block of eight program instructions (the equivalent of one data register) where the address of the first instruction is an even multiple of eight.

	NAME OF POSITION	
Location of Step in Block	First Digit of Program Code	Second Digit of Program Code
1	A	B
2	C	D
3	E	F
4	G	H
5	I	J
6	K	L
7	M	N
8	O	P

Returning to the example presented in the introduction:

Program Step	Program Instruction	Program Code	Corresponding Names
576	EE	52	AB
577	D	14	CD
578	lnx	23	EF
579	1/X	35	GH
580	Nop	68	IJ
581	RCL	43	KL
582	24	24	MN
583	RST	81	OP

Now, comparing the above to the value in data register 47, the following relationship can be observed:

Register 47

-8.124436835231 x 10^{45}

O.PMNKLIJGHEFC x 10^{DA}

Admittedly, this relationship is not immediately obvious as the duplication of several of the digits in order to fill all sixteen positions obscures the one-to-one correspondence that does exist. However, experimentation will show that the above description is correct.

Two things remain unexplained by the above. First, the origin of the negative sign in front of the mantissa has not been accounted for; and second, the value stored in position B does not appear in the number retrieved from the data register. Obviously, these factors must be related. As will be shown below, the value of position B controls the signs of the mantissa and of the exponent, as well as telling the calculator whether or not an error state exists in the register. But before continuing with the details concerning position B, a formal description of the structure of a data register will be presented.

A data register consists of sixteen digits or positions arranged in the following format where the letters assigned to each position correspond to the definition given for the same memory area treated as a group of eight program instructions. The apparently "backwards" orientation of the positions coincides with the fact that program steps and data registers are numbered in opposite directions as illustrated in the page V-42 of "Personal Programming."

Note, that even though the calculator may display the register's contents in floating point notation, it is always stored in scientific notation. The mantissa to the number is stored in the first thirteen digits. Positions E, F, and C hold the guard digits which are retained to improve accuracy. The next two digits, D and A, are reserved for the exponent. The last position in the register (B) is used to control the sign of the mantissa and exponent.

ADDRESS CHANGES

In order to ensure uninterrupted service, please submit address changes to PPX at lease six weeks prior to the effective date of the change. Send your name, membership number, old and new addresses to:

PPX
P.O. Box 53
Lubbock, TX 79408

REPRESENTING DIGITS

Each position in a register is made up of four bits. A bit is a memory location which can have two states, on and off. A bit that is turned on is represented by the number 1; 0 is used if the bit is turned off. By assigning values to each of the four bits the digits can be represented as illustrated by the following:

Digit	Bit 3 $2^3 = 8$	Bit 2 $2^2 = 4$	Bit 1 $2^1 = 2$	Bit 0 $2^0 = 1$
0 =	0	0	0	0
1 =	0	0	0	1
2 =	0	0	1	0
3 =	0	0	1	1
4 =	0	1	0	0
5 =	0	1	0	1
6 =	0	1	1	0
7 =	0	1	1	1
8 =	1	0	0	0
9 =	1	0	0	1

For example:

$6 = 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
$6 = 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1$
$6 = 0 + 4 + 2 + 0$

The numeric value of the digit stored in the position B has no meaning in itself. Its meaning is derived from the assignments of the bits which make up position B. Some of these bits have multiple assignments; but for purposes of this discussion the assignments given below are all that are necessary.

Bit 0 - No assignment
Bit 1 - Indicates a Negative Mantissa when On
Bit 2 - Indicates a Negative Exponent when On
Bit 3 - Indicates the Existence of an Error State when On

Combining these assignments with the chart presented above provides the following information concerning the meaning of a digit stored in position B when treated as a data memory.

Digit	0,1	2,3	4,5	6,7
Mantissa	+	−	+	−
Exponent	+	+	−	−

Additionally, a value of either 8 or 9 indicates that an error state exists in the data register.

USING STO AND RCL

Obviously, the use of the data memory instructions is necessary when treating program instructions as data. Unfortunately, these instructions do not always "see" the entire eight program codes and in other cases will actually modify their format.

The first problem lies in the fact that bit 0 of position B has no meaning as a part of data memory. Consequently, this location is effectively ignored by the data memory instructions. To illustrate this point, key in the sequence (D lnx 1/X 2nd Nop RCL 24 RST beginning at step 576. Now, repartition as in the introductory example and examine the contents of register 47. The value retrieved is identical to the value observed using the previous sequence. To continue the illustration restore the value called to the display by pressing STO 47, repartition the calculator, and examine program step 576. Upon doing this you will find that the "(" instruction has been changed to "EE" as in the original example.

To understand how this has happened, note that the "(" instruction placed the digit "3" in position B. In recalling and storing register 47, bit 0 of position B is discarded, or turned off, causing this 3 to become a 2.

The second problem concerns the assignment of bit 3 of position B. Turning on this bit by placing 8 or 9 in position B tells the calculator that an error state exists in the corresponding data register. To see this, modify the above example once more by beginning the sequence with a 2nd Fix instruction at step 576. Now, recalling register 47 places a flashing $9.99999999 \times 10^{99}$ in the display because the error state has been interpreted as an overflow condition and filled the display accordingly. Attempting to transfer or restore the information found in register 47 is impossible in this case because the display can only move nines into any data register. Executing a STO 47 and examining program steps 576-583 reveals a 2nd List command (program code 90) at step 576 and 2nd Prt instructions (program code 99) at the remaining locations. Observe that even though position B has been cleared the overflow state still exists due to the presence of all nines in the data register.

The final problem involves the value of the digit placed in position 0. This position is used to hold the most significant digit of the mantissa of the number in the data register. However, if this value is a zero, the data register operations select position P as the most significant digit because leading zeroes are dropped from the display. This problem can be illustrated by changing the last program step in the sample sequence from "RST" to the digit "1". After keying in this sequence, repartitioning, and recalling register 47 the value displayed is $-1.2443684 \times 10^{44}$ instead of 0.1244368×10^{45} as some might expect. This difference occurs because the latter is an improper representation of a number in scientific notation. Storing the displayed number in register 47 and examining program memory once more reveals that the entire sequence has been modified due to shifting the digits of the mantissa and reducing the exponent of the original number.

BENDING THE RULES

When using this technique to simply load data registers, the restrictions discussed above pose no problems as there is no need to use bits "0" and "3" of position B. Also, even if you format your number incorrectly, the restriction concerning position "0" is not a problem as the calculator will correct the format for you without altering the actual value of the number.

The one problem that might arise is a number that requires the use of an instruction not accepted by program memory such as LRN or SST. Changing the value to be stored in register 47 to $-8.124435635231 \times 10^{45}$ requires changing step 580 of the input sequence to a 2nd Del instruction. The program code for this instruction (56) is easily entered by

continued on page 4

pressing STO 56 to create the desired code and then deleting the unwanted STO instruction as in the following sequence: EE D Inx 1/X STO 56 BST BST 2nd Del SST RCL 24 RST. Verify that this procedure worked by retrieving the contents of register 47.

If the objective is to modify or relocate pieces of program memory, however, writing general purpose routines is impractical as strict attention must be given to these rules. But developing these routines is relatively easy to handle for a specific program by inserting program steps which have no apparent use other than as "space-savers."

If the eighth step in a block of eight program instructions has a zero for the first digit in the program code simply place a 2nd Nop instruction in this location instead and push the required instruction down one step. Likewise, if the first instruction in the octet requires that an "8" or "9" must be kept in the position for the second digit, place another "space-saver" in this location and load the desired instruction into the next step. In the latter case, however, 2nd NOP may not be used, as its program code (68) cannot be maintained in this position when treated as data memory. Program codes commonly used to hold this position are 2nd Deg, 2nd Rad, and 2nd Grad as the angular mode setting frequently has no effect on the program; and if it does, one particular angular mode is typically needed throughout the program.

A concentrated effort on the part of the programmer is needed when dealing with the techniques discussed here; but the results can be very rewarding.

Fast Mode *(continued)*

come to be called "Fast Mode".

Early in 1980 Martin Neef of West Germany discovered program sequences such that subsequently entered programs would run at approximately twice the speed of normal calculator operation (TI PPC Notes, Volume 5 Number 6). One sequence which will work is, starting from powerup:

GTO 005 LRN Pgm 02 SBR 239 9 LRN RST R/S

The calculator will run for a short period of time and stop with a zero in the display. Pressing LRN yields 000 09 in the display. Another LRN yields 0 in the display. Yet another LRN yields 000 00 in the display. At this point the calculator is in the learn mode at program location 000. The user may key in his program in a normal manner, and, if appropriate restrictions are observed, the program will run in the fast mode. Entry to the fast mode with the Neef technique clears all memory, the equivalent of a combined CP and CMs from the keyboard. Manual key-in of short programs was acceptable with this method, but the capability to run fast mode cards would have been advantageous. Attempts to load memory with magnetic cards in the standard manner were, however, unsuccessful. In mid-1980 I discovered that magnetic cards could be entered in fast mode in a "load-and-go" sort of operation under program control. The technique is similar to the "Reading a Card from a Program" idea described on page VII-5 of "Personal Programming."

Some insight as to what might be happening during fast mode initialization can be gained by examining the downloaded ML-02 program in the neighborhood of location 239.

237	29	CP
238	67	EQ
239	03	03
240	31	31
241	43	RCL
242	07	07

Entry to the library program at location 239 reads the code as 03 LRN, where the LRN is an unintended code caused by the use of the PGM 02 SBR 239 sequence to enter the module in the middle of an absolute address. The calculator logic seems to get confused by the code 31 (LRN) read from the library module and fast mode somehow results. A code 31 in user memory will simply cause the calculator to enter the LRN mode and stop with a LRN type display for the following program location. Fast mode cannot be attained manually by using the proper keystrokes starting with the program pointer at 005. Only the callup of the unintended code 31 using a program in user memory provides fast mode entry.

Use of the Master Library module is required if the above sequence is to provide entry into the fast mode. I have searched for unintended code 31's in the other modules with the exception of the Securities Analysis and the Agriculture modules. Of the 48 code 31's found, only the one at location 423 of program LE-11 in the Leisure Library will also provide entry into the fast mode. With the Leisure Library module installed the use of the sequence Pgm 11 SBR 423 9 will provide appropriate fast mode entry.

The **KNOWN** limitations and restrictions when using fast mode include:

1. Entry to the fast mode clears all memory.
2. Entry to fast mode returns the calculator to the powerup partitioning of 479.59 (6 OP 17). Subsequent program control must be used to change to any other partitioning.
3. Entry to the fast mode leaves the calculator in the Fix 0 mode with the Master Library and in the Fix 2 mode with the Leisure Library. Subsequent program control must be used to change to any other Fix mode.
4. Entry to the fast mode resets all the flags if there are zeroes in memory locations 011 through 015. Use of any other code in those locations may result in other flag status. For example, an Op 00 command at locations 014-015 will set flags 1 and 7.
5. Programs may be entered or corrected from the keyboard using the LRN mode. Use of the edit functions (SST, BST, Ins, and Del) is permitted.
6. Entry of numbers into the display using the number keys, + / −, the decimal point, CE, CLR and EE is permissible.
7. The use of most of the function keys will typically remove the calculator from the fast mode as a minimum. Other erratic results have been encountered including a "crash" from which recovery can be made only by turning the calculator off. More specific details on the use of selected function keys follow.
8. Use of the RST from the keyboard with the calculator stopped will remove the calculator from the fast mode. A RST used in a fast mode program removes the calculator from fast mode, and may cause a "crash".
9. Use of R/S from the keyboard can be used to start a program in the fast mode. SBR nnn may be used from the keyboard to start a program at a location different from the current location. SBR followed by a user defined label, say SBR A or simply pressing a user defined label will remove the calculator from the fast mode and may cause other complications.
10. Once the program is running in fast mode it cannot be

stopped by pressing R/S or RST. If a stop is not provided by program control it will be necessary to turn the calculator off to regain control.

11. While running in the fast mode the calculator will not recognize a R/S command unless it is immediately preceded by CLR 2nd CLR (Code 20) Pause Prt, or the sequence EE INV EE.
12. The RTN command (code 92 entered via INV SBR) cannot be used as an alternate to the R/S command (code 91) even if the RTN is preceded by one of the commands listed in paragraph 11.
13. An attempt to change the location of the program pointer from the keyboard with a GTO nnn sequence will remove the calculator from the fast mode and may cause other complications.
14. Subroutines may not be used in the fast mode. This includes the callup of library programs with sequences such as Pgm mm with any of the options on page V-62 of "Personal Programming" or even with the Pgm mm SBR nnn sequence which was used to enter the fast mode.
15. Neither user defined labels nor common labels can be used with the transfer instructions such as the GTO, the "t" register comparisons, the DSZ, or the flag tests. In other words, the transfer address must always be an absolute address.
16. Some operations do not run at increased speed in the fast mode. Examples include the trigonometric functions, and the statistics and conversion functions, which operate at the fast mode speed even in normal mode. The heavy use of these functions in the module diagnostic program explains why the typical two-to-one speed advantage for fast mode does not occur for that program.

A sample program provides an introduction to the use of the fast mode. Locations 000 through 015 provide for the fast mode entry. Locations 016 through 035 and 240 through 253 provide for safe entry of four data banks using the "load-and-go" technique and a prompting message for the program to follow. Locations 254 through 275 provide a short demonstration program which is the equivalent of the one first used by Neef in his demonstration of the fast mode. A bank 4 card is also required with the number 100 in register 02. All three card sides should be recorded with the partitioning at 479.59 (6 Op 17). The sequence of operator actions to run the program are, starting from powerup:

1. Enter bank 1. The display will show a one.
2. Press RST and then R/S. The calculator will run for a short time and stop with a zero in the display. The calculator is now in the fast mode, but the memory has been cleared.
3. Re-enter bank 1. The printer will print a one to indicate which bank was loaded. The display will show a zero, indicating it is ready to load the next bank. This step was needed to restore the bank 1 data which was cleared by the fast mode entry.
4. Enter bank 2. The printer will print a two to indicate which bank was loaded. The display will show a zero, indicating that it is ready to load the next bank.
5. Press 3 and then +/− to enter minus three in the display. Enter bank 4. The printer will print a minus three as a reminder that another bank was forced into bank 3. This step loads the number 100 into data register 32 where it will be used by the program. The display will show a zero, indicating it is ready to load the next bank.
6. Enter bank 4. The display will show a zero and the printer will print a four followed by the prompting message "RUN?".
7. Press R/S to start the demonstration program. After about 37 seconds the calculator will stop with a 200 on the printer and in the display.
8. Press RST to remove the calculator from the fast mode, then press A. The calculator will stop after about 70 seconds with 200 on the printer and in the display, thus demonstrating the speed advantage of the fast mode.

FAST MODE DEMONSTRATION PROGRAM

Bank 1

000	00	0	012	00	0	024	25	CLR
001	00	0	013	00	0	025	91	R/S
002	00	0	014	00	0	026	99	PRT
003	00	0	015	00	0	027	25	CLR
004	00	0	016	69	OP	028	91	R/S
005	36	PGM	017	00	00	029	99	PRT
006	02	02	018	22	INV	030	20	CLR
007	71	SBR	019	58	FIX	031	91	R/S
008	02	02	020	22	INV	032	99	PRT
009	39	39	021	57	ENG	033	61	GTO
010	09	9	022	01	1	034	02	02
011	00	0	023	99	PRT	035	40	40

Bank 2

240	03	3	252	25	CLR	264	95	=
241	05	5	253	91	R/S	265	32	X:T
242	04	4	254	76	LBL	266	69	OP
243	01	1	255	11	A	267	20	20
244	03	3	256	00	0	268	43	RCL
245	01	1	257	42	STO	269	00	00
246	07	7	258	00	00	270	22	INV
247	01	1	259	43	RCL	271	67	EQ
248	69	OP	260	02	02	272	02	.02
249	02	02	261	85	+	273	66	66
250	69	OP	262	43	RCL	274	99	PRT
251	05	05	263	32	32	275	91	R/S

Bank 4

0.	00
0.	01
100.	02
0.	03
0.	04

The PPX Exchange is published bimonthly and is the only newsletter published by Texas Instruments for TI-59 owners. Members are invited to contribute articles and items of general interest to other TI-59 users. Authors of accepted feature articles for the newsletter will receive their choice of either a one year complimentary PPX membership or a Solid State Software™ module. Please double-space and type all submissions, and forward them to:

Texas Instruments, PPX
P.O. Box 53
Lubbock, Texas 79408
Attn: PPX Exchange Editor

COMPLEX NEWTON RAPHSON ROOT FINDER

By Jay Claborn

This program uses the iteration process known as Newton Raphson to find the roots of an up to 17th order polynomial with complex coefficients.

METHOD

An initial guess of $(1+j)$ where $j = \sqrt{-1}$ is used to start the iteration process. The guess is divided into the polynomial by means of synthetic division. If the remainder of this division is zero, the "guess" is a root. Since the calculator can only find a root to limited accuracy, the absolute value of the real and imaginary parts are tested to see if they are less than some small user entered number epsilon (ϵ). If the real or imaginary part is greater than ϵ, the guess is adjusted using the values of the first derivative at the guess. Then the process is performed over again (iterated).

If a guess is found to be a root, the real and imaginary parts are stored and the resulting polynomial of reduced order is used to find the next root. This scheme continues until all the roots have been found. There are two drawbacks inherent in this computing method. First, each successive root is found with reduced accuracy since the order of the polynomial is reduced as roots are found. This program lists the roots in the order in which they were found, so that the user can discern their relative accuracy. Second, it is possible for the program to get "hung", and never be able to converge close enough to the root to satisfy the epsilon test. The first problem can be helped by using a smaller ϵ; however, using a smaller ϵ increases the chances of the second problem occurring and also increases run time.

This program can be used with or without the PC-100A/C Print Cradle. If the printer is used prompted input and labeled output are produced. The **MASTER LIBRARY** module is required to run this program.

RECORDING PROCEDURE

1. Key in the Input Section and record it (bank 1) on one side of magnetic card (use standard partitioning).
2. Press CP and 9 2nd Op 17 to partition the TI-59 to 239.89.
3. Key in the Working Section and record it (bank 1) on the other side of the magnetic card.
4. Press CP.
5. Key in the Output Section and record it (bank 1) on a second magnetic card.

USER INSTRUCTIONS

1. Read in the Input Section of the program with the calculator in standard partition.
2. Enter the order (N) of the polynomial and press A.
3. Enter ϵ and press R/S.
4. Enter the coefficients as follows:

Enter	Press
Real part of the coefficient of X^N	R/S
Imaginary part of the coefficient of X^N	R/S
Real part of the coefficient of X^{N-1}	R/S
Imaginary part of the coefficient of X^{N-1}	R/S
•	
•	
•	
Real part of the coefficient of X^0	R/S
Imaginary part of the coefficient of X^0	R/S

At this point there will be a "1" in the display.

5. If an error was made in the entry of a particular coefficient, enter the exponent of X for that coefficient and press E. Then enter the real part of the coefficient, press R/S, enter the imaginary part, and press R/S.
6. Read in the Working Section of the program and press R/S. This part will run for several minutes and stop with a "1" in the display.
7. Read in the Output Section of the program.
8. If the printer is attached, press R/S. If additional copies of the roots are desired press A.
9. If the printer is not used, press R/S to display the real part of the first root. Press R/S to display the imaginary part of the first root. Continued pressing of R/S will continue to display the real and imaginary part of each root. If you press R/S after the imaginary part of the last root has been displayed, the display will show "479.59" to indicate that all the roots have been displayed. Pressing R/S again will start the output sequence over.

SAMPLE PROBLEM

Solve the following cubic equation for all three roots:

$$4X^3 + (8+8j)X^2 + (-7+8j)X + (-6-3j) = 0.$$

The actual roots for the equation are:

$$\frac{\sqrt{2}}{2} - \frac{1}{2}j,\ -\frac{\sqrt{2}}{2} - \frac{1}{2}j,\ -2 - j.$$

Enter	Press	Display	Comment
Read in the Input Section.			
3	A	0	Enter order.
.00000001	R/S	0	Enter ϵ.
4	R/S	0	Enter the
0	R/S	0	coefficients.
8	R/S	0	
8	R/S	0	
7 +/−	R/S	0	
8	R/S	0	
6 +/−	R/S	0	
3 +/−	R/S	1	
Read in the Working Section.			
	R/S	1	Program runs for about 7 minutes.
Read in Output Section. If the printer is attached just press R/S. The printer output is shown below. Otherwise,			
	R/S	.7071067812	
	R/S	-0.5	
	R/S	-.7071067812	
	R/S	-0.5	
	R/S	-2.	
	R/S	-1.	
	R/S	479.59	All roots have been displayed.

PRINTER OUTPUT

```
    NEWTON RAPHSON

                          EP

0.00000001

   COEF. OF X↑  03
          4.          +
          0.          J

   COEF. OF X↑  02
          8.          +
          8.          J

   COEF. OF X↑  01
         -7.          +
          8.          J

   COEF. OF X↑  00
         -6.          +
         -3.          J

    ROOTS

.7071067812           +
        -0.5          J

-.7071067812          +
        -0.5          J

         -2.          +
         -1.          J
```

INPUT SECTION LISTING

000	76	LBL	050	02	2	100	69	DP	150	00	0
001	11	A	051	03	3	101	02	02	151	54	)
002	42	STO	052	03	3	102	03	3	152	22	INV
003	03	03	053	06	6	103	02	2	153	59	INT
004	75	-	054	69	DP	104	02	2	154	65	×
005	01	1	055	03	03	105	01	1	155	01	1
006	95	=	056	03	3	106	00	0	156	00	0
007	42	STO	057	02	2	107	00	0	157	54	)
008	00	00	058	03	3	108	04	4	158	71	SBR
009	85	+	059	01	1	109	04	4	159	01	01
010	02	2	060	00	0	110	06	6	160	97	97
011	95	=	061	00	0	111	00	0	161	65	×
012	42	STO	062	00	0	112	69	DP	162	01	1
013	01	01	063	00	0	113	03	03	163	00	0
014	09	9	064	00	0	114	01	1	164	00	0
015	69	DP	065	00	0	115	04	4	165	95	=
016	17	17	066	69	DP	116	42	STO	166	69	DP
017	00	0	067	04	04	117	05	05	167	04	04
018	42	STO	068	69	DP	118	71	SBR	168	69	DP
019	08	08	069	05	05	119	01	01	169	05	05
020	05	5	070	98	ADV	120	28	28	170	69	DP
021	02	2	071	69	DP	121	97	DSZ	171	33	33
022	42	STO	072	00	00	122	01	01	172	04	4
023	06	06	073	01	1	123	01	01	173	07	7
024	69	DP	074	07	7	124	18	18	174	69	DP
025	00	00	075	03	3	125	61	GTO	175	04	04
026	03	3	076	03	3	126	02	02	176	25	CLR
027	01	1	077	69	DP	127	32	32	177	91	R/S
028	01	1	078	04	04	128	98	ADV	178	72	ST*
029	07	7	079	69	DP	129	43	RCL	179	05	05
030	69	DP	080	05	05	130	03	03	180	69	DP
031	01	01	081	25	CLR	131	55	÷	181	06	06
032	04	4	082	91	R/S	132	01	1	182	69	DP
033	03	3	083	99	PRT	133	00	0	183	25	25
034	03	3	084	42	STO	134	95	=	184	02	2
035	07	7	085	11	11	135	59	INT	185	05	5
036	03	3	086	01	1	136	71	SBR	186	69	DP
037	02	2	087	05	5	137	01	01	187	04	04
038	03	3	088	69	DP	138	97	97	188	25	CLR
039	01	1	089	01	01	139	65	×	189	91	R/S
040	00	0	090	03	3	140	01	1	190	72	ST*
041	00	0	091	02	2	141	00	0	191	05	05
042	69	DP	092	01	1	142	00	0	192	69	DP
043	02	02	093	07	7	143	85	+	193	06	06
044	03	3	094	02	2	144	53	(	194	69	DP
045	05	5	095	01	1	145	53	(	195	25	25
046	01	1	096	04	4	146	43	RCL	196	92	RTN
047	03	3	097	00	0	147	03	03	197	32	X:T
048	03	3	098	00	0	148	55	÷	198	09	9
049	03	3	099	00	0	149	01	1	199	32	X:T

Listing Continued Above

200	22	INV	209	92	RTN	218	85	+	227	42	STO
201	67	EQ	210	76	LBL	219	01	1	228	05	05
202	02	02	211	15	E	220	95	=	229	71	SBR
203	06	06	212	42	STO	221	65	×	230	01	01
204	85	+	213	03	03	222	02	2	231	28	28
205	02	2	214	94	+/-	223	85	+	232	01	1
206	85	+	215	85	+	224	01	1	233	91	R/S
207	01	1	216	43	RCL	225	04	4	234	81	RST
208	95	=	217	00	00	226	95	=			

WORKING SECTION LISTING

000	69	DP	060	69	DP	120	03	03	180	22	INV
001	28	28	061	27	27	121	69	DP	181	74	SM*
002	69	DP	062	36	PGM	122	26	26	182	06	06
003	36	36	063	04	04	123	73	RC*	183	69	DP
004	01	1	064	71	SBR	124	06	06	184	26	26
005	72	ST*	065	00	00	125	42	STO	185	43	RCL
006	06	06	066	64	64	126	04	04	186	02	02
007	69	DP	067	73	RC*	127	69	DP	187	22	INV
008	36	36	068	05	05	128	36	36	188	74	SM*
009	72	ST*	069	44	SUM	129	36	PGM	189	06	06
010	06	06	070	01	01	130	04	04	190	69	DP
011	00	0	071	69	DP	131	71	SBR	191	36	36
012	42	STO	072	25	25	132	00	00	192	61	GTO
013	52	52	073	73	RC*	133	64	64	193	00	00
014	42	STO	074	05	05	134	73	RC*	194	11	11
015	53	53	075	44	SUM	135	07	07	195	43	RCL
016	43	RCL	076	02	02	136	44	SUM	196	00	00
017	00	00	077	69	DP	137	01	01	197	85	+
018	85	+	078	25	25	138	69	DP	198	02	2
019	03	3	079	43	RCL	139	27	27	199	75	-
020	75	-	080	01	01	140	73	RC*	200	43	RCL
021	43	RCL	081	72	ST*	141	07	07	201	08	08
022	08	08	082	07	07	142	44	SUM	202	95	=
023	95	=	083	69	DP	143	02	02	203	65	×
024	42	STO	084	27	27	144	69	DP	204	02	2
025	10	10	085	43	RCL	145	27	27	205	95	=
026	42	STO	086	02	02	146	97	DSZ	206	42	STO
027	09	09	087	72	ST*	147	09	09	207	09	09
028	01	1	088	07	07	148	01	01	208	01	1
029	04	4	089	97	DSZ	149	17	17	209	04	4
030	42	STO	090	09	09	150	36	PGM	210	42	STO
031	05	05	091	00	00	151	04	04	211	05	05
032	05	5	092	36	36	152	10	E'	212	05	5
033	03	3	093	42	STO	153	43	RCL	213	04	4
034	42	STO	094	13	13	154	12	12	214	42	STO
035	07	07	095	69	DP	155	42	STO	215	07	07
036	69	DP	096	37	37	156	01	01	216	73	RC*
037	37	37	097	73	RC*	157	43	RCL	217	07	07
038	73	RC*	098	07	07	158	13	13	218	72	ST*
039	06	06	099	42	STO	159	42	STO	219	05	05
040	42	STO	100	12	12	160	02	02	220	69	DP
041	01	01	101	00	0	161	36	PGM	221	27	27
042	69	DP	102	42	STO	162	04	04	222	69	DP
043	26	26	103	01	01	163	18	C'	223	25	25
044	73	RC*	104	42	STO	164	43	RCL	224	97	DSZ
045	06	06	105	02	02	165	11	11	225	09	09
046	42	STO	106	43	RCL	166	32	X:T	226	02	02
047	02	02	107	10	10	167	50	I×I	227	16	16
048	69	DP	108	75	-	168	77	GE	228	43	RCL
049	36	36	109	01	1	169	01	01	229	00	00
050	73	RC*	110	95	=	170	78	78	230	32	X:T
051	07	07	111	42	STO	171	43	RCL	231	43	RCL
052	42	STO	112	09	09	172	01	01	232	08	08
053	03	03	113	05	5	173	50	I×I	233	22	INV
054	69	DP	114	04	4	174	22	INV	234	67	EQ
055	27	27	115	42	STO	175	77	GE	235	00	00
056	73	RC*	116	07	07	176	01	01	236	00	00
057	07	07	117	73	RC*	177	95	95	237	01	1
058	42	STO	118	06	06	178	43	RCL	238	91	R/S
059	04	04	119	42	STO	179	01	01	239	81	RST

OUTPUT SECTION LISTING

000	43	RCL	011	14	14	022	36	36	033	69	DP
001	16	16	012	42	STO	023	43	RCL	034	00	00
002	94	+/-	013	03	03	024	02	02	035	98	ADV
003	42	STO	014	43	RCL	025	72	ST*	036	03	3
004	01	01	015	15	15	026	06	06	037	05	5
005	43	RCL	016	42	STO	027	69	DP	038	03	3
006	17	17	017	04	04	028	36	36	039	02	2
007	94	+/-	018	36	PGM	029	43	RCL	040	03	3
008	42	STO	019	04	04	030	01	01	041	02	2
009	02	02	020	18	C'	031	72	ST*	042	03	3
010	43	RCL	021	69	DP	032	06	06	043	07	7

Listing Continued on Page 8

044	03	3	062	01	1	080	06	06	098	02	2
045	06	6	063	95	=	081	32	X:T	099	05	5
046	69	DP	064	42	STO	082	69	DP	100	69	DP
047	02	02	065	09	09	083	36	36	101	04	04
048	69	DP	066	05	5	084	73	RC*	102	32	X:T
049	05	05	067	02	2	085	06	06	103	69	DP
050	02	2	068	42	STO	086	87	IFF	104	06	06
051	00	0	069	06	06	087	07	07	105	97	DSZ
052	69	DP	070	69	DP	088	00	00	106	09	09
053	07	07	071	00	00	089	96	96	107	00	00
054	69	DP	072	98	ADV	090	91	R/S	108	72	72
055	19	19	073	04	4	091	32	X:T	109	06	6
056	24	CE	074	07	7	092	91	R/S	110	69	DP
057	76	LBL	075	69	DP	093	61	GTO	111	17	17
058	11	A	076	04	04	094	01	01	112	91	R/S
059	43	RCL	077	69	DP	095	05	05	113	61	GTO
060	00	00	078	36	36	096	69	DP	114	00	00
061	85	+	079	73	RC*	097	06	06	115	59	59

Inverse Days *(continued)*

days between two dates was simply the difference in their factors.

At this point I was ready to **FORMULATE A GENERAL METHODOLOGY.** The basic scheme I came up with follows:

1. Calculate the factor of the entered date.
2. Add the number of days to the target date to this factor to get the target date factor.
3. Approximate the target date.
4. Calculate the factor for the approximate target date.
5. Compare the factor calculated in step 4 to the factor calculated in step 2. If they are identical the target date has been found.
6. If the factors did not match in step 5, the target date would be reapproximated and we would go back to step 4.

The general methodology serves to break the whole problem into several sub-problems which are, hopefully, easier to accomplish. As one attacks each of these sub-problems, often a new perspective is gained on the whole problem and the general methodology is revised or completely scrapped for a better methodology. In actuality, the method shown above was my second try. Originally I had tried to solve for the date explicitly from the formulas on page 76 of the Master Library manual. I found this task to be extremely laborious, so I opted for the scheme outlined above.

The format of the general methodology varies from programmer to programmer. A flow chart is one tool that can be used. It provides a clear and orderly diagram of the process. Step-by-step instructions, such as I have used, can also be employed. This technique is basically a flow chart that has been written out instead of diagramed. Other programmers can do this step in their heads.

My next step was to **DETERMINE THE METHOD OF SOLUTION FOR EACH SUB-PROBLEM.** Sub-problem one was easily performed using ML-20 as a subroutine. Step two was also done with ease. Figuring out how to do step three was more difficult. I found it necessary to approximate the date in two stages and again revise my methodology accordingly. Starting with step three my new process looked like this:

3. Calculate the approximate year in which the target date falls so that the approximation is either in the actual year or the year after. This can be accomplished by taking the integer part of the quotient of the target date factor divided by 365.24. (The factor "365.24" was sort of a compromise factor. Every year has 365 days except for leap years which have 366. In order to make up for the inperfect adjustment of the leap year system, the leap day in the year at the end of a century (years ending in 00) is omitted, except for years evenly divisible by 400 in which the leap year remains. Experimentation showed that 365.24 is slightly less than the average number of days per year.)
4. Generate the factor for January 1 of the approximate target year.
5. If the factor generated in step 4 is greater than the factor generated in step 2, decrease the approximate year by 1 and calculate a new approximate target date factor.
6. The correct target year is now known. To arrive at an estimate for the target month that is either correct or 1 month past, take the difference of the target factor and the approximate target date factor, divide it by 29, add 1, and take the integer portion of the result. (The division by 29 will yield the approximate decimal number of months. A factor any larger, say 30, will create problems on March 1. The "1" is added so that when the integer function is applied the resulting number is the month number or the month number plus one.)
7. Generate the factor for the first day of the approximate target month.
8. If the factor generated in step 7 is greater than the target date factor, decrease the approximate month by 1 and generate a new approximate target date factor.
9. The correct target month has been found. To find the target day, subtract the factor that was last calculated (either step 7 or 8) from the target factor and add 1.
10. Display the answer.

Now that the method of solution was well defined, it was time to **ASSIGN DATA REGISTERS AND CODE THE PROGRAM.** My program is shown below.

000	76	LBL	034	00	00	068	43	RCL
001	11	A	035	41	41	069	01	01
002	36	PGM	036	69	DP	070	65	×
003	20	20	037	39	39	071	01	1
004	11	A	038	71	SBR	072	00	0
005	91	R/S	039	00	00	073	00	0
006	76	LBL	040	85	85	074	85	+
007	13	C	041	32	X:T	075	43	RCL
008	85	+	042	50	I×I	076	09	09
009	43	RCL	043	55	÷	077	55	÷
010	04	04	044	02	2	078	04	4
011	95	=	045	09	9	079	22	INV
012	42	STO	046	85	+	080	28	LOG
013	05	05	047	01	1	081	95	=
014	55	÷	048	95	=	082	58	FIX
015	03	3	049	59	INT	083	04	04
016	06	6	050	42	STO	084	91	R/S
017	05	5	051	01	01	085	43	RCL
018	93	.	052	71	SBR	086	09	09
019	02	2	053	00	00	087	42	STO
020	04	4	054	85	85	088	03	03
021	95	=	055	77	GE	089	36	PGM
022	59	INT	056	00	00	090	20	20
023	42	STO	057	63	63	091	71	SBR
024	09	09	058	69	DP	092	00	00
025	01	1	059	31	31	093	86	86
026	42	STO	060	71	SBR	094	75	-
027	01	01	061	00	00	095	43	RCL
028	42	STO	062	85	85	096	05	05
029	02	02	063	32	X:T	097	95	=
030	71	SBR	064	50	I×I	098	32	X:T
031	00	00	065	85	+	099	00	0
032	85	85	066	01	1	100	92	RTN
033	77	GE	067	85	+			

To run the program, enter the date in the MMDD.YYYY format of ML-20 and press A. Enter the number of days to the target date (use a negative number for a target date before the entered date) and press C. The target date will be displayed. (Note: The Master Library module is required.)

Once the program was keyed in, the next step was to **TEST THE ALGORITHM** using "worst case" conditions. In this situation, if any problems were to occur it would probably be when the target date landed on the first or last of a month or year or around leap days. These cases were tested and the program was found to perform satisfactorily.

The final step in this program development was to **ANALYZE THE LOGIC AND CODE FOR OPTIMIZATION.** In this case, since I had plenty of program space, ease of use and execution speed were my prime objectives. I have two suggestions on how to accomplish this phase. The first is to put the program away and forget about it for a couple of days, then reapproach the problem. Often gross inefficiencies can be detected this way. My second suggestion is to get a second opinion on the program. Another person provides a fresh vantage point from which to view the problem and can sometimes offer suggestions for improvement. I did both these things. In correspondence with PPX member Bill Beebe, he helped me derive the factor of 365.24 (steps 015-020) instead of a previous factor of 365. This change shortened the run time.

If you have been hesitant to take on a programming task because you did not know where to start, application of these seven principles should help you. But remember, no worthwhile piece of software comes easily; it takes time and brainstorming.

Letters to the Editor

Do you have comments, compliments or (shudder) complaints about PPX? We have always welcomed letters from our membership, and therefore, we are providing space in each newsletter to share your views on PPX with your fellow members. Approximately 2-4 letters dealing with issues of general interest will be featured in each issue. Letters will be edited to fit available space.

Dear Editor:

Is there a way to increase data registers above 99 so that there are, say, 112 or 250, or whatever individually accessible registers for more data storage in a program. Perhaps a module could be used.

G.B. Stanton

Dear Mr. Stanton,

Your question is one that I am frequently asked. There is no way to obtain data registers on the TI-59 above the alloted 99. The Solid State Software™ Modules are ROMs (Read Only Memory) on which the codes are indelibly "written" on a silicon chip and as such, the information on a module cannot be changed.

Editor

Dear Editor:

The piece on "Hard-Wired Functions", Mar/Apr 1981, drew my attention, particularly the Angle Conversion routines on p. 9, which I related to a routine I had devised for converting the difference between two dates into years and a decimal fraction (see my PPX #198054). As I looked over the Angle Conversion and Inverse steps, it appeared some steps could be saved by rearranging the sequences.

303	53	(	321	55	÷	339	75	-
304	53	(	322	06	6	340	22	INV
305	82	HIR	323	00	0	341	59	INT
306	08	08	324	85	+	342	65	×
307	22	INV	325	82	HIR	343	93	.
308	59	INT	326	18	18	344	04	4
309	65	×	327	59	INT	345	54	)
310	01	1	328	54	)	346	65	×
311	00	0	329	92	RTN	347	93	.
312	00	0	330	53	(	348	00	0
313	85	+	331	53	(	349	01	1
314	22	INV	332	82	HIR	350	85	+
315	59	INT	333	08	08	351	82	HIR
316	55	÷	334	22	INV	352	18	18
317	01	1	335	59	INT	353	59	INT
318	93	.	336	65	×	354	54	)
319	05	5	337	06	6	355	92	RTN
320	54	)	338	00	0			

By use of a little gimmickry, namely an intermediate special divisor 1.5 and .4 for the respective sequences, I was able to reduce the number of steps from 77 to 53 steps. I realize of course that my shortened steps are merely an academic exercise, since the actual routines are "carved in stone" or maybe "etched in silicon" in the innards of the TI-58/59, not to be changed while these machines are in production. Nevertheless, I feel it is instructive to show the reduction in steps, as a short study in programming technique. What follows is an explanation of the high-level cerebration by which I laid hold on the idea of the intermediate factors of 1/1.5 and .4.

I was contemplating the second half, where in my example 47.2202 degrees is converted back to 47.131272 deg.min.sec. Here the decimal fraction .2202 is multiplied by 60, giving 13.212, which must be brought to 13.1272, thence to .131272, to be added to 47 for the answer.

I was "doodling" figures on the paper, and subtracted 13.1272 from 13.212, which gave .0848, when FLASH! I saw that .0848 was .4 times .212. Just like that! But if the simple relationship hadn't been almost self-evident, I doubt that it would otherwise have had the wit to devise my final routine. After seeing that .0848 = .4x.212, I turned to the first half, where that same difference of .0848 had to be related to .1272, a relationship which at that point wasn't too hard to determine must be 1/1.5.

All this with not a thought as to any formal statement of an algebraic equation to express the algorithm. Later, by hind-

sight, I derived the following formulas:

$$D.d = DD + \frac{MM}{60} + \frac{SS}{3600}$$

$$= DD + \frac{MM}{60} + \frac{SS \times 2.5}{9000}$$

$$= DD + \frac{MM}{60} + \frac{\frac{SS}{100}\left[1 + \frac{1}{1.5}\right]}{60};$$

$$D.d = DD + \frac{MM.SS + \frac{.SS}{1.5}}{60}.$$

And for the reverse process:

$$DD.MMSS = D + \frac{.d \times 60 - .m \times .4}{100},$$

where .d = decimal degree, and .m = decimal minute generated by .d x 60.

Sincerely,
Ralph W. Snyder

Dear Mr. Snyder,

Your intermediate divisor idea is a good one. I hope other members will be able to use it. Perhaps we should all "doodle" more often.

Editor

from the Analyst's Desk

• PPX member Clyde Durbin has discovered a new way of calling subroutines from the keyboard. To illustrate this method, consider the partial listing shown below.

```
000  91 R/S
001  22 INV
002  01  1
003  43 RCL
004  01  01
005  99 PRT
006  02  2
007  03  3
008  42 STO
```

If these steps are in the program memory of the calculator then pressing

SBR 1 SBR will call the subroutine at	Lbl INV
SBR 2 SBR	Step 143
SBR 3 SBR	Lbl RCL
SBR 4 SBR	Step 199
SBR 5 SBR	Lbl Prt
SBR 6 SBR	Step 203
SBR 7 SBR	Step 342

The explanation of this phenomenon is that the calculator interprets the SBR N SBR (where N is a integer between 0 and 99 inclusive) as a special kind of subroutine call. N is limited to 99 since pressing a third digit would constitute a normal subroutine call. The SBR N SBR sequence causes the TI-59 to perform the last SBR command beginning execution at step N which is interpreted as the address for the subroutine call. If program step N contains a code between 10 and 99 (except codes 21, 26, 31, 41, 46, 51, or 56) the calculator will perform a label search for the code at N. If the label is found, program execution continues at the label location and, if the subroutine is terminated by a RTN, execution will be returned to step N + 1 when the subroutine is completed. As with the ordinary subroutine call, if the label is not found the program aborts and the display flashes. If step N is a code from 0 to 9, steps N and N + 1 will be interpreted as an absolute address and the program will branch accordingly if the address is within the current partitioning of the calculator.

One possible use of this technique could be in calling up different routines that are all stored on the same magnetic card. Admittedly, the call requires three keystrokes, but numerical tags for routines could come in quite handy. If, for instance, you have a magnetic card with more than ten user accessed routines this method of routine calling might be easier to use than having to resort to subroutine calling with common labels. This method also allows for 100 different subroutine calls, but, of course, requires as many program steps as the number of different calls required. Numerical data can be entered before this type of subroutine call and it will remain in the display register during the transfer. Routines called by this method should probably be terminated with a RST command (hence the R/S at step 000 of the above listing) to clear the subroutine return register of the return address (location N + 1 or N + 2 depending upon whether the transfer was with a label or done with absolute addressing) that was placed there by the subroutine call.

• In last issue's Analyst Desk, a PPX member pointed out that the INV log function does not produce "exact" answers. PPX member Charlie Wiliamson points out that the sequence EE INV EE will truncate the guard digits so that what is shown in the display is all that is in the display register.

Précis

This column presents some of the new PPX programs which have been recently accepted. The abstracts here are from programs that the Analysts thought would be of special interest to members. You can purchase these programs at a cost of $4.00 each. Send your order to: Texas Instruments: C/O PPX Department: P.O. Box 109, Lubbock, Texas 79408. Include an additional $2.00 to cover postage and handling.

If you have a need for a specific program, send a note to PPX. There is a chance that the program may have already been written. If it has, we will put the abstract in the next

issue of the Exchange. Requests for programs not yet written will be placed in the "Programming Corner" column.

158002H Payback Time With Inflation
An earnings will increase over a period to time due to inflation yet it decreases due to the interest rate. Given this data, this program calculates the breakeven or payback period of an investment due to inflation and interest.
George F. Polan, Providence RI
181 Steps

188042H Searching for Call Option Spreads
Program provides a data entry routine for entering a series of option prices, after which it looks at 21 buy and sell combinations to determine if the profit/investment ratio meets a prescribed threshhold. If so, program prints maximum profit and investment for each combination with brokers commissions considered. The program treats a price spread where you are bullish on the stock.
Robert S. McGihon, Alexandria, VA
455 Steps, PC-100A

188043H Portfolio Values and Breakdown
On a given date when prices or value of all components of the portfolio are available, this program calculates the value of each component, the value of the portfolio, and the percentage of the portfolio in each of 4 categories.
John E. Binns, Stuart, FL
406 Steps

228066H Mantel-Haenszel Test
Based on the hypergeometric distribution, this one-degree-of-freedom procedure tests the null hypothesis that the rate of occurance of an observed event is the same for two groups after adjusting for some characteristic which is distributed in both groups.
Jim Gibbons, CSW, Great Neck, NY
170 Steps, PC100-A, Mod 2

398238H Division
Divides 25 digits by 12 digits to return 30 digits. Program is intended for practical calculations rather than a demonstration of capability. Run times less than 90 seconds.
Laurance M. Leeds, Sun City, AR
198 Steps

398239H Triple Precision Arithmetic
Given two 36-digit floating-point numbers (A and B), calculates: A+B, A−B, AxB, A/B. After an operation, the result becomes the new A, and a new B can be entered for chain calculations, A and B can be exchanged. Results are rounded to 32 digits for display. For simple operations, round-off error is less than 1 unit of the 34th digit.
Jacques Gelinas, Saint-Jean, Quebec, CA
376 Steps

418120H Kirchoff's Law of Enthalpy
This program finds the enthalpy change of a particular material (gas, liquid, or solid) from its heat capacity, initial and final temperature, and the reactions's standard room temperature constant. It uses Kirchoff's law in finding the enthalpy change. There are also a few energy and temperature conversion subroutines that can help with the data inputs. The standard room temperature constant of the reaction, also called the heat of reaction, can be found by Hess' Law and is needed as an input for this program
Gregory C. Franz, St. Louis Park, MN
217 Steps

418121H Chemical Equation Balancer
Given the number of elements (or radicals which are not broken up) the number of compounds involved, and the subscripts of the elements, this program will compute the coefficients for each compound. Ths maximum number of elements and compounds is 8 and 9 respectively. Subscripts may be as high as 99. Printer optional.
Kirt Gibble, Manheim, PA
356 Steps, Mod 1

468017H Directional Drilling Log Calculations
Calculates locations and depth data from a "Directional Well Survey" Record for regions of interest between survey points. Calculates the "Polar Coordinates" of the depth point from the well site for better directional control. Results are labeled and printed when used with PC-100C Printer.
Robert W. Pice, Gainesville, FL
361 Steps

578006H Hard Contact Lens Specifications
From measurements of corneal power and astigmatism (from Keratometer readings, K1;K2), pupil and corneal diameter, lid tension, the patients spectacle prescription and the vertex distance at which the R_x was obtained, this program will compute the parameters necessary for the optician/optometrist/opthalmologist to order the initial set of hard, tricurve contact lenses.
Sylvan J. Hotch, Wayland, MA
442 Steps

628199H Deflections Due To Distributed Loadings
This program accurately and quickly computes the deflection of a simply supported beam due to a uniformly distributed load. Fully prompted I/O.
Jay Lamb, Midland, TX
700 Steps, PC100A

628200H Force Distribution (with Printer)
Given a maximum of 20 elements (Areas or Rigidities) in X and, or Y direction, will compute the location of center of rigidity of these elements as well as the location of and magnitude of external forces applied on these elements and determine the force distribution to each element and location of each element and force from the center of rigidity.
Dan S. Hirschfeld, Niles, IL
896 Steps, PC-100A, Mod 1

708006H Resonant Frequency
This program provides the resonant frequencies for any dimension room. Lowest and highest possible frequencies, based on program design and room dimensions, are available.
Ross D. Litman, II, Ft. Wainwright, AK
115 Steps, Mod 1

PROGRAMMING CORNER

This column is potentially a very powerful tool for you in getting custom written programs from some of the more advanced PPX programmers. In order to take advantage of this valuable opportunity, simply send us a description of your programming needs along with your address.

Our author incentive program still applies to those who write programs to fill these needs. When submitting such programs please include a note stating that the program was written to fill a Programming Corner request. Programs submitted to fill the following requests should be postmarked before November 1, 1981 in order to be considered for the author incentive program.

- A program that takes a number (integer plus decimal) and packs it into a required set of registers to any required precision (with space limitation). It then allows the operators x, ÷, +, and − to be performed on the entire content as a unit with the result stored in a different set of registers.
- A program that will store up to 100 stock (inventory) code numbers with their corresponding quantity. Then it should be able to search and find any requested stock code number and allow the user to change or simply recall the quantity of that item within a reasonable amount of time.
- A program to show the effects of taxes and inflation on an investment while being able to alter the investment amount, tax bracket, inflation assumption, and yield assumption.

Membership Renewals

Is your membership about to expire? To ensure that you will miss no newsletters, catalogs, or ordering privileges, check the renewal table to find out if your membership will soon expire. (If your number is not included in the range of the table, it is not time for you to renew). The next issues of the Exchange will list additional renewal dates.

A renewal card and reminder will be sent to each member before the time to renew. Return the card to PPX with your check or money order for $20.00. Be sure to include your membership number on both your card and your check and mail to: Texas Instruments PPX Department, P.O. Box 109, Lubbock, TX 79408.

MEMBERSHIP NUMBER	RENEWAL MONTH
909340-910093	September
920097-920478	September
927573-928026	September
910094-910895	October
920479-921594	October
928027-928279	October

TI-59 Programming Seminar

A Texas Instruments Programming Seminar may be coming to your area. These seminars will provide beginning and intermediate programming training on the TI-59. Tuition for the two day class is $150.00 per person. This includes the instruction, workbook and luncheon for the two days. You should supply your own TI-59. To register send your check for $150.00 payable to Texas Instruments to:

TI-59 Seminar
Texas Instruments
P.O. Box 10508 MS 5820
Lubbock, Texas 79408

If you have further questions regarding the seminar call Sherry Schroeder at 806-741-3277. The schedule for 1981 is:

SEMINAR DATES		LOCATION
August	13-14	Minneapolis, MN
August	27-28	San Francisco, CA
September	10-11	Houston, TX
September	24-25	Denver, CO
October	8-9	Washington, DC
October	22-23	Detroit, MI
November	5-6	Cincinnati, OH

TEXAS INSTRUMENTS
INCORPORATED
PPX • P.O. Box 53 • Lubbock, Texas 79408
U.S. CALCULATOR PRODUCTS DIVISION

ADDRESS CORRECTION REQUESTED

BULK RATE
U.S. POSTAGE
PAID
Permit No. 1
Lubbock, Texas