PPX EXCHANGE

Vol. 4 Number 1 Copyright 1980 January/February 1980

TI-59 GOES TO LAKE PLACID

The TI-59 programmable calculator is making an Olympic debut at Lake Placid this winter. Using a Solid State Software™ module developed for the Olympics by Texas Instruments, the TI-59 will aid in verifying the results computed by the TI-SCORE™ Computer System (System for Computerized Olympics Results and Events). The four major events include: bobsled competition, cross-country skiing, speed skating, and the biathalon. The Olympic module is programmed to handle all aspects of scoring for each event; thus permitting immediate handheld verification. Due to the limited quantity manufactured, this module is not available to the public.

PPX POTPOURRI

1. **Order Forms —** Please use PPX order forms when ordering TI-59 programs and acessories from PPX. These forms were designed for two purposes: First, for our member's convenience. Second, they are used by PPX to keep a record of the date a member's order was received, filled and shipped. If you do not have access to an order form, please write your member number and the program/accessory number and name on a sheet of 8½ x 11" paper. Due to the quantity of orders received, **we can no longer document order forms for those members who write their orders on checks and money orders.** For this reason, such orders will be returned (unfilled) to the requestor.

2. **Ordering Hint —** The number of PPX filled orders that are being lost within members' companies are on the increase. This is primarily because the self addressed mailing label contained only the name of the company to which it was addressed. For this reason, we ask that all corporate members (memberships which are under a company's name) have their orders shipped to the attention of a particular person within the company. This can be done by simply writing the person's name on the order form label.

PPX-59 PROGRAMMING CORNER

This column is devoted to PPX-59 programming suggestions. If you have a program(s) that you would like to see made available through PPX-59, send your suggestions to PPX. In this way, members who enjoy programming are made aware of your programming needs. PPX-59 is not staffed to do custom programming; therefore, member suggested programs will become available only if another member of PPX-59 comes to the rescue.

Our members would like to see:

- The following relative motion programs for navigation:
  1. Course, time and distance to intercept another moving vessel.
  2. Time and distance of closest point of approach (CPA).
  3. Other ships course and speed from two bearings and distance.
  4. Time and distance from storm (CPA) and course to avoid storm.
- Programs for analysis and design of carbon dioxide and Halon fire extinguishing systems.
- A program to alphabetize alpha data.
- A chemical equation balancing program.

MORE SUBROUTINE LEVELS FOR THE TI-59

Barry S. Tepperman

Editor's Note: Dr. Tepperman is an active member of PPX as can be seen by the number of excellent programs he has in the catalog. While programming on the TI-59, he found it necessary to use recursive programming routines (that is, a routine that repeats itself until a certain preset condition is met). Upon finding that the TI-59's six levels of nested subroutines were too restrictive, he came up with a solution which uses the memory registers as a push-down stack. Recognizing that this solution could help others, he wished to share it with other PPXers. The following article describes this method and also supplies an example of how it is used.

When a SBR command is encountered under program control, the flow of processing is immediately diverted to the subroutine called. The location number following the subroutine call is stored in the subroutine return register. The INV SBR command terminates each subroutine and processing transfers back to the location stored in the subroutine return register. Up to six return locations can be stored in the subroutine return register, allowing the nesting of up to six subroutines. (See pages IV-46 and 47 of Personal Programming for a more indepth explanation.)

By simulating the functions of SBR and INV SBR with GTO and GTO IND, respectively, the limitation of only six subroutine levels can be overcome. To do so, the function of the subroutine return register must also be duplicated. The implementation of a push-down stack in the data memory registers can be used to accomplish the functions of the subroutine return register.

A push-down stack consists of a block of memory registers that are accessed indirectly through a pointer address — a memory register containing the register number of the next empty register in the stack. Absolute return addresses are stored in the stack on a last-in/first-out basis. Each return address is stored in whichever memory register happens to be at the top of the stack at that moment (the current value of the stack pointer). The pointer is then incremented by one so it will contain the address of the next empty register. Recalling a return address requires decrementing the pointer by one and indirectly recalling the contents of the register indicated by the pointer.

When a subroutine is to be called, the return address is stored on the stack using STO IND, and the pointer is incremented by one. Now, instead of using a SBR command, GTO xxx (where xxx is the absolute address of the subroutine) is used. At the end of the subroutine the return

Subroutine Levels (Cont.)

address is recalled by the method outlined above and then stored in a preassigned register which we will call the return register. By using GTO IND XX (where XX is the address of the return register) in place of INV SBR, program execution will be transferred to the correct location (the location following the GTO command).

It should also be pointed out that the stack need not only be used to store return addresses. Intermediate results can also be stored in the stack. If the stack is to be used in this way it is important that the stack contain enough registers to store these results. In addition, care must be taken that the segment of the program following the return location removes all the intermediate results that are above the next return location in the stack and uses them in reverse order in which they were put on the stack. Failure to do this could cause the program to incorrectly interpret the data as an address.

Let's take Ackermann's function as an example to demonstrate the above technique.

The function, A (m,n) is defined by two non-negative integers, m and n, such that:

A(0,n) = n+1
A(m,0) = A((m-1), 1)
A(m,n) = A((m-1), A(m,(n-1))) for m,n≠0

By using the simple case, A(1,1), we can see how Ackerman's function is computed:

A(1,1) = A((1-1), A(1, (1-1)))
= A(0, A(1,0))
= A(0, A((1-1), 1))
= A(0, A(0,1))
= A(0, (1+1))
= A(0,2)
= 2+1
= 3

Fortunately, the simple example didn't require recursive programming; however, if we choose a less trivial case, such as A(3,2), we would have to use the definition of Ackermann's Function recursively. Try computing A(3,2) by hand. (Warning: this computation may take up to one hour). Before A(3,2) is solved it expands to:

A(3,2) = A(2, A(2, A(0, A(0, A(0, A(0,1))))))

Below is a program listing for computing Ackermann's function using a push-down stack. Note that the current value of m and n are stored in registers 00 and 01, respectively. For this example we have assigned register 02 as the stack pointer and register 03 as the return register. Registers 04 - 99 make up the stack.

000	76	LBL	023	69	OP	046	03	03	069	09	9
001	11	A	024	22	22	047	83	GO*	070	72	ST*
002	42	STO	025	61	GTO	048	03	03	071	02	02
003	00	00	026	00	00	049	43	RCL	072	69	OP
004	99	PRT	027	33	33	050	01	01	073	22	22
005	29	CP	028	43	RCL	051	22	INV	074	69	OP
006	91	R/S	029	01	01	052	67	EQ	075	31	31
007	76	LBL	030	99	PRT	053	00	00	076	61	GTO
008	12	B	031	98	ADV	054	62	62	077	00	00
009	42	STO	032	92	RTN	055	69	OP	078	33	33
010	01	01	033	43	RCL	056	30	30	079	69	OP
011	99	PRT	034	00	00	057	69	OP	080	32	32
012	01	1	035	22	INV	058	21	21	081	73	RC*
013	00	0	036	67	EQ	059	61	GTO	082	02	02
014	69	OP	037	00	00	060	00	00	083	42	STO
015	17	17	038	49	49	061	33	33	084	00	00
016	04	4	039	69	OP	062	43	RCL	085	69	OP
017	42	STO	040	21	21	063	00	00	086	30	30
018	02	02	041	69	OP	064	72	ST*	087	61	GTO
019	02	2	042	32	32	065	02	02	088	00	00
020	08	8	043	73	RC*	066	69	OP	089	33	33
021	72	ST*	044	02	02	067	22	22			
022	02	02	045	42	STO	068	07	7			

To use the program, enter the integers m and n and press labels A and B, respectively. After B is pressed, program execution begins. The computed value of A(m,n) will be displayed. To solve the above example, A(3,2) = 29, the calculator takes about 12 minutes and fills the stack 55 registers deep, which is 27 nested subroutines. If the stack is overflowed ((3,7) will do it), a flashing zero will result.

In our program listing:

- Steps 000-027 contain the initialization and input routines.
- Steps 028 - 032 are the display answer routine.
- Steps 033-048 calculate A(0,n) and indirectly address the stack to find a return location if m = 0 (Note that the initialization routine stored the absolute address (28) of the display answer routine at the bottom of the stack). If m ≠ 0 then the program branches to location 049.
- Steps 049 - 061 calculate A (m,0) if n = 0, otherwise a branch is made to location 062.
- Steps 062 - 078 deal with the case where m, n ≠ 0. The current value of m and the absolute return address (79) are put on the top of the stack and program control sent back to step 033 to begin another recursion.
- Steps 079 - 089 recall the value of m from the stack, decrement it by one, and return to step 033 for yet another recursion.

True, the example of Ackermann's function is a showpiece of recursive programming and may have little real value to you. However, the technique used in simulating a push-down stack in data memory could be of value to you in your own applications.

NEW USES FOUND FOR FIX, LBL, AND THE DECIMAL POINT

Donald R. Lambert

Editor's Note: PPX member Donald Lambert, of Los Angeles, California, has been an avid supporter of PPX through program submissions and inputs to the newsletter. While being a professional programmer and serving clients through his programming service, he has come across various methods that enable him to get a little more out of the TI-59 calculator.

After two years of use, I have found that no matter how much you get out of a calculator, you seem to want more. The following examples represent methods I have used to squeeze a little more out of my TI-59.

- **A commodity trading system** required that after completing calculations, one of the following five messages be printed using a PC-100A/C:

```
OPEN SHORT POSITION
HOLD SHORT POSITION
 OPEN LONG POSITION
 HOLD LONG POSITION
 CLOSE OUT POSITION
```

Normally, this would have required about 50 steps per message, for a total of 250 steps exclusive of the testing required to determine the proper message. But note that the messages were composed of only six parts: HOLD, OPEN, LONG, SHORT, CLOSE OUT and POSITION. Subroutines could be written to print out the common words. But the problem with this is that the positioning of the words OPEN and HOLD change due to the fact that LONG has four letters and SHORT has five letters. The program below uses the FIX 2 keystrokes to make the necessary positioning change. You can see how this works by doing the following: Enter the alphanumeric code for OPEN (1331731) into the display and press OP, 01, OP, 05. Observe the location of the printout of OPEN. Again enter the code for OPEN, but this time press Fix, 2, OP, 01, OP, 05. The Fix 2 adds two 0's which, when stored in OP 01, are recognized as a blank space. To see how this is done in program format, enter the program below and press labels A, B, C, D, and E to print out the five respective commodity trading messages given above.

```
000  76 LBL|029  07   7|058  07   7|087  01  01
001  11   A|030  03   3|059  01   1|088  22 INV
002  22 INV|031  01   1|060  06   6|089  58 FIX
003  76 LBL|032  61 GTO|061  61 GTO|090  03   3
004  12   B|033  00  00|062  00  00|091  03   3
005  86 STF|034  86  86|063  86  86|092  00   0
006  00  00|035  76 LBL|064  76 LBL|093  01   1
007  03   3|036  13   C|065  15   E|094  03   3
008  06   6|037  22 INV|066  01   1|095  06   6
009  02   2|038  76 LBL|067  07   7|096  02   2
010  03   3|039  14   D|068  00   0|097  04   4
011  00   0|040  86 STF|069  00   0|098  69  OP
012  01   1|041  00  00|070  00   0|099  03  03
013  03   3|042  02   2|071  01   1|100  03   3
014  05   5|043  07   7|072  04   4|101  07   7
015  03   3|044  00   0|073  01   1|102  02   2
016  07   7|045  01   1|074  03   3|103  04   4
017  58 FIX|046  03   3|075  07   7|104  00   0
018  02  02|047  01   1|076  69  OP|105  01   1
019  69  OP|048  02   2|077  02  02|106  03   3
020  02  02|049  02   2|078  01   1|107  01   1
021  87 IFF|050  61 GTO|079  05   5|108  00   0
022  00  00|051  00  00|080  02   2|109  00   0
023  00  00|052  19  19|081  07   7|110  69  OP
024  53  53|053  02   2|082  00   0|111  04  04
025  01   1|054  03   3|083  01   1|112  69  OP
026  03   3|055  00   0|084  03   3|113  05  05
027  03   3|056  01   1|085  06   6|114  92 RTN
028  01   1|057  02   2|086  69  OP|
```

• **Lbl can be used as a next-step-Nop.** This is useful whenever there is need to use a keystroke only under certain conditions. An example of this is given by the routine below which uses the keystroke +/- (Step 014) as a Nop when it follows Lbl during program execution. This routine was taken from a moving average program which used register 00 as a pointer for indirectly storing and recalling registers 01 through 50 in descending order.


```
000  76 LBL
001  16  A'
002  42 STO
003  00  00
004  59 INT
005  32 X:T
006  01   1
007  22 INV
008  67  EQ
009  00  00
010  14  14
011  04   4
012  09   9
013  76 LBL
014  94 +/-
015  44 SUM
016  00  00
017  43 RCL
018  00  00
019  92 RTN
```

The program that this routine was taken from only allowed entries greater than or equal to one. Therefore, two cases must be examined.

1) When the number entered is between one and two, it is first integered to equal one, and then placed in the t-register (step 005). Subsequently, a one is placed in the display register and compared to the t-register's contents (this test is performed by INV EQ, where EQ stands for x=t). Since the answer to the INV EQ is "no", the accompanying address is skipped and processing continues at step 011. This results in a 49 being summed into register 00. (The +/- is ignored due to the Lbl preceding it.)

2) When a number greater or equal to two is entered, that number is placed in the t-register. This causes the INV EQ to be answered "yes", and program control is transferred to step 014, which is the +/-. The result is that a negative one is summed into register 00.

The above example is only one of the many possible uses of Lbl as a next-step-Nop. Any key that can be used as a Lbl can be used in a similar manner.

• **The last routine given below,** shows two more uses for the decimal point.

First, at step 013 it is used to clear the display. This use does not effect pending operations, or the error signal as would the CLR or CE keys.

Second, the decimal points between steps 16-48 are used to separate a number of possible outputs. By pressing E', the program below squares integers from 0 to 10, so there are ten possible outputs. Notice that starting at location 19 is the square of 1, at location 22 is the square of 2, at location 25 is the square of 3, etc. The calculator ignores all but the first decimal point it passes and prints/displays the result.

```
000  76 LBL|013  93   .|026  93   .|039  06   6
001  10  E'|014  83 GO*|027  01   1|040  04   4
002  50 IxI|015  00  00|028  06   6|041  93   .
003  59 INT|016  93   .|029  93   .|042  08   8
004  42 STO|017  93   .|030  02   2|043  01   1
005  00  00|018  00   0|031  05   5|044  93   .
006  03   3|019  01   1|032  93   .|045  01   1
007  49 PRD|020  93   .|033  03   3|046  00   0
008  00  00|021  00   0|034  06   6|047  00   0
009  01   1|022  04   4|035  93   .|048  59 INT
010  05   5|023  93   .|036  04   4|049  99 PRT
011  44 SUM|024  00   0|037  09   9|050  92 RTN
012  00  00|025  09   9|038  93   .|
```

MEMBER # ☐☐☐☐☐☐☐☐

NOTICE OF CHANGE OF ADDRESS

In order to ensure uninterrupted service, please submit change of address at least six weeks prior to change. Please mail to:

Texas Instruments, Inc.
PPX Department
P. O. Box 53
Lubbock, TX 79408

NAME: ______________________

OLD ADDRESS: ______________________

____

NEW ADDRESS: ______________________

____

EFFECTIVE DATE: ______________________

PHONE: ______________________

FROM THE ANALYST'S DESK

• For those members who have the program "Linear Programming with Mixed Constraints" (PPX-59 #388004D), the author has informed us that this program does not always give correct solutions. To correct this problem, the following corrections must be made.

1. Listing changes.

OLD			REVISED		
070	50	I×I	070	68	NOP
107	42	STO	107	42	STO
108	08	08	108	03	03
120	42	STO	120	42	STO
121	03	03	121	89	89
123	72	ST*	123	72	ST*
124	03	03	124	89	89
138	72	ST*	138	72	ST*
139	03	03	139	89	89
146	42	STO	146	42	STO
147	03	03	147	89	89
149	72	ST*	149	72	ST*
150	03	03	150	89	89
169	97	DSZ	169	97	DSZ
170	08	08	170	03	03

2. Insert a 0 at location 231 of listing Part II.

• PPX member William H. Beebe, Lilburn, Georgia, uses the following routine when testing the status of flags:

000	76	LBL
001	11	A
002	42	STO
003	01	01
004	87	IFF
005	40	IND
006	01	01
007	09	09
008	99	99
009	91	R/S

Enter the number of the flag to be tested and press label A. If that flag is set, the calculator display will flash the flag number.

• PPX member, Ralph W. Synder of Indianapolis, Indiana, sent us a program which demonstrates two programming tricks which can save program steps. The program he wrote approximates the rate of interest on an ordinary annuity or annuity due based upon an expanded form of Bailey's formula. One unusual aspect of this program is that the equation for ordinary annuity and annuity due differ only by three signs. Mr. Snyder noted this and took advantage of it. Given below is the program plus two program notes which point out the programming "tricks".

ORDINARY ANNUITY LISTING

000	43	RCL	015	01	01	030	53	(	045	54	)
001	03	03	016	87	IFF	031	53	(	046	54	)
002	65	×	017	02	02	032	87	IFF	047	55	÷
003	43	RCL	018	00	00	033	02	02	048	53	(
004	01	01	019	21	21	034	00	00	049	87	IFF
005	55	÷	020	85	+	035	37	37	050	02	02
006	43	RCL	021	75	-	036	75	-	051	00	00
007	04	04	022	01	1	037	85	+	052	54	54
008	95	=	023	95	=	038	53	(	053	75	-
009	45	YX	024	75	-	039	24	CE	054	85	+
010	53	(	025	01	1	040	65	×	055	03	3
011	02	2	026	95	=	041	43	RCL	056	95	=
012	55	÷	027	55	÷	042	01	01	057	24	CE
013	53	(	028	02	2	043	75	-	058	99	PRT
014	43	RCL	029	65	×	044	06	6	059	91	R/S

User Instructions:

1. Enter program.
2. Initialize by pressing RST.
3. Enter present value, press STO 04.
4. Enter number of payback periods, press STO 01.
5. Enter the amount of payments, press STO 03.

6a. Compute the ordinary interest rate by pressing R/S.
6b. Compute the annuity due interest rate by pressing, St flg 2, R/S.

The following data was entered and the computation was traced using a PC-100A. Due to limited space, the trace given below is only for the case where the + followed by a - causes an error condition.

Data for trace:

Ordinary annuity

Present value	$6115.646855
# periods	20
Payments	$450.

PC-100A Trace for ordinary annuity:

```
           0.      RCL
                     3
         450.
         450.      ×
         450.      RCL
                     1
          20.
          20.      ÷
        9000.      RCL
                     4
 6115.646855
 6115.646855       =
 1.471635007
 1.471635007       YX
 1.471635007       (
           2.      ÷
           2.      (
           2.      RCL
                     1
          20.
          20.      IFF
                     2
                    21
          20.      +    } Error condition caused by two signs
          20.      -    } encountered in succession
           1.    ? =
 1.037482937     ?
 1.037482937     ? -
           1.    ? =
 .0374829371     ?
 .0374829371     ? ÷
           2.    ? ×
 .0187414686     ? (
 .0187414686     ? (
 .0187414686     ? IFF  ←Duplicates display value.
                     2
                    37
 .0187414686     ? -
 .0187414686     ? +
 .0187414686     ? (    ←CE reenters display value
 .0187414686       ×     and removes error condition.
 .0187414686       RCL
                     1
          20.
          20.      -
           6.      )
-5.625170629
-5.625170629       )
 5.643912097
 5.643912097       ÷
 5.643912097       (
 5.643912097       IFF  ←Duplicates display value.
                     2
                    54
 5.643912097       -
 5.643912097       +
           3.    ? =
 .0400070794     ?      ←CE removes error condition.
 .0400070794       PRT
                   R/S
```

Program Notes:

1. The first trick uses a key sequence (shown bracketed in the Ordinary Annuity Listing) which allows the legal use of a plus sign followed by a minus sign. This technique, which could be particularly useful when programming equations such as the quadratic formula, is explained below.

If flg	
2	
n	Transfer address varies depending upon the
nn	location of second sign.
+	Order varies depending upon the two
-	equations used.
CE	See program step 57.

CARTESIAN GRAPH

This program graphs ordered pairs of the form X.Y providing X and Y are both positive integers between 1 and 9 inclusive. As many points as desired of the possible 81 can be plotted. Run time for the graph is about 25 seconds. This program can be used as a subroutine or in conjunction with another program to produce graphs of calculated data points. A PC-100A/C is required.

PPX wishes to thank the author of "Cartesian Graph", Jared Weinberger, for his excellent program.

User Instructions:

1. Partition to 399.69 by pressing 7 OP 17.
2. Enter Program.
3. Enter the following contents into registers R_{40} to R_{69}. (by entering the constant and pressing STO nn, where nn is the appropriate register number).

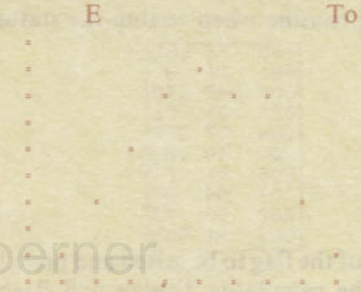
Constant	Register	Constant	Register
40004000.	40	2.	55
4.	41	2.	56
8.	42	2.	57
12.	43	3.	58
16.	44	3.	59
20.	45	4000400040.	60
24.	46	400000.	61
28.	47	40.	62
32.	48	40000000.	63
36.	49	4000.	64
0.	50	4000000000.	65
0.	51	400000.	66
0.	52	40.	67
1.	53	40000000.	68
1.	54	4000.	69

4. Repartition to 479.59 by pressing 6 OP 17 and record magnetic cards.
5. Press A to initialize (clears R_{00} to R_{39} and sets correct partition).
6. Enter data point in X.Y form (where X and Y are positive integers between 1 and 9 inclusive). Press C. The coordinates X.Y will be printed. (Note: To supress automatic printing of coordinates set flag 0 by pressing St flg 0)
7. Repeat step 6 for all data points.
8. Press E to print Cartesian Graph of entered data points.
9. To delete a point, enter the coordinate in X.Y form and press C'.
   (Note: Deleting a point that has not been entered or entering a point that was previously entered will result in an incorrect graph.)

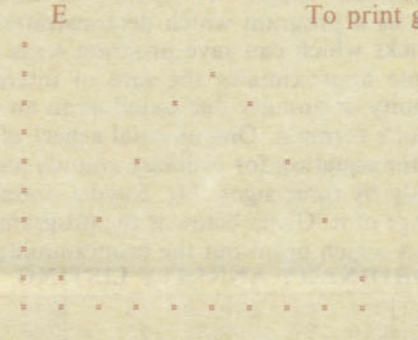
Example:

Graph the following coordinates:
(1,1), (2,3), (3,5), (4,7), (5,8), (6,7), (7,7), (8,3), (9,1) and produce the graph. Then delete point (7,7) and add point (7,5) and print the resulting graph.

Enter	Press	Display	Comments
	A	0	Initialization
1.1	C	1.1	Data point (1,1)
2.3	C	2.3	
3.5	C	3.5	
4.7	C	4.7	
5.8	C	5.8	
6.7	C	6.7	
7.7	C	7.7	
8.3	C	8.3	
9.1	C	9.1	
	E		To print graph



Enter	Press	Display	Comments
7.7	C'	7.7	To delete (7,7)
7.5	C	7.5	To add (7,5)
	E		To print graph



TI-59 Listing*

000	76	LBL	018	16	A'	036	05	5	054	50	50	072	91	R/S	090	69	OP	108	69	OP	126	69	OP
001	11	A	019	42	STO	037	00	0	055	22	INV	073	76	LBL	091	01	01	109	04	04	127	04	04
002	04	4	020	50	50	038	85	+	056	87	IFF	074	15	E	092	69	OP	110	69	OP	128	69	OP
003	69	OP	021	22	INV	039	43	RCL	057	05	05	075	03	3	093	30	30	111	05	05	129	05	05
004	17	17	022	59	INT	040	50	50	058	00	00	076	07	7	094	73	RC*	112	97	DSZ	130	98	ADV
005	47	CMS	023	65	X	041	54	)	059	61	61	077	42	STO	095	00	00	113	50	50	131	98	ADV
006	07	7	024	01	1	042	42	STO	060	22	INV	078	00	00	096	69	OP	114	00	00	132	98	ADV
007	69	OP	025	00	0	043	00	00	061	74	SM*	079	09	9	097	02	02	115	82	82	133	91	R/S
008	17	17	026	85	+	044	73	RC*	062	00	00	080	42	STO	098	69	OP	116	43	RCL	134	76	LBL
009	25	CLR	027	04	4	045	00	00	063	43	RCL	081	50	50	099	30	30	117	60	60	135	18	C'
010	91	R/S	028	00	0	046	95	=	064	50	50	082	69	OP	100	73	RC*	118	69	OP	136	86	STF
011	76	LBL	029	95	=	047	42	STO	065	75	-	083	30	30	101	00	00	119	01	01	137	05	05
012	13	C	030	42	STO	048	00	00	066	06	6	084	73	RC*	102	69	OP	120	69	OP	138	61	GTO
013	87	IFF	031	00	00	049	06	6	067	00	0	085	00	00	103	03	03	121	03	03	139	13	C
014	00	00	032	73	RC*	050	00	0	068	95	=	086	85	+	104	69	OP	122	43	RCL			
015	16	A'	033	00	00	051	44	SUM	069	22	INV	087	43	RCL	105	30	30	123	40	40			
016	99	PRT	034	75	-	052	50	50	070	86	STF	088	65	65	106	73	RC*	124	69	OP			
017	76	LBL	035	53	(	053	73	RC*	071	05	05	089	95	=	107	00	00	125	02	02			

**Note: Key in steps 112 through 115 by pressing Dsz, 1x1, STO 82, BST, BST, and 0. Then single step (SST) to location 116.*

From The Analyst's Desk (Cont.)

When these steps are executed in succession (as happens when flag 2 is not set), the calculator performs the first operation and the result is correct. The CE keystroke is used to remove the error condition.

2. The second trick uses the If flg to mimic the CE keystroke in duplicating the display value (see steps 32, 39, and 49 of the Ordinary Annuity Listing and Personal Programming page V-15). The difference is that the If flg does not remove the error condition.

• To 'bug' is human . . . every programmer that has written a program is familiar with errors or program bugs. If you find an error in a program that you have submitted to PPX, please send a corrected page to be exchanged with the page that has the errors. PPX analyst's will replace the old with the new. When changing the listing of a program, always send magnetic cards with the revised listing.

THE NEW "E" ADDENDUM

With this addendum, which all members should have received by March 15, there are now over 2100 programs listed in our catalog. Due to limited space we can only mention a few of the 500 programs in the E addendum. Here are a few programs PPX analysts found to be especially interesting.

• With the housing situation being what it is, "Buying vs. Renting a House" (PPX #088013E) could be a timely investment. This program calculates the buyer's first year federal tax advantage, equity value, and the renter's comparative financial position.

• Graphic output of deviations is now available with "List and Plot Deviations from the Mean" (PPX #268028E). Using up to 42 entries, this program finds the mean and plots the number of standard deviations from the mean for each entry. In addition, it generates a plot showing the amount each entry differs from a user selected value.

• "Section Properties — Complex Areas" (PPX #668079E) calculates just about everything you ever wanted to know about a section: composite area, centroid location, polar moment of inertia, moment and product of inertia about the axes parallel to user defined axes, orientation of the principal axes, and maximum and minimum moment of inertia. This program can be used for any complex plane area that can be divided into rectangular, triangular, circular, semi-circular, rounded corner, and general section areas.

• Do you dread the chore of checking your child's math homework? Well, never fear, "Print Long Multiplication" (PPX #928025E) is here. Using a PC100A/C, this program prints out the whole process of long multiplication as taught in grade school text books.

• PPX analyst's are still trying to guess the secret number in "Son of Jive Turkey" (PPX #918142E). The fiendish critter, unlike our original "Jive Turkey", generates a new truth probability which is displayed before each guess.

• New capabilities are possible with "Decimal/Fraction Conversion" (PPX #368011E) Using this program, non-repeating and repeating (up to 10 repeating digits) decimal numbers can be converted to exact fractions, and vice versa.

• You and your TI-59 can make beautiful music together with "Guitar Chord Teacher" (PPX #986009E). Upon the entry of one to twelve notes making up a musical chord, this program prints, on the PC-100A/C, the entire neck of a guitar showing all the positions on frets one through fifteen in which the chord may be played.

• A new program that could prove very useful is "Progressions" (PPX #398058E). This program can compute the Nth term and the sum of the first N terms of an arithmetic, geometric, harmonic, or arithmetic/geometric progression.

The PPX [Exc]hange is published every other month and is the only newsletter published by Texas Instruments for TI-59 owners. You are invited to submit items you feel are of general interest to other TI-59 users. Inputs should be limited to 3 double-spaced typed pages. Please forward your newsletter inputs and any questions to:

TEXAS INSTRUMENTS PPX
P.O. Box 53
Lubbock, TX 79408
Attn: PPX Exchange Editor

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

#1034532-19