* TI PPC NOTES *

NEWSLETTER OF THE TI PERSONAL PROGRAMMABLE CALCULATOR CLUB.

v7n9, 1982.

9213 Lanham Severn Road, Lanham MD 20706, USA.

Dear Friends, you are too well aware of the fact that there is a time for coming and another one for going. If you overstay your welcome, you risk to become a terrible bore. So, what that in mind, I have decided to retire as the editor of the Notes, and to give a young and dynamic new editor a chance to show off his stuff. This means I will leave you in the able hands of Palmer O.Hanson Jr., extremely well-known in these pages as one of the most prolific contributors. I can't think of anybody more capable of continuing the work we started three years ago. I am confident that under his guidance the Notes will bloom as never before.

Of course, I will not completely disappear from the scene. I will, so to say, stand in the wings and perform any duty requested of me, giving encouragement and making this way a general nuisance of myself.(!!) I suppose one of my duties will be to keep contact with the many European clubs, unless somebody wants to take over the duty of translating newsletter articles from Danish, Swedish, Dutch, German, French and Spanish.(don't go tell anybody now that I speak all of these languages, please. I just translate them, that's all.)

To most of the members, the actual mechanics of the transfer will be completely transparent. The very last issue (the next one, in November) will contain a subscription form that will have your mailing label pasted on it (to eventually correct your name or address) and it will have Palmer's address on it, to tell were to send it to. I will send Palmer all the as-yet-unpublished manuscripts and business will go on as usual. If Palmer has some ideas how to run the newsletter, and I am sure he has some innovations up his sleeve, he will certainly expound them in the very first issue of 1983. I hope you will all stick with him (as you were stuck with me for three years) in this coming year, the exciting year of the TI-88. I hope you will continue sending those fantastic articles that have made the Notes the most readable newsletter in the business. In one sentence, I hope you will support Palmer in any way you can. I certainly will.

From your many letters I read that you are all hungry for some concrete news and details about the TI-88. I have included quite a lot in this issue and I will try to publish some more in the next one. I have tried to concentrate on the features not found in the 59: new ways of printing, HIR control, prompting and on defining variables and inputting them. I also include a short and very preliminary speed check of the various functions. It will at least give you a grosso modo idea as to what speeds to expect. I translated several simple, but slowly executing, TI-59 programs and found on the average a speed increase of two to threefold. User-friendliness has increased by at least a tenfold factor.

The highlight in this issue: without a doubt Dejan Ristanović's TI-59 SUPERTEST. If you pass this one with a minimum of 70% you know the 59 and you may graduate to the 88.

Maurice E.T. Swinnen.

IN THIS ISSUE:

```
0000 Lbl E
0002 Dfn A
0004 Prt
0005 Dfn B
0007 Prt
0008 A
0009 ×
0010 B
0011 =
0012 ᴬₗᴾₕ
0013 C
0014 =
0015
0016 $
0017 ■'
0018 ■'
0019 ■'
0020 ■'
0021 ■'
0022 ■'
0023 ■'
0024
0025 B
0026 A
0027 L
0028 ᴬₗᴾₕ
0029 Prt
0030 Adv
0031 R/S
0032 Nop
0033 Nop
0034 Nop
0035 Nop


           12.35
           10.89
C= $134.491 BAL

          123.45
          456.12
C= $56308.0 BAL

          2.3+06
          1.8+04
C= $4.14+10 BAL

      1.234568+07
      2.345679+07
C= $2.8959+ BAL

      4567899876.
       987654321.
C= $4.51150 BAL
```

PRINTING ON THE TI-88.- Besides the PRT command, both from the keyboard and under program control, the TI-88 has a couple of specialized forms of printing of which the first one is roughly equal to our familiar OP 4 OP 6 printing, but of which the second one is far superior to anything we have seen up to now. But even the simple PRT has been improved in that it will print anything you put in the display: alpha characters or numerics, for a total of 16 columns. Alpha characters cannot be stored in data registers, tough, as in the 59. They will have to written in-line in a program.

Now with respect to the two forms of specialized printing: The first one requires you to call or bring otherwise the numeric result in the display, after which you simple write up to 4 alpha characters in-line and follow everything with a PRT command. The alpha characters will be printed and displayed in the first 4 columns on the left and the numerical data will occupy the rest. This is preferrable above OP 4 OP 6 printing in the 59, as we now can show much clearer results, such as (for example) "TOT= 1234.56." The fact that it is also displayed (for printer-only use you could replace the PRT by a PAU or an R/S) will encourage a lot of people to use the calculator all by itself, without having to purchase a printer right away.

The second and much more powerful form of printing is called the BLOCK function and is similar to the PRINT USING command in Basic. It allows you to put a few alpha characters in the display, follow it with a predetermined number of digits which are being pulled from the numeric register (which contains the result of the computation), again followed by some more alpha.

The sample program on the left demonstrates how this block function may be used and sometimes abused. We have 16 columns at our disposal. The first four are occupied by alpha (C, =, a space and $). Then we placed 7 blocks, to pull 7 digits from the result. And to top it off we added three more alpha characters (BAL). We used only 15 columns, so we could have placed one more block. Up to the first three samples, everything is OK. But in sample # 4 the +14 exponent is missing, due to space limitation. In the fifth sample the error committed is much more severe. Here not only the exponent is missing, but the number itself is severely truncated.

Note that the Dfn function, used twice in this sample program, is described somewhere else in this issue. Suffice it to say here that when the calculator encounters this function, it stops and waits for your entry of data. Data will be stored in the corresponding letter register. The "define" function is very similar to the INPUT statement in Basic.

The way printing is provided for in the TI-88 is definitely an improvement upon the arrangement in the TI-59. But what is unfortunate is that TI elected to come out with a 16-column printer, as opposed to the 20-column one we were using already. That is not exactly what I would call progress. A 32-column printer, such as the one used with the TI-99/4A home computer would have been ideal.

DUNGEONS AND DRAGONS: Dave Leising brings to my attention the marvelous program called MISADVENTURE that appeared in the Sept/Oct 1981 issue of the PPX newsletter. David S. Lane, who wrote this masterpiece surely has to be congratulated. I once tried but gave up. I concluded that there were not enough steps and/or registers available to do it with. Dave Leising came to about the same conclusions when he and Ken Ward tried it. Now this David Lane did it with 4 registers and 16 steps to spare! A real "fugue for the TI-59."

So, Dave Leising opted for the next best thing to do: Write a solution to this game. Dave says he suspects that this is not the only solution possible. As such a solution program executes rather slowly Dave wrote it in Fast Mode. The program fits on four card sides, i.e. two mag cards. Everybody who remembers his war movies will easily decipher the words on top of the print-out as meaning "ferry zekret."

```
7373733637.   04
3517312200.   05
2217231724.   06
3073737373.   07
        0.    08
        0.    09
        0.    10
        0.    11
2732131600.   12
1513351636.   13
  33351736.   14
3600170000.   15
3100310031.   16
  37320016.   17
3232350017.   18
  37320000.   19
3124152317.   20
  30132224.   21
1500221737.   22
3600261745.   23
 200141315.   24
2600430031.   25
  37233541.   26
  16323235.   27
3013222415.   28
  22173736.   29
  26174503.   30
  17003732.   31
3532323000.   32
4100221737.   33
  15132217.   34
  14131526.   35
1600170022.   36
1737003532.   37
3317004100.   38
2217370000.   39
2241310014.   40
1315260016.   41
  14131526.   42
  43004300.   43
3100331336.   44
3600152724.   45
2121003100.   46
1513221700.   47
1424351600.   48
1600373200.   49
1513421700.   50
1600221737.   51
3013362600.   52
1600430022.   53
1737002241.   54
3000301320.   55
2224150022.   56
1737360015.   57
1331164500.   58
1413152600.   59
1700141315.   60
2600410041.   61
  41003100.   62
```

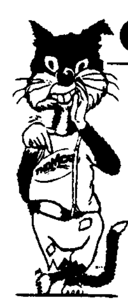
```
3723354100.   63
1417173600.   64
4100310016.   65
  25413033.   66
3600131420.   67
4536360031.   68
  26242727.   69
  36311326.   70
1700410000:   71
3732003132.   72
3226001700.   73
3013222415.   74
  22173736.   75
3324444503.   76
  14131526.   77
  43001600.   78
3100144500.   79
3324444502.   80
  31003313.   81
3636002231.   82
3230170031.   83
3723354100.   84
2213371700.   85
3100373200.   86
2217370000.   87
7373737373.   88
7373732232.   89
2716737373.   90
7373737373.   91
```

```
???STRENG GEHEIM????

LOAD CARDS PRESS E
N N N TO DOOR E TO
NICHE MAGIC GETS KEY
1 BACK W N THRU DOOR
MAGIC GETS KEY2 E TO
ROOM U GET CAGE BACK
D E GET ROPE U GET
GUN BACK D BACK W W
N PASS CLIFF N CAGE
BIRD D TO CAVE D GET
MASK D W GET GUM MA-
GIC GETS CANDY BACK
E BACK U U U N THRU
BEES U N D JUMPS AB-
YSS N KILL SNAKE U
TO NOOK E MAGIC GETS
PIXY2 BACK W D N BY
PIXY1 N PASS GNOME N
THRU GATE N TO GET
????????GOLD????????
```

```
000  00  0
001  00  0
002  00  0
003  00  0
004  00  0
005  36  PGM
006  02  02
007  71  SBR
008  02  02
009  39  39
010  09  9
011  00  0
012  76  LBL
013  11  A
014  22  INV
015  58  FIX
016  22  INV
017  57  ENG
018  01  1
019  99  PRT
020  25  CLR
021  91  R/S
022  99  PRT
023  25  CLR
024  91  R/S
025  99  PRT
026  25  CLR
027  91  R/S
028  99  PRT
029  98  ADV
030  98  ADV
031  98  ADV
032  01  1
033  00  0
034  69  OP
035  17  17
036  02  2
037  02  2
038  42  STO
039  00  00
040  04  4
041  42  STO
```

```
042  01  01
043  73  RC*
044  01  01
045  69  OP
046  01  01
047  69  OP
048  21  21
049  73  RC*
050  01  01
051  69  OP
052  02  02
053  69  OP
054  21  21
055  73  RC*
056  01  01
057  69  OP
058  03  03
059  69  OP
060  21  21
061  73  RC*
062  01  01
063  69  OP
064  04  04
065  69  OP
066  21  21
067  69  OP
068  05  05
069  97  DSZ
070  00  00
071  00  00
072  43  43
073  06  6
074  69  OP
075  17  17
076  98  ADV
077  98  ADV
078  98  ADV
079  25  CLR
080  91  R/S
081  61  GTO
082  00  00
083  29  29
```



Chit-Chat!

Pssst! Nice program, you know! And very eenteresteeng!

EXECUTION SPEED ON THE TI-88. Maurice Swinnen. I ran a preliminary speed check on the 88 and compared it to the execution speed on the 59. The method I used is shown in the program on the left.

```
0000 Lbl E
0002 Sto A
0004 Sto B
0006 AlPh
0007 N
0008 =
0009 AlPh
0010 Prt
0011 Time
0012 OP 28
0015 Sto C
0017 CLR
0018 Dsz A
0020 Gto 0017
0025 Time
0026 OP 28
0029 -
0030 C
0031 =
0032 ÷
0033 B
0034 =
0035 OP 29
0038 ×
0039 1
0040 EE
0041 7
0042 =
0043 Inv
0044 EE
0045 AlPh
0046 m
0047 S
0048 =
0049 AlPh
0050 Prt
0051 Adv
0052 R/S
0053 0
0054 0
0055 0
0056 0
0057 0
0058 0
0059 0

N=          100.
mS=  96.99999984
```

When I want to check the speed of a computer using Basic, I write something like:

100 FOR I= 1 to 1000
110 (function to be timed)
120 NEXT I
130 STOP

I first run the FOR-NEXT loop without the function to be timed and check the time it takes to complete 1000 runs through the loop. Next I insert line 110 with the function to be timed, such as, for example, LET A=25, or sin(A), or SQR(A), or what have you. Then I run it again and time it. I subtract Time(1) from Time(2) and divide by 1000 to get the exact execution time of the function. It usually will result in the number of milliseconds needed.

In calculator language we can do something very similar by using the DSZ command. Look at the program on the left. At line 0018 is says: DSZ A. On the next line it says: GTO 0017. This means: DSZ register A. If the contents of A is not zero, go to line 0017. If it is zero, jump over line 0020 and continue. Line 0017 contains CLR, which is the function to be timed. I inserted here one or more lines with functions to be timed, as shown in the table that follows.

This program allows you to enter the number of loops you desire: 100, 500, 1000. The more loops, the higher the accuracy of timing, but the more tedious the job. When you enter the number of loops, say 100, and press E, the number ends up in register A. Then that number is printed as N= nnn. Next the internal time is called and converted to decimal time by means of OP 28. That result is stored in register C. Then the time loop, described above, is performed. So we know exactly when the time loop started. It is stored in C. After the time loop is finished we call the time again, convert it to decimal time and subtract the starting time from it. Because the number of loops had to be stored in A and subsequently DSZed, we had it also stored in B at the beginning. It is now still available to used as the divider to divide the difference in starting and stopping time by. The result of that division is the exact time it took for the timing loop plus the function to be timed, here CLR. As we had already a prior run of the timing loop alone, without the CLR , we mentally subtract both times form each other to obtain the correct time of the function alone. In my calculator the timing loop alone ran consistently at 86 mSec. From the example you can see that CLR took 97 mSec. Thus, CLR took 97 - 86 = 11 mSec.

In line 0035 the OP 29 converts decimal time back to HH.MMSSd time. The rest of the program simply prints the result as mS= nn.dddddd.

The reason I give this method in such detail is to allow others to duplicate this method and fine-tune execution times some more. It is possible that we will find even faster times in production models of the 88. The calculator I have is a prototype (# 0000285) which, according to the experts in Lubbock, is not fine-tuned at all.

FUNCTION on TI-88	TIME on 88 in mSec	Equiv. on 59 (mSec)	FUNCTION on TI-88	TIME on 88 in mSec	Equiv. on 59 (mSec)
Nop	4	15	LBL A	18	65
RCL Z	18	132	STO+ Z	45	132
STOX Z	60	182	LOG =	640	220
LN =	620	140	100^2	70	133
$\sqrt{100}$	85	143	STO IND Z	60	162
STO+ IND Z	70	192	STOX IND Z	110	212
STF 0	18	96	IFF 0	20	156
INV STF 0	20	172	10!	615	3000
69!	610	15000	INT =	25	40
IF>Z GTO 0024	80	328	INV INT =	25	56
DSZ Z GTO 0024	78	338	SIN 30 =	600	452
COS 30 =	600	452	TAN 30 =	430	342
30⇄ 30 P→R	1700	1282	INV SIN .5 =	530	468
30⇄ 30 R→P	950	1298	INV COS .5 =	530	468
π ↑ 5 =	215	412	INV TAN .5 =	355	412
1.5 PAU	1525	---	CE	9	15
			CLR	11	17

From the foregoing table we see that the common functions, such as STO and RCL are lightning fast. Even all the indirect register functions are almost three times as fast as on the 59. The flags again don't take any time whatsoever. Also the comparisons zip along at a better than fourfold increase in speed. The trigonometric functions have not improved in speed whatsoever and neither has the P to R and R to P functions. All in all, it is going to be a worthy contender in future challenges with the HP club.

To see if my method was in the ball park, I checked my loop time versus the pause time. The latter can be set by means of an OP code. I ran a consistent 25 mSec over the set time, which could quite possibly be the overhead time the PAU command needs. Or it could also be the error I made in timing my loop. Future measurements will tell. I hope this method will be refined.

But, as always and as I often reiterated in these pages, the proof is in the pudding. So, let's write some speedy factor finders, calendars, pi to 1000 places and other "geschwindigkeitsprogrammen" and let's show that we can make this baby sing!

SUPERCHECKSUM, Erratum. Björn Gustavsson tells me that he managed to confuse me completely in last issue (v7n7/8p8) by sending me an erratum at the last possible moment, before the issue was poured in concrete. He sent me an error in the erratum. All the addresses should have been 3 higher. So, the sequence to correct the bug should be:

GTO 159 LRN CLR RCL 35 LRN 1 WRT (insert card) The LOG is located at step 162 and the error condition is cleared at step 183.

I (the editor) tried this correction and I am convinced the superchecksum program works now perfectly.

So, please try it again and let me know what you think of using this one as THE offical checksum program of the TI PPC NOTES.

This correction was obviously needed because without it, the FIX 0 EE in steps 163-165 will round the exponent to one significant digit. This is because of the EE mode. Therefore the one's digit will be lost. Inserting CLR before recalling R35 will remedy this by clearing the EE mode.

HIERARCHY REGISTER CONTROL IN THE TI-88. The calculator has 13 instructions that allow you to access the 63 hierarchy registers (HIRs), addressed 00 through 62, either directly or indirectly. Of course, before we want to do this, we should have a good understanding of what these HIRs do and how the digits internally are positioned and what their individual meaning is. TI has published quite a lot this time. But don't be lulled into a soft sleep either. The 63 HIRs are the ones TI chooses to leave unlocked, so we can access them. I have good reasons to believe there are more HIRs, but , for the time being, we have no way to access them. Just give us time, though, we will find a way.

But first, let's concentrate on the ones we have and on their functions. The first thing to remember is: NO ACCESS FROM THE KEYBOARD. Everything has to be done in a program. A nice program to list all the 63 HIRs is: LBL E OP 14 2nd ALPH 2nd Time 2nd ALPH 0 INV Lst INV 2nd ALPH 2nd Time 2nd ALPH OP 15 INV SBR. Once you have keyed in this program and you list it, you will see that the special sequence 2nd APLH 2nd Time 2nd ALPH will list as $. This is the sequence that places the calculator in HIR mode or when preceded by INV will take it out of HIR mode. The hex code for the $ function is FC. Looking at what the program does: When you press E, OP 14 places the calculator in UNFORMATTED mode, that is you have access to all 16 internal digits in a register. Next the $ function places the calculator in HIR mode, after which the 0 INV LST does the listing of all 63 HIR registers, the same as in the 59 a 0 INV LIST will list all data registers, starting with register 00. After the listing, the INV $ takes the TI-88 out of the HIR mode and OP 15 brings it back into FORMATTED mode, the normal mode one should do computations in.

The Utility register is located in HIR 49. That is at least the address you should use to store something in it or recall from it, even if one day you should notice a copy of its contents somewhere else. To prove it is located in HIR 49, we will store a bunch of 5's in HIR 49 and hope to find it back later by pressing the Utility register key. By the way, this is a simple exchange register much like our familiar t-reg. Storing into a HIR should be done under program control only. So, in LRN mode, we key in: LBL A 5555555555555555 2nd ALPH 2nd SHIFT (this is the +/- key) 2nd ALPH 49 (the address) CE/C CE/C R/S and we execute this one by pressing A. Now we press the Utility register key and we see 5.555556 ↑+14. Which proves.....

I also pressed from the keyboard, before executing the program, the following sequence: 1 + (2+ (3 + (and so on up to 8 + (, at which time the calculator begged for mercy and flashed the message AOS STACK FULL, just as I expected it to do. Then, when to program executed the 1, 2, 3 and so on, up to 8, showed up in the first eight HIRs, showing that there is the exact location of the AOS stack. (Even if TI says it is there, we don't have to believe it and it is always prudent to check it out for ourselves, witness the utility register)

I will not bore you with giving you the exact sequences for all 13 HIR instructions. Once you have your calculator, you will also have a book and hopefully chapter 4, under Advanced Programming, will tell you how to do it. Otherwise, we will put everything we know in the NOTES. Suffice it to say that there are instructions for: 1. Placing the calculator in HIR mode, hex code FC. 2. Cancel HIR mode. 3. Placing the calculator in INDIRECT HIR mode, hex code FD. 4. Cancel INDIRECT HIR mode. (both "cancel"s are done by placing "INV" in front of the instruction) 5. RCL HIR, hex code FE. 6. STO HIR, hex code FF. 7. STORE DIGIT, hex code FA. 8. RECALL DIGIT, hex code FB. 9. SET BIT, hex code F6. 10. RESET BIT, hex code F7. 11. FLIP BIT, hex code F8. 12. TEST BIT AND EXECUTE IF SET, hex code F9 and 13. TEST BIT AND EXECUTE IF RESET, obtained again by placing INV in front of # 12.

Needless to say that TI places a stern warning in the book, telling you that these registers are used internally by the calculator and that changing their contents without knowing how these registers are used, can result in loss of option settings (not severe), memory loss (rather severe) and locking up the calculator (fatal). In the latter case you sometimes might recuperate by turning everything off and starting all over again. I found one instance where it didn't help at all. I had to remove the battery and lost my time and date settings in the process. The only things saved were my few utility programs in the constant memory module. After I installed the battery again, everything worked just fine.

HIR Listing.

HIR Listing	ADDRESS	FUNCTION
1000000000000000	00 or A	AOS stack
2000000000000000	01 or B	AOS stack
3000000000000000	02 or C	AOS stack
4000000000000000	03 or D	AOS stack
5000000000000000	04 or E	AOS stack
6000000000000000	05 or F	OAS stack
7000000000000000	06 or G	AOS stack
8000000000000000	07 or H	AOS stack
0000000000000000	08 or I	Yn statisticics register (CLR by OP 37)
0000000000000000	09 or J	Qyn statistics register (CLR by OP 37)
0000000000000000	10 or K	N statistics register (CLR by OP 37)
0000000000000000	11 or L	Xn statistics register (CLR by OP 37)
0000000000000000	12 or M	Qxn statistics register (CLR by OP 37)
0000000000000000	13 or N	Rn statistics register (CLR by OP 37)
1200000000000010	14 or O	Yn statistics register (CLR by OP 37)
7500000000000000	15 or P	System operations
6200000000000010	16 or Q	Systems operations
1700000000000010	17 or R	Systems operations
1500000000000000	18 or S	Systems operations
C00A201002700412	19 or T	Systems operations
00087F60013F6A17	20 or U	Systems operations
00022C5D523B4C01	21 or V	Systems operations
000F300000000022	22 or W	Systems operations
0000000000000023	23 or X	Systems operations
6200000000000010	24 or Y	Systems operations
0012000000000003	25 or Z	Systems operations
005990010180201C	26	Position 7= SBR stack, position 8=language digit
0000000000000014	27	Systems operations
0006320C30000000	28	Subroutine stack
082346OE50000000	29	Subroutine stack
0043544D30000000	30	Subroutine stack
0311968E50000000	31	Subroutine stack
1186360E50000000	32	Subroutine stack
2000000000000000	33	Systems operations
8000000000000022	34	Systems operations
0047900000000040	35	Systems operations — Pos D-F= regs
00960000A0020060	36	Systems operations — Pos 1-4= max pgm steps possible
0047900603602201	37	Systems operations — Pos 1-4= last pgm step in partition
0826860000000001	38	Systems operations
1599900000000101	39	Systems operations
5555555555555555	40	Systems operations.
080003010B400900	41	System Operations
E200000000000000	42	System operations
0000000000000000	43	System operations
000F0000008F0181	44	Digit in position 5 = Cursor pointer
4424442442204422	45	Alpha display register
E7F8B6ADDED068DE	46	Alpha display register
0000000000000000	47	System operations
0000000000000000	48	System operations
4900000000000010	49	System operations (utility register)
0000000000008F01	50	Auxiliary operator stack (for pending unary operations)
5000000000000000	51	System operations
5100000000000000	52	System operations
5200000000000010	53	Numeric display register
1000000000000000	54	System operations
808000F000010010	55	System operations
2424242424242424	56	Operator stack for pending operations
0000600000000006	57	System operations
082319824400030	58 *date*	System operations
211A552499102010	59 *time*	System operations
093800006870E010	60 *alarm*	System operations
2222222222222222	61	System operations
2222222222222222	62	System operations

59 time, pos.A: 1=D/M/Y, 0=M/D/Y

Language digit (register 26, position 8):

0=English
1=German
2=French
3=Italian
4=Dutch
5=Swedish
6=Spanish
7 to F= Future use

```
0000 Lbl J
0002 "
0003 L
0004 I
0005 S
0006 T
0007
0008 H
0009 I
0010 R
0011 s
0012 "
0013 Pau
0014 Rtn
0015 Lbl E
0017 OP 14
0020 $
0021 0
0022 Inv
0023 Lst
0024 Inv
0025 $
0026 OP 15
0029 Adv
0030 Adv
0031 Rtn
```

Digit positions in each register.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

SUPER TI-59 TEST- Dejan Ristanović sent me this one just a few days before mine appeared in the PPX newsletter. This test, of course, is not for just anybody. It is intended for the "fanatics", the ones that sleep with their calculator and in the middle of the night wake up to try out some routine they have been dreaming about. If you are just an average TI-59 user, may I wish you luck?

As this is a TI-59 test, the calculator itself will be the judge as to how good you did. First, key in the program (funny, where are the answers? The TI-59 knows, rest assured!) and when you are ready to take the test, just press A. The TI-59 will not only take into account the correctness of your answers, but it will also measure the actual time you used to arrive at each answer!!!!

While doing the test, you are allowed pencil and paper, but having another TI-59 at your disposal is considered "tacky". A better, but unfortunately untranslatable word I learned in my youth back in Flanders, says it is "haarzak", which contains the connotations of "dishonest, unfair, crude, sneaky and boorish." Whatever, I wouldn't do it.

So, read the questions and when you think you are ready to provide answers, press R/S and hold it for 0.5 to 1 sec. The TI-59 will display the digits 1, 2, 3, 4 and finally 5. Once you see the digit belonging to the right answer displayed, press R/S again, and go on to the next answer. After you give the last answer, the calculator will generate and print a table of right answers. (you see, they are in there somewhere!) Then, below the print out of the table, points for "time" will be given, between -5 and +5. About 10 minutes should be an average answering time. Then the total is printed and you will receive your final grade. You need at least a "6" to pass.

If at any time you hold R/S too long, program execution will be terminated. Don't worry, though. Just press R/S again to restart. Good luck!

Question 1, worth 6 points: From cold start you pressed RST LRN GTO LRN, thus creating a one-line program GTO. Now press RST SST 1 2 3 4 5. The display reads

[1] 12345 Blinking
[2] 5 Blinking
[3] 0 Blinking
[4] 0
[5] 45

Question 2, worth 6 points: In a program there is a 1123 STO 00 GTO IND 00 sequence. After its execution:

[1] the calculator will detect an error by blinking its display but will continue execution.
[2] GTO 123 will be performed.
[3] GTO 123 will be performed but the display blinks after program execution is terminated.
[4] program execution is terminated and display blinks.
[5] no operation is performed and program execution is terminated. That means, this sequence acts as if it were an R/S.

Question 3, worth 1 point: Pressing CP from the keyboard, besides clearing program and T-register, also clears:

[1] flags, subroutine return register, previous calls to library routines, and all HIRs.
[2] flags, subroutine return register and previous calls to library routines.
[3] flags and calls to library routines.
[4] flags and subroutine register
[5] all of the flags.

Question 4, worth 4 points: Load-and go method of initializing Fast mode (PGM 02 SBR 240..etc.) clears everything, except:

[1] flags
[2] HIRs.
[3] OP 01 through OP 04 contents.
[4] partitioning.
[5] subroutine return register & HIRs.

Question 5, worth 1 point: Pressing IND E from the keyboard will:

[1] act as E.
[2] act as E, but cause the display to blink.
[3] just cause the display to blink, without any other operation.
[4] act as GTO IND E.
[5] execute LBL 00 if it exists.

Question 6, worth 3 points: the sequence 2 LOG INV LOG WRITE will:

[1] cause an error.
[2] save bank 3.
[3] save bank 2.
[4] save bank 2 as a protected program.
[5] save bank 1.

Question 7, worth 5 points: At the end of program memory starting at step 475 we have: ADV LIST STO 25 ADV. The last step is 479. The partitioning is 6

OP 17. This program was executed by pressing SBR 475. How many lines, including blank lines if any, were printed by this program execution?
[1] 5 (five)
[2] 4 (four)
[3] 3 (three)
[4] 2 (two)
[5] 1 (one)

Question 8, worth 9 points: You are accumulating data and you want to determine their mean. You write the following program, starting at step 000: DIV 2 = SUM+ R/S RST and press CMS RST after you go out of learn again. Then you enter 6 R/S 8 R/S 12 R/S . At that moment you realize that you should have entered 20 instead of 12. So you enter 12 INV SUM+ 20 R/S. (SUM+ is the SIGMA+ used in statistical entries) Now the display shows:
[1] 0 blinking.
[2] 2.
[3] 3.
[4] 20.
[5] 20 blinking.

Question 9, worth 7 points: You want to synthesize some hex codes and therefore you want to have a look at the internal 59 ROM. Thus, you press n OP 17 CLR PGM 19 SBR 045 P/R LRN. That "n" can be:
[1] any integer larger than 8, that is 9, 10, 11....
[2] 9 only.
[3] 9 or 10 only.
[4] 8, 9 or 10 only.
[5] 8 or 9 only.

Question 10, worth 5 points: Somewhere in user memory there is the following program: 5 +/- EE 99 X:T 6 EE 99 INV EE INV GE CLR 2 R/S LBL CLR 1 R/S After its execution, the display will read:
[1] 2 blinking.
[2] 2.
[3] 1 blinking.
[4] 1.
[5] 6 99 blinking.

Question 11, worth 5 points: This questions is the same as question 10, to a certain extent. But the program is slightly modified. Instead of INV we have now 2nd INV (code 27) before the GE instruction. After execution the display now reads:
[1] 1.
[2] 1 blinking.
[3] 2.
[4] 2 blinking
[5] 6 99 blinking.

Question 12, worth 2 points: A simple program, starting at step 000: 1000 STO 00 DSZ 0 006 R/S will work for about:
[1] 4 seconds.
[2] 30 minutes.
[3] 15 minutes.
[4] 6 minutes.
[5] 3 minutes.

Question 13, worth 10 points: An integer stored in R00 has 2 to 12 digits. The first digit is a 5, the last one is a 9. One of the following procedures will, among other things, print all of the digits of that number:
[1] RCL 00 OP 1 OP 5 0 RCL 0 FIX 9 OP 1 OP 5
[2] STF IND 00 RCL 00 EE INV EE STO 01 FIX IND 01 OP IND 01 OP 05 PRT
[3] CLR STF IND 00 EXC 00 OP 01 OP 06 PRT
[4] STF IND 00 RCL 00 FIX IND 00 PRT
[5] RCL 00 OP 04 12 STO 01 FIX IND 01 RCL 00 OP 06

Question 14, worth 7 points: You want to store the display in HIR 8 and the T-register in HIR 7. You need at least:
[1] 7 steps.
[2] 5 steps.
[3] 3 steps.
[4] 1 step.
[5] no steps at all; they are stored there automatically.

Question 15, worth 9 points: The result of pressing 9 INV LOG is called "x". The result of pressing 10 INV LOG is called "y". Then one of the following is correct:
[1] $x = 10^9$ and $y < 10^{10}$
[2] $x = 10^9$ and $y > 10^{10}$
[3] $x > 10^9$ and $y = 10^{10}$
[4] $x < 10^9$ and $y = 10^{10}$
[5] $x < 10^9$ and $y < 10^{10}$

Question 16, worth 5 points: Placing 21 38 codes in your program will result in a crash. But if there are 50 NOPs between the 21 and the 38:
[1] nothing will happen.
[2] CP will be performed.
[3] PGM 01 SBR 00 will be performed.
[4] a crash will result, but you may recover from that by pressing RST, which is not the case when 38 follows 21 immediately, because RST will do no good then.
[5] a crash anyway.

Question 17, worth 1 point: In HIR 8

you have the number 0.0001111111111; if you now execute under program control HIR 8 OP 05, the PC100 will print:
[1] 88888
[2] 8887
[3] 8888
[4] 0888
[5] 00088

Question 18, worth 8 points: You want to give somebody a protected program that has to run in Fast mode. To initialize the Fast mode you will have to use:
[1] the method of storing 2000000000002 in status register 0 by asking the user to press 7 EE after STF IND at the end of the program.
[2] the same method but by using hex code h12.
[3] either 1 or 2 above, at will.
[4] the load-and-go method, using PGM 02 SBR 240, etc. at the beginning of the program.
[5] any of the methods decribed above.

Question 19, worth 9 points: At step 000 of program memory there is short program as follows: R/S LBL A STO 00 RCL IND 00 RST. You want to trace the routine by pressing STF 9 10 A. If 125 is stored in R10, a few lines will be printed. But the last three lines of the print out will be:
[1] RC* 0 10 125
[2] RC* 0 125 RST
[3] 10 125 RST
[4] * 0 125 RST
[5] RC* *0 125 RST

Question 20, worth 2 points: You want to write a program and include a partitioning-changing routine in it. You also want to protect that program at the same time. What method will you use to change partitioning?
[1] I won't use any. It simply is not possible.
[2] I'll use h12 to store the size needed, in the correct place in status register 0.
[3] I'll use the same idea as in 2, but I'll ask the user tp press 7 INV after execution of STF IND at the end of the program.
[4] either method 2 or 3, as needed.
[5] the standard n OP 17 method.

You will have to choose the unique and complete answer for each question. Dejan says that this program is a synthesis of all the quirks and programming tricks discovered by so many different people that naming them all would fill a whole typewritten page. Everybody will recognize his or her own discovery and be proud that it has been incorporated in this definitive TI-59 Supertest. That should be reward enough.

SEE PROGRAM ON NEXT PAGE, PLEASE.

SUPERTEST,(59)- A few remarks and some clarification are in order, I think. When you have read all of the questions and think you are ready to answer them, press A.

You will see, briefly displayed, a "1", during two pause periods. Then the display will go blank. This means that the internal wheels of the 59 have started to grind away and are timing your response time to question # 1. A response time of 10 minutes is considered average and will earn you zero points with respect to time. Of course, you will still earn some points for that question with respect to the accuracy of your answer. Taking more than 10 minutes for an answer will get you gradually down towards -5 points for time. Doing better than (less than) 10 minutes will get you plus points, up to 5 in total.

When you are sure about your answer to question # 1, tap the R/S key lightly, about .5 to 1 sec long. Now you will see in the display, in succession, a "1", a "2", a "3", a "4" and a "5". Keep your finger on the R/S key and press it down, again for about .5 to 1 sec, once you see the digit corresponding to the correct answer. The display will momentarily go blank, after which a "2" will be flashed, telling you question # 2 is now being timed. Once you think you know the answer....well, you know by now what to do, I suppose. There are 20 questions in total.

After question # 20 the printer will go into action, printing out a table with your score. The table is easy to understand and printing one here would reveal the game. I you didn't think I would do THAT, would you now?

The techniques used in this program are incredible. Try to decipher what Dejan did and how he arrived at such an interactive program. This program is just beautiful.

SUPERTEST TI-59, Dejan Ristanović.

000	69	OP
001	20	20
002	43	RCL
003	00	00
004	32	X:T
005	02	2
006	09	9
007	77	GE
008	02	02
009	86	86
010	98	ADV
011	05	5
012	01	1
013	10	E'
014	01	1
015	05	5
016	42	STO
017	09	09
018	01	1
019	00	0
020	42	STO
021	00	00
022	03	3
023	01	1
024	03	3
025	02	2
026	04	4
027	00	0
028	00	0
029	00	0
030	69	OP
031	01	01
032	03	3
033	05	5
034	02	2
035	04	4
036	02	2
037	02	2
038	02	2
039	03	3
040	03	3
041	07	7
042	69	OP
043	02	02
044	04	4
045	05	5
046	03	3
047	02	2
048	04	4
049	01	1
050	03	3
051	05	5
052	69	OP
053	03	03
054	03	3
055	03	3
056	03	3
057	07	7
058	04	4
059	00	0
060	69	OP
061	04	04
062	69	OP
063	05	05
064	02	2
065	00	0
066	10	E'
067	69	OP
068	00	00
069	43	RCL
070	00	00
071	75	-
072	09	9
073	95	=
074	19	D'
075	65	×
076	01	1
077	00	0
078	00	0
079	95	=
080	69	OP
081	01	01
082	36	PGM
083	15	15
084	71	SBR
085	88	DMS
086	65	×
087	05	5
088	85	+
089	01	1
090	95	=
091	59	INT
092	42	STO
093	02	02
094	19	D'
095	69	OP
096	02	02
097	73	RC*
098	00	00
099	19	D'
100	69	OP
101	03	03
102	36	PGM
103	15	15
104	71	SBR
105	88	DMS
106	65	×
107	01	1
108	00	0
109	85	+
110	01	1
111	95	=
112	59	INT
113	42	STO
114	03	03
115	73	RC*
116	00	00
117	32	X:T
118	43	RCL
119	02	02
120	22	INV
121	67	EQ
122	01	01
123	31	31
124	43	RCL
125	03	03
126	44	SUM
127	05	05
128	19	D'
129	69	OP
130	04	04
131	69	OP
132	05	05
133	69	OP
134	20	20
135	43	RCL
136	00	00
137	32	X:T
138	02	2
139	09	9
140	77	GE
141	00	00
142	67	67
143	03	3
144	07	7
145	02	2
146	04	4
147	03	3
148	00	0
149	01	1
150	07	7
151	69	OP
152	00	00
153	69	OP
154	01	01
155	03	3
156	00	0
157	00	0
158	00	0
159	75	-
160	43	RCL
161	01	01
162	95	=
163	29	CP
164	67	EQ
165	01	01
166	93	93
167	55	÷
168	01	1
169	00	0
170	00	0
171	95	=
172	42	STO
173	08	08
174	69	OP
175	10	10
176	65	×
177	43	RCL
178	08	08
179	50	I×I
180	59	INT
181	95	=
182	32	X:T
183	05	5
184	22	INV
185	77	GE
186	01	01
187	93	93
188	94	+/-
189	77	GE
190	01	01
191	93	93
192	32	X:T
193	44	SUM
194	05	05
195	19	D'
196	69	OP
197	04	04
198	69	OP
199	05	05
200	02	2
201	00	0
202	10	E'
203	03	3
204	07	7
205	03	3
206	02	2
207	03	3
208	07	7
209	01	1
210	03	3
211	02	2
212	07	7
213	69	OP
214	00	00
215	69	OP
216	01	01
217	43	RCL
218	05	05
219	19	D'
220	69	OP
221	04	04
222	69	OP
223	05	05
224	06	6
225	04	4
226	10	E'
227	43	RCL
228	05	05
229	55	÷
230	01	1
231	01	1
232	95	=
233	58	FIX
234	00	00
235	52	EE
236	22	INV
237	52	EE
238	58	FIX
239	09	09
240	19	D'
241	69	OP
242	00	00
243	69	OP
244	03	03
245	03	3
246	00	0
247	01	1
248	03	3
249	03	3
250	05	5
251	02	2
252	06	6
253	00	0
254	00	0
255	69	OP
256	02	02
257	04	4
258	05	5
259	03	3
260	02	2
261	04	4
262	01	1
263	03	3
264	05	5
265	00	0
266	00	0
267	69	OP
268	01	01
269	69	OP
270	05	05
271	05	5
272	01	1
273	10	E'
274	98	ADV
275	43	RCL
276	05	05
277	98	ADV
278	91	R/S
279	76	LBL
280	11	A
281	47	CMS
282	01	1
283	00	0
284	42	STO
285	00	00
286	36	PGM
287	09	09
288	71	SBR
289	00	00
290	58	58
291	36	PGM
292	09	09
293	51	BST
294	43	RCL
295	00	00
296	75	-
297	09	9
298	95	=
299	66	PAU
300	66	PAU
301	69	OP
302	21	21
303	61	GTO
304	03	03
305	01	01
306	76	LBL
307	10	E'
308	55	÷
309	09	9
310	09	9
311	85	+
312	01	1
313	95	=
314	82	HIR
315	05	05
316	82	HIR
317	06	06
318	82	HIR
319	07	07
320	82	HIR
321	08	08
322	69	OP
323	05	05
324	92	RTN
325	76	LBL
326	19	D'
327	29	CP
328	67	EQ
329	03	03
330	52	52
331	32	X:T
332	22	INV
333	77	GE
334	03	03
335	37	37
336	02	2
337	00	0
338	42	STO
339	06	06
340	32	X:T
341	50	I×I
342	55	÷
343	28	LOG
344	59	INT
345	42	STO
346	08	08
347	69	OP
348	28	28
349	22	INV
350	28	LOG
351	85	+
352	01	1
353	85	+
354	28	LOG
355	59	INT
356	65	×
357	01	1
358	00	0
359	00	0
360	49	PRD
361	06	06
362	02	2
363	75	-
364	59	INT
365	44	SUM
366	06	06
367	95	=
368	65	×
369	01	1
370	00	0
371	97	DSZ
372	08	08
373	03	03
374	51	51
375	25	CLR
376	48	EXC
377	06	06
378	92	RTN
379	76	LBL
380	16	A'
381	87	IFF
382	00	00
383	04	04
384	11	11
385	36	PGM
386	09	09
387	71	SBR
388	00	00
389	58	58
390	36	PGM
391	09	09
392	51	BST
393	86	STF
394	00	00
395	05	5
396	42	STO
397	07	07
398	06	6
399	75	-
400	43	RCL
401	07	07
402	95	=
403	72	ST*
404	00	00
405	66	PAU
406	66	PAU
407	97	DSZ
408	07	07
409	03	03
410	98	98
411	81	RST

NEWCOMERS' CORNER, by Bob Fruit. One of the more interesting things you can do with your TI-59 is simulations. Even if the TI-59 has limited memory, it is rather easy to do simulations on it. With proper planning, and reducing a simulation to its essential elements, it should work.

I would like to use as an example the problem of how many check-out clerks are needed to optimize the earnings of a store. I personally have nothing to do with retailing, so the numbers I use may not be 100 % realistic.

A store owner must choose between having check-out clerks on hand and the likelyhood of driving costumers away because they have to wait too long to be checked out. The clerks make $ 10.00 per hour. Costumers come to the check-out line on the average of one per minute. It takes between 1 and 7 minutes to check out a costumer. The store makes on the average $ 1.00 per minute of check out time, taking into account the overhead expenses of the store, including the cost of the check-out clerks. If costumers find they have to wait longer than 15 minutes to be checked out, they will stop coming to the store.

Those are the essential ingredients of the problem I propose. First I will write some of the routines that will be used, before tying everything together into a single program that becomes the simulator.

The first routine will be the random number generator. I prefer to write my own one, rather than use the one from the ML-Library. The latter uses too many data registers. I did use it, however, as a guide to write the following routine for numbers between 0 and 1:

LBL LNX ((RCL 00 X 199017 X 24298 + 99991) DIV 199017) INV INT STO 00 RTN

This random generator uses only one single data register and 39 program steps. The next program is a simulation routine to have the costumers arrive at the check-out counters on the average of 1 per minute. I choose that no costumers arrive if the random number output is less than .3, 1 costumer arrives if the random number output is between .3 and 2 costumers arrive when the random number output is greater than .7. This program then becomes:

LBL X SBR LNX X:T (0 + .3 GE ABS 1 + .7 GE ABS 1 + LBL ABS 0) CP RTN

If I later want to change the schedule of the costumers arriving, it will be easy to do so, because this routine is separate from the rest of the program. If I want to write a program that I intend to use for solving a particular problem, to be used only once, I write each routine separately. It find it makes debugging so much easier.

The routine that decides how long it takes to check out a costumer is a simple linear one:

LBL DIV (SBR LNX X 6 + .5) RTN

The hardest part to write in this simulator program is the routine that figures out which line the next person should go to. I have assigned every fifth data register, starting with register 10, as the check-out clerks' memory area. This means that the first costumer will be checked out and there will be four costumers waiting. If more than that number of costumers show up, the extra people will throw their selected purchases to the ground and walk out of the store. The check out time remaining will be in the data register for the costumer being checked out. When another costumer gets in line and cannot be checked out immediately, the time on the clock (backwards counter) will be saved in the data register. When a costumer moves up to be checked out, the clock at the time he got in line may be compared with the current clock to see how long that costumer was in line.

If no one is in a "lines" position, the value in the data register will be zero.

The routine that selects a "lines" position has three nested loops: costumers to put in line, check-out clerks, and position in a line. As an open place is found, its location is saved if it is closer to the check out position than a previously found one. After all positions are checked, the lowest one found is where the next costumer is placed. The routine that finds a position in line, as well as the one that locates the first position for a given check-out clerk, and the others that run the program are not show here, since they can all be found in the program itself.

The data registers are assigned as follows:

newcomer's corner- Bob Fruit (cont.)

REG	USE	REG	USE
0	random number gemerator	6	costumer counter
1	cost.checked out/wait time	7	clerk counter
2	cost.leave store/ch.out time	8	clerk's memory location
3	number of clerks	9	fives counter
4	not used	HIR 7	clerk's number-low position
5	clock, time to run in min.	HIR 8	clerk's empty low position

This simulator lets you determine several things about the given problem.
1. Did the costumers leave the store because they could not find a place in line? Integer value in REG 2.
2. How many costumers were checked out? Integer value in REG 1.
3. What was the total wait time for costumer's check-out? Fractional part of REG 1 times 10000.
4. What was the total amount of money earned by the store in the given time? Fractional part of REG 2 times 10000.
5. What did the check-out clerks cost the store? REG 3 times 10 times the number of minutes the clock was set to (initial value of REG 5, printed by PC100) divided by 60.

You can calculate the store profit, money earned minus the cost. The average wait time, total wait time divided by costumers checked out (from the given conditions at the start. This should be below 15 minutes). The average earned per costumer, total check-out time divided by the number of costumers checked out.

I ran this program and came up with the following results. The simulator was set for 180 minutes:

ckeck-out clerks	2	3	4
costumers left store	85	42	0
costumers ckecked out	107	143	166
total wait time	1170	1830	1882
total earned	307.36	466.42	611.96
total cost	30.00	45.00	60.00
profits	277.36	421.42	551.96
average wait time	11	13	11

I hope this shows that it is easy to use your TI-59 as a simulator. If you program with separate routines (sometimes also called modules, not to be confused with "solid-state modules") you can make changes to some parameters without needing to find all of the other places it might affect. For instance, if you now feel the check out time's linear nature is wrong, just rewrite that particular module (routine). Anything you do does not affect the rest of the program as long as you don't get into the other data register used by the other parts of the program. Have fun making your own simulators.

See program on next page.

TIBBETTS' CONJECTURE.- Lester Tibbetts of Emporium, PA, saw Ulam's Conjecture in v6n9/10p13 and felt the irresistible urge to enhance.(I think nobody in our club ever felt that way, but there is always the odd ball.) So he reduced the program from 45 steps to 37. What is remarkable, though, is that his program uses only one, single data register, versus the original one four, that he uses label addresses and still managed to run at about 30 to 40 % faster.

And now for Lester's Conjecture:" After running any number, just keep pressing A so you continuously operate on the number of steps it took to complete the previous example, and eventually the number of steps required will decline to one."
Lester says that he doesn't know why it works, and if it always will, but up to now it always did. It must be a great paper waster!

```
LBL A  CMS  LBL INV OP 20  ) PRT  DIV  2  X:T  EQ  LOG  CP  - INV INT  EQ  INV  0  )
X  6  +  1  GTO  INV  LBL  LOG  )  PRT  ADV  RCL 00  PRT R/S
```

NEWCOMER'S CORNER, Bob Fruit.

```
000 76 LBL
001 23 LNX
002 53 (
003 53 (
004 43 RCL
005 00 00
006 65 ×
007 01 1
008 09 9
009 09 9
010 00 0
011 01 1
012 07 7
013 65 ×
014 02 2
015 04 4
016 02 2
017 09 9
018 08 8
019 85 +
020 09 9
021 09 9
022 09 9
023 09 9
024 01 1
025 54 )
026 55 ÷
027 01 1
028 09 9
029 09 9
030 00 0
031 01 1
032 07 7
033 54 )
034 22 INV
035 59 INT
036 42 STO
037 00 00
038 92 RTN
039 76 LBL
040 33 X²
041 71 SBR
042 65 ×
043 67 EQ
044 38 SIN
045 42 STO
046 06 06
047 76 LBL
048 34 ΓX
049 43 RCL
050 03 03
051 42 STO
052 07 07
053 06 6
054 82 HIR
055 08 08
056 76 LBL
057 39 COS
058 43 RCL
059 07 07
060 71 SBR
061 86 STF
062 76 LBL
063 30 TAN
064 73 RC*
065 08 08
066 22 INV
067 67 EQ
068 25 CLR
069 05 5
070 75 -
071 43 RCL
072 09 09
073 75 -
074 82 HIR
075 18 18
076 95 =
077 77 GE
078 25 CLR
079 85 +
080 82 HIR
081 18 18
082 95 =
083 82 HIR
084 08 08
085 43 RCL
086 07 07
087 82 HIR
088 07 07
089 82 HIR
090 18 18
091 67 EQ
092 24 CE
093 61 GTO
094 29 CP
095 76 LBL
096 25 CLR
097 69 OP
098 28 28
099 97 DSZ
100 09 09
101 30 TAN
102 76 LBL
103 29 CP
104 97 DSZ
105 07 07
106 39 COS
107 76 LBL
108 24 CE
109 82 HIR
110 18 18
111 75 -
112 05 5
113 95 =
114 22 INV
115 77 GE
116 22 INV
117 69 OP
118 22 22
119 61 GTO
120 28 LOG
121 76 LBL
122 22 INV
123 69 OP
124 21 21
125 82 HIR
126 17 17
127 71 SBR
128 86 STF
129 82 HIR
130 18 18
131 44 SUM
132 08 08
133 43 RCL
134 05 05
135 72 ST*
136 08 08
137 43 RCL
138 08 08
139 55 ÷
140 05 5
141 95 =
142 22 INV
143 59 INT
144 22 INV
145 67 EQ
146 28 LOG
147 71 SBR
148 55 ÷
149 72 ST*
150 08 08
151 76 LBL
152 28 LOG
153 97 DSZ
154 06 06
155 34 ΓX
156 76 LBL
157 38 SIN
158 92 RTN
159 76 LBL
160 42 STO
161 43 RCL
162 03 03
163 42 STO
164 07 07
165 76 LBL
166 47 CMS
167 43 RCL
168 07 07
169 71 SBR
170 86 STF
171 73 RC*
172 08 08
173 67 EQ
174 44 SUM
175 75 -
176 01 1
177 95 =
178 72 ST*
179 08 08
180 77 GE
181 44 SUM
182 04 4
183 44 SUM
184 08 08
185 69 OP
186 39 39
187 25 CLR
188 76 LBL
189 48 EXC
190 63 EX*
191 08 08
192 69 OP
193 38 38
194 97 DSZ
195 09 09
196 48 EXC
197 75 -
198 43 RCL
199 05 05
200 95 =
201 55 ÷
202 01 1
203 00 0
204 00 0
205 00 0
206 00 0
207 95 =
208 50 I×I
209 44 SUM
210 01 01
211 71 SBR
212 55 ÷
213 72 ST*
214 08 08
215 55 ÷
216 01 1
217 00 0
218 00 0
219 00 0
220 00 0
221 95 =
222 44 SUM
223 02 02
224 76 LBL
225 44 SUM
226 97 DSZ
227 07 07
228 47 CMS
229 92 RTN
230 76 LBL
231 65 ×
232 71 SBR
233 23 LNX
234 32 X:T
235 53 (
236 00 0
237 85 +
238 93 .
239 03 3
240 77 GE
241 50 I×I
242 01 1
243 85 +
244 93 .
245 07 7
246 77 GE
247 50 I×I
248 01 1
249 85 +
250 76 LBL
251 50 I×I
252 00 0
253 54 )
254 29 CP
255 92 RTN
256 76 LBL
257 55 ÷
258 53 (
259 71 SBR
260 23 LNX
261 65 ×
262 06 6
263 85 +
264 93 .
265 05 5
266 54 )
267 92 RTN
268 76 LBL
269 86 STF
270 53 (
271 53 (
272 24 CE
273 75 -
274 05 5
275 42 STO
276 09 09
277 01 1
278 54 )
279 65 ×
280 05 5
281 85 +
282 01 1
283 00 0
284 54 )
285 42 STO
286 08 08
287 92 RTN
288 76 LBL
289 12 B
290 59 INT
291 99 PRT
292 82 HIR
293 08 08
294 82 HIR
295 07 07
296 43 RCL
297 03 03
298 82 HIR
299 37 37
300 32 X:T
301 43 RCL
302 00 00
303 47 CMS
304 82 HIR
305 37 37
306 42 STO
307 00 00
308 82 HIR
309 18 18
310 42 STO
311 05 05
312 32 X:T
313 42 STO
314 03 03
315 29 CP
316 82 HIR
317 17 17
318 67 EQ
319 35 1/X
320 98 ADV
321 76 LBL
322 60 DEG
323 71 SBR
324 33 X²
325 71 SBR
326 42 STO
327 43 RCL
328 05 05
329 66 PAU
330 97 DSZ
331 05 05
332 60 DEG
333 25 CLR
334 22 INV
335 90 LST
336 92 RTN
337 76 LBL
338 11 A
339 59 INT
340 42 STO
341 03 03
342 99 PRT
343 92 RTN
344 76 LBL
345 15 E
346 35 1/X
347 22 INV
348 59 INT
349 42 STO
350 00 00
351 92 RTN
352 76 LBL
353 35 1/X
354 00 0
355 35 1/X
356 92 RTN
```

LABELS.

```
001 23 LNX
040 33 X²
048 34 ΓX
057 39 COS
063 30 TAN
096 25 CLR
103 29 CP
108 24 CE
122 22 INV
152 28 LOG
157 38 SIN
160 42 STO
166 47 CMS
189 48 EXC
225 44 SUM
231 65 ×
251 50 I×I
257 55 ÷
269 86 STF
289 12 B
322 60 DEG
338 11 A
345 15 E
353 35 1/X
```

PROMPTING IN THE TI-88.- The new calculator is equipped with some powerful OP codes. Four of them are particularly interesting, as they allow us to write most of our "bread-and-butter" programs with nice prompting. As the prompting will be uniform from program to program, much less written documentation will be required to run these programs.

When the calculator is in that special PROMPTING STATE the top row of user-defined keys, A through E, will no longer function as such. Instead, they are used as answering keys marked YES, NO, UNK (unknown), ENT (enter) and CONT (continue).

The most powerful of the four prompting codes is OP 04, the ALL-RESPONSE CUE. In programming, this one has to be followed by four 2-digit numeric fields (no short-form here) of which each field will transfer program execution to one of four possible numeric labels, depending on user response and the pressing of one of the four prompting keys. Responding to the fifth one, CONT, simply will skip over the first four and program execution will continue there. If all that longwinded explanation confuses you a bit, lets do an example and see how OP 04 could be used in a real program.

Suppose you display the message NUMBER? and if the user answers YES, you display 1234. If the answer is NO, though, you display four alphanumeric characters, say ABCD. If, on the other hand, the user professes indecision by pressing UNK, you might display a mixture of digits and alpha characters, such as 1B3D. If the user wants to enter his own number, he just enters it and presses ENT. And finally, if the user decides "none of the above" and presses CONT, everything is bypassed and program execution simply continues, signalled here by displaying 0000. The program looks deceptively simple. And in fact it is very uncomplicated, although I would hate to program that sequence (and do all the overhead now supplied by one single OP code) on my 59:

LBL E CE CLR ALPH NUMBER? ALPH OP 04 01 02 03 04 APLH 0000 ALPH R/S LBL 01 ALPH 1234 ALPH R/S LBL 02 ALPH ABCD ALPH R/S LBL 03 ALPH 1B3D ALPH R/S LBL 04 STO A R/S

When you press E, program execution will be interrupted and the message NUMBER? displayed. If you answer YES, the program will branch to LBL 01. If you answer NO, the branch will be to LBL 02. If the answer is UNK, branching will be done to LBL 03. And if the user wants to enter any number and presses ENT, branching to LBL 04 will store the entered value in register A. And if the user presses CONT, program execution will continue, here a display of 0000.

The second OP code of interest here is OP 05, less powerful, but very handy. It is called the YES/NO REPSONSE CUE. The three others, UNK, ENT and CONT will be inactive now and only a YES or a NO response will do something. Suppose you sell tires, white walls and black walls. So, when you make up an envoice, a natural question to ask is WHITE WALLS?, because each class supposedly carries a different price. Now, when you answer YES, the calculator will GTO a segment containing pricing for white walls. Otherwise, it will simply fall through to the segment on black walls. As opposed to the OP 04 technique of branching à la 59, with OP 05 you need an INSTRUCTION BLOCK following it. If YES, the program will execute this instruction block, otherwise it will skip that block and continue program exection. An instruction block may be any valid instruction, such as SBR 00, GTO LBL 00, STO B, GTO 0134, or GBR 29 (go backwards 29 steps).

The third OP code of interest here is OP 06, the ENT/CONT RESPONSE CUE. The same programming requirement are used as in OP 05: if ENT is pressed, the first instruction block following OP 06 will be executed. Otherwise, if CONT is pressed, program exection will continue. Here, the designers obviously had an entry in mind. Thus, a sequence such as OP 06 STO A RCL A is a natural. If you enter a value it will be stored in register A. If CONT is pressed, the former value in A will be recalled and used in the subsequent calculation. Needless to say that in OP 06, the YES, NO and UNK keys are ignored.

The last OP code, OP 07 is less powerfull, but could have practical use. It is called the CONT RESPONSE CUE. All other keys are inactive and only a CONT repsonse will illicit program continuation. This is handy when you want to make sure that the user sees a particular message and signals receipt of it by pressing CONT. No branching takes places. Program execution simply continues after CONT is pressed.

When the program encounters any of the above mentioned OP codes, program execution stops temporarily and waits for a user response. If any function affecting the program counter, such as GTO, SBR, RST, or user-defined keys F through J (or old A' through E') is executed while the calculator is waiting in this special cue response state, this state is cancelled and the user-defined keys A through E are in effect again.

DEFINE (TI-88).- Imagine you had a two-keystroke function on the 59 that could do all of the following: 01 01 OP 04 RCL 00 OP 06 R/S LBL D STO 00 LBL E. Well, that is in a nutshell what Dfn N or Dfn rrr is on the TI-88.

When the program encounters Dfn N or Dfn rrr, program execution is temporarily suspended and the calculator waits for either a numeric entry followed by pressing ENT or no entry and pressing of CONT. In the expression Dfn N, the "N' is one of the 26 first data registers that may be addressed by means of a letter of the alphabet . In Dfn rrr, the "rrr" is any data register within the current partitioning.

As you can see from the 59 analogy, the calculator will stop with Dfn N= (or Dfn rrr=) on the left side and with the current value of N or rrr on the right side in the display. Entering a new value and pressing the ENT key (the D key in the 59) will store that value in N or rrr. Pressing CONT (the E key in the 59) will leave the present value in N or rrr and program execution will continue.

Consider this example: LBL E Dfn A Dfn B Dfn C Dfn D Dfn E 0 INV Lst R/S . When you start the program by pressing E, the display will show " A: 0." Enter a value and press ENT. Now the display will show "B: 0." Don't enter a value this time, but press CONT. Again the display will show "C: 0." Enter a value, and so on up to E. Then the calculator will do an INV LIST of all data registers, starting with 000 (=A). This will be printed as well as shown in the display. Stop the listing by pressing R/S.

This is another one of the step savers the TI-88 is loaded with. It also diminishes our dependence on the user-defined keys. By choosing letters that adequately represent the variables in the equation to be solved, one could write a program that almost doesn't need written documentation. All the prompting can be done in the display. P could stand for price, E for voltage, V for velocity, R for resistance, L for length, and so on.

TRUTH IN LENDING. In v6n9/10p22 we had such a program. Many members complained about the instructions going with it. They state that, without the printer, Payment is displayed. It is not. And, perusing the program, there is no provision for it either. Jorge Valencia in Lima, Peru, suggests the following modification of the User Instructions for off-printer use:

Press C to find the Payment Period. Then press R/S for the Sum of Payments. Press R/S again for the inflation effect on this one. Another R/S and out comes the sum of the interest portions. Another R/S again will show the effect of inflation on the last one. The next R/S gives the SUM of the principal and a last R/S shows the inflation effect on that one too.

The listing should be modified by inserting FIX 2 RCL 02 R/S between LBL C' and RCL 13. (steps 354 and following)

LADDER NETWORK ANALYSIS, the poor man's CAD (Computer Aided Design) Gerald W. Williams, in Microwaves, January 1981, pp 82-78. Mr. Williams is a member of the Tecnical Staff, Hughes Aircraft Co., Torrance Research Center, 3100 W. Lomita Blvd. Torrance, CA 90509, USA. The article describes how one can use a TI-59 program to analyze ladder networks over any range of frequencies. The calculator figures s-parameters input impedance, insertion loss, reflection coefficient, and VSWR. The 522-step calculator-only program is included. It requires the use of either the Master or the EE library module.
