52-NOTES

Volume 2 Number 3 48/39 March 1977

Newsletter of the SR-52 Users Club
published at
9459 Taylorsville Road
Dayton, OH 45424

RANDOM NUMBERS

Gerald Donnelly (203) brings up the topic of random numbers, and how best to generate them for particular applications. I should note at the onset that mathematicians are not even in agreement on a definition, let alone how best to generate them! However, for games applications with the SR-52/56 machines perhaps we can agree on the following admittedly fuzzy statement: The "best" random number generator will select numbers one at a time from a given set, producing a sequence whose ordering and distribution characteristics most closely resemble those produced by successive spins of an appropriately marked, unbiased roulette wheel. There are two general mechanization approaches: what I will term open, and closed. Open requires manual intervention via program interrupt; closed is self-contained, and produces sequences which are predetermined by the value of an initializing "seed" number. The open approach comes the closest to the stated concept of best, but imposes a chore on the user; the closed approach produces predictable sequences, but is automatic. The closed approach is used most often, and is generally satisfactory, as most users find it either distastefully ungentlepersonly or too much bother (or both) to determine in advance each predictable sequence. But perhaps the open approach deserves some attention, since it is easily mechanized in relatively few steps.

If random numbers in, say, the range 1-6 do not need to be hidden from the user, something along the lines of: *LBL A 5 = 1 = 4 = 6 = 3 = 2 = GTO A will work quite well: press A, then HLT. The longer the time between pressing A and HLT, the more "random" the selection is likely to be. If the numbers need to be hidden: *LBL A RCL 05 *EXC 03 *EXC 06 *EXC 01 *EXC 04 *EXC 02 STO 05 *pi sin *ifzro A 0 HLT might do, and is run with prestored values in Reg 01-06 by repeating: press A and switch from radian to degrees at an arbitrary time. A main program would use the contents of any one of Reg 01-06.

The closed approach poses two problems: how to keep generated numbers within a desired range, and how to get a "good" distribution, all in a minimum number of steps. The severity of the range problem depends on the required range. If the only restriction is that generated numbers fall between 0 and 1, the range problem is trivial. But a requirement, say, to produce a random ordering of the first ten primes might take such a chunk of memory space and/or execution time as to defy practical implementation. But let's look at something in between. As a "working" number, pi is a handy 13-digit, middle-magnitude, positive real whose digit values form a "randomly" distributed string of

The SR-52 Users Club is a non-profit loosely organized group of SR-52/56 owners/users who wish to get more out of their machines by exchanging ideas. Activity centers on a monthly newsletter, 52-NOTES edited and published by Richard C Vandorburgh in Dayton, Ohio. The SR-52 Users Club is neither sponsored nor officially sanctioned by Texas Instruments, Incorporated. Membership is open to any interested person: $6.00 includes six future issues of 52-NOTES; back issues start June 1976 @ $1.00 ea.

8 of the 10 numerals, and every closed random number generator I've seen for the personal programmables uses it in one way or another. A general approach is to combine pi with a seed by some succession of arithmetic operations, replace the seed with the fractional part of the result, and convert the most significant digit(s) of this result into an integer which may need to be scaled, depending upon range requirements. Each successive generation combines pi with the previously generated seed. Although I've seen square roots and fifth powers applied, a simple multiplication of pi by the seed seems to be quite satisfactory. As an example, let's look at Routine D' of Alan Charbonneau's Yahtzee program (V2N2p5). The requirement is to randomly select one of the numerals 1-6 each time the routine is called. The fractional part of the seed-times-pi product is saved, and then multiplied by the scaling factor: 5.5 which guarantees a real roundable to an integer in the 0-5 range. The + 1 offsets the range to 1-6. Although this scaling method works (one of my revisions to Alan's program), Steve Marum (188) points out that there is a bias favoring the numerals 2-6. His revision: 022: RCL 98 X 6 + .5 = ... removes the bias (caution: when making this substitution into Alan's program, either include a neutral spacer, or change all absolute addresses and reposition the unrelocatable code at steps 208-223).

Rather than go on with other requirements and approaches now, I'll wait for input from the membership. Send me your prize random number generators, hard-to-meet requirements, pedagogical arguments, etc and we'll resume discussion in a future issue of 52-NOTES.

A SUM OF THE DIGITS CHALLENGE

Jared Weinberger (221) has been experimenting with various approaches to summing the digits of a number. For the SR-52 the sequence: *LBL A (EE INV *fix EE 00 - INV *D.MS INV *D.MS * fix 0 EE SUM 69) INV *ifzro A *EXC 69 *rtn is Jared's fastest, while: *LBL B X 1 SUM 69 - EE div EE 00 = INV *ifzro B *EXC 69 *rtn is shorter but slower. Both require a one-time zeroing of Reg 69. Upon call, the displayed number's digits are summed, and the sum returned. For the SR-56, advantage can be taken of the Int function, and the routine starting at 00: (EE INV *fix EE 00 - *Int SUM 1) *x=t 16 RST *EXC 1 *rtn is both shorter and faster than either of the SR-52 ones. The routine: X 1 SUM 1 - EE div EE 00 = *x=t 15 RST *EXC 1 *rtn is one step shorter, but takes longer to run. To run these SR-56 routines, zero Reg 1 once along with the t register, then RST R/S with the number whose digits are to be summed in the display. Can anyone beat 21 SR-52 steps or 18 SR-56 steps?

PPX-52 GEMS AND REJECTS (52)

Stan Wagon (456) notes that program #390016: Extra Precision Factorials by one Greg Evans "... is interesting and might prove useful to anyone contemplating using double precision in other contexts."

E. S. (Mack) Maloney (246) has a great circle navigation program rejected by TI as a duplicate, but which calculates an initial course from the point of departure (which #949025 reportedly does not). Send Mack a SASE and 25¢ if you would like a copy of his program.

- - - - - - - - - -

BOOK REVIEW

STATISTICS FOR THE SR-52: KEYBOARD DATA, 188 pages $10.00; STATISTICS FOR THE SR-52: CARD DATA, 100 pages $11.00; and STATISTICS FOR THE SR-56, 79 pages $9.30; all by C Donaldson Ellis, published 1977 by Wilmardon Publishing Box 461 Storrs, CT 06268.

All three volumes are printed in a pica typewriter font, one side of 8½ x 11 paper, with direct copy of PC-100 printer listings opposite manually generated mnemonics for program listings. Each program begins with a brief description of what it does (assuming user familiarity with statistics usage), along with special requirements and limitations. User instructions are contained in a step-numbered Procedure section. This is followed by a table of register contents, and the program listing. Applicable formulas and algorithms are noted as being found in one or more of five references, but are not printed in the text; there are no sample problems. All programs are written for use with the PC-100, with notes covering required modifications for SR-52/56 use only.

The SR-52 keyboard manual covers 54 statistics programs, some requiring more than 224 steps. Input data are entered as required from the keyboard. The card manual arranges half of the keyboard programs to allow for data storage and retrieval via mag cards. Data packing and unpacking routines are given, but actual card storage is by program step only (no mention is made of direct storage to/from program registers. Register tables show data assignments to Reg 98, 99, 00, ... 19. I will not comment on the degree to which programs operate satisfactorily or have been optimized, except to note that a 2 X 2/ 3 X 3 matrix inversion program requires 445 steps, and one titled "Four-way ANOVA--Unweighted Means" requires 1736 steps (8 cards).

The SR-56 manual follows along the same lines, covering 40 programs, some of which require more than 100 steps. The longest program is "Mixed ANOVA 1 Within and 1 Between" which requires 242 steps (3 programs to key in).

It would appear that these manuals would be the most useful to statisticians who have access to SR-52/56 machines, but who don't know how (or don't care) to program them. Users are invited to communicate with the author via the publisher.

A SUBTLE DANGER IN USING iferr AS A FLAG (52)

Barbara Osofsky (420) notes a potential danger in using an undefined letter function to set an error condition flag in programs where program storage registers are used for data. In my version of her 5X5 program (V2N2p3,4) if any of the first 3 data entries begin with the sequence: 1346 or contain this sequence in the following patterns: XX1346, XXXX1346, or XXXXXX1346, then LBL C becomes defined (see V1N1p4) and the user call to C would cause undesired execution of the code (or data) following the LBL C. While the probability is small that LBL C would be unintentionally defined, the potential is there. As an example of a worst (albeit improbable) case, if the first 2 elements of a 5X5 matrix are: 8.594011346 and -1.234567895D64 respectively, when C is pressed, there is total wipeout! (see V1N2p2). As it turns out, Barbara was able to accomplish the conditional branch with a real flag in a recent version of her program which I may publish later. In the meantime, users of the V2N2 5X5 program who expect to input matrix elements containing more than 3 digits may wish to make the following

changes: re-write, beginning at step 192: STO 69 + = D HLT. (The HLT at step 199 may be either left alone, or deleted). Re-write step 6 of the user instructions to read: 6. Key 9n, press RUN, then E; jth column printed, j=2,3,4,5, see 1 displayed; repeat for n=6,7,8,9. Re-write note 3 to read: 3. At step 6, following the RUN, 5 is flashed.

It should be noted that any time program registers are used for data the possibility of unintentional labeling exists. However, if such registers are chosen at the high end (... 95, 96, 97) duplicate labels pose no problem since the machine always searches for labels starting at step 000.

A UTILITY PROGRAM FOR FRACTURED DIGITS (52)

In modern programming terminology, a utility program is one which aids the programmer in developing other programs. Routine A in V1N5p5 (top) is a short special purpose utility routine that helps the user synthesize look-up tables which become parts of other programs. As noted, once a main program has been written, Routine A is no longer needed, and it can be shelved until such time as it might be used to help in writing another program. A while back I attempted to arrive at the "best" display format for a game program I was working on by trying out various sequences of fractured digits mixed in with numerical results. I soon realized that it would help to have a fractured digits synthesizer program that could produce specified patterns in the SR-52 display, so they could be seen and judged as they actually appear. In the following program, the user specifies the character he wishes to appear at each of the 14 display positions (within the constraints described in V1N2). 0 produces the numeral 8 (which arbitrarily represents any desired numeral), 2 produces the " symbol, 3 produces a blank, 4 a ' symbol, 5 a ° symbol, and 6 a - symbol. Incidently, a 6 for position 11 followed by 3s for positions 12 and 13 will produce a - at position 11 and blanks at positions 12 and 13 (contrary to a statement in V1N2p5). As each position is processed, the number left in the display indicates the next position to be specified.

SR-52 Utility Program: Display Variations Synthesizer Ed

User Instructions:

1. Key position 0 code, press A; see 1 displayed
2. Key ith position code, press RUN; see i+1 displayed
   Repeat for i=1,2,...13
3. Press =, see synthesized display
4. To synthesize a new display, press CLR and go to step 1

Program Listing:

```
000: *LBL B = SUM 98 *LBL C 1 SUM 18 RCL 18 INV EE HLT *rtn *LBL A INV
022: *stflg 0 INV *stflg 1 - 3 = *ifzro 036 *stflg 0 1.11 STO 98
043: RCL 92 STO 19 1 STO 18 HLT div 1000 B div 1 EE 4 B div 1 EE 5
069: B div 1 EE 6 B div 1 EE 7 B div 1 EE 8 B div 1 EE 9 B div 1 EE 10
095: B div 1 EE 11 B = C - 3 = *ifzro 113 *stflg 1 C div 1 EE 12 B X
121: 10 = STO 18 10 y^x RCL 18= EE *fix0 *PROD 98 1 EE 88 INV *ifflg 1
147: 151 *1/x INV *ifflg 0 158 +/- *PROD 19 RCL 98 + STO 60 RCL 19
171: INV EE *fix 0 HLT 0 *ifpos *2' *2' *2' *2' *2' A
```

- - - - - - - -

AN AOS ADVANTAGE FOR A PROBLEM IN LOGIC (52)

As most of us know, AOS vs RPN arguments rarely end in conclusive victories for either side. But here is one application where AOS does appear to have a clear advantage. Logician Stan Wagon (456) found that the SR-52's nested parentheses could be made to order hierarchies in certain types of logical statements. In a branch of mathematical logic sometimes referred to as propositional calculus, 2-state variables which assume values of either true or false are operated on by the 5 logical operators: NOT, AND, OR, IF...THEN, and IF AND ONLY IF. All but NOT are "connectives", that is, they link two variables or paired parenthetically grouped variables. A string of variables, logical operators, and parentheses forms a logical statement or sentence. For each possible configuration of variable states, the entire statement is either true or false, and a tabulation of all such configurations is known as a Truth Table. (See Chapter 1 of the Boolean Algebra and Switching Circuits Schaum's Outline if you are new to the subject). Given the elementary truth tables:

NOT	
A	¬A
T	F
F	T

		AND	OR	IF... THEN	IF AND ONLY IF
A	B	A & B	A ∨ B	A → B	A ↔ B
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

Stan's Truth Tables program that follows accepts a statement of n variables (0 LT n LT 10) and up to 27 characters in length, and prints (or displays) the statement's truth or falsity for all 2^n possible ways to assign truth values to the n variables. Each statement is written in LRN mode as the body of Routine E, and must be contained within a parenthesis pair, 4 of the 5 logical operators are assigned to letter-functions as follows: A'=OR, B'=NOT, C'=IF...THEN, D'=IF AND ONLY IF; the "times" operator X=AND. Parentheses should be used extravegantly (when in doubt, use them). The 5 letter functions A, B, ... E correspond to the first 5 variables, For up to 4 more: RCL 06, RCL 07, RCL 08, RCL 09 would be written in Routine E to represent the additional variables. In order to adopt the output convention that 0=false and 1=true, Stan needed a way to vary the number of displayed digits, and worked out a dynamic-code-change method. A skeleton block of 8 instructions (steps 184-191) held permanently in Reg 93 is modified during program execution to supply the proper value for n in "fix n", then put into Reg 91, which supplies steps 168-175. A logical statement (S) that might be written out as: ((a&b)∨(c→¬b)) (a∨b) would be written in Routine E'as: *LBL*E' (((A X B) *A' (C *C' (*B'B))) *C' (A *A' B)) *rtn.

Each displayed line of the truth table shows the truth or falsity of the statement as an integer 1 or 0. Variable truths or falsities are shown to the right of the decimal. In the above example, the tenths place indicates the truth or falsity of variable a, the hundredths place b, and the thousandths c. The resulting truth tables are displayed as: 1.111, 1.110, 1.101, 1.100, 1.011, 1.010, 0.001, and 0.000, and translate to:

A	B	C	S
T	T	T	T
T	T	F	T
T	F	T	T
T	F	F	T
F	T	T	T
F	T	F	T
F	F	T	F
F	F	F	F

A printer version of this program would require error test code to terminate execution, in addition to replacement of the HLT at step 179 with a *prt. However, the user could save the extra steps by halting execution manually, and ignoring printout past the ?.

Corrections to the preceding page: two references to Routine E should have been to Routine E'. An IF...THEN arrow is missing in the written sample statement; the Routine E' code is correct.

SR-52 Program: Truth Tables — Stan Wagon (456)

User Instructions:

1. Write the statement at label E': GTO *E' LRN (statement) *rtn LRN
2. Key number (n) of variables 0 LT n LT 10, press GTO GTO
3. Press RUN, see ith line of Truth Table
   Repeat for i=1,2,...2^n; the 2^nth line is flashed.
   For new statement, go to step 1.

Program Listing:

```
000: *LBL A RCL 01 *rtn *LBL B RCL 02 *rtn *LBL C RCL 03 *rtn
018: *LBL *A' + (STO + 1) X *rtn *LBL *B' 1 + *rtn *LBL *C' + 1 +
038: (STO - 1) X *rtn *LBL *D' + 1 + *rtn  *LBL D RCL 04 *rtn
057: *LBL E RCL 05 *rtn *LBL GTO *CMs STO 18 +/- EE 70 + RCL 93 =
078: STO 91 2 y^x RCL 18 = EE STO 19 *fix 8 1 INV SUM 19 RCL 18 *EXC 00
104: STO 17 RCL 19 STO 16 *1/x 1 SUM 17 (RCL 16 - 2 y^x (RCL 00 - 1))
133: EE *ifpos 143 0 GTO 154 STO 16 RCL 17 +/- INV *log + 1 *IND STO 17
158: *dsz 114 (*E' div 2 - INV
176: 2 +/- = HLT GTO 091 EE *fix 0 *D.MS *fix 0) X *LBL *E'
```

- - - - - - - - -

TI NOTES

SR-52 and SR-52A Trouble Shooting Guide: reported by Marshall Williams (227) and later confirmed by TI as available to the public. Send $11.00 plus local sales tax to TI Box 53 Lubbock, TX 79408. This manual reportedly contains schematics and I/O interface information.

PC-100 Modifications: to upgrade to PC-100A reported by T S Cox (9) and later confirmed by TI as a possibility. If implemented, TI would modify PC-100s for an as yet to be determined fee to accomodate "other" machines, but such modifications would not include the battery charger feature of the PC-100A.

ROUTINES

An Efficient Input Routine (52): Ed Parsons (65) has found a way to save 2 steps for a routine that sequentially stores input data and displays a data counter. The usual approach is along the lines of: *LBL A *IND STO 69 1 SUM 69 RCL 69 HLT. Ed replaces the RCL 69 with +, which works just as well, provided processing begins with CLR or =.

A Short Integer Test: Stan Wagon (456) suggests the sequence: ... - *D.MS = *ifzro ... as a quick test to determine whether or not a number is an integer. However, one limitation is that the number must be less than 13 digits for older machines (see V2N2p6). A comparable SR-56 sequence would be: ... - *INT = *x=t, with a zeroed t register.