The Midnite Software Gazette was published quarterly from September 1980 to July 1982 by members of the Central Illinois PET Users Group, and paid for by donations and ads. It went to all who provided stamped self-addressed envelopes. Beginning with this issue, the Midnite also goes to subscribers of The PAPER, one of the first PET resources, adopting their subcription policies. This means:

* The Midnite/PAPER is published bi-monthly by Midnite Software, Inc.
* Subscriptions cost $20 (US) via first class mail in the U.S., $25 (CDN) in Canada, $30 by surface mail overseas, and $40 via air mail overseas.
* Pre-paid camera-ready advertisements now cost $100 per page.
* Orders of 6+ copies to one address are welcome, at one-half cost.
* The Midnite/PAPER now also accepts articles, as well as reviews.
* The Midnite/PAPER has the large format and type of The PAPER.

****

TABLE OF CONTENTS

MIDNITE SOFTWARE GAZETTE

THE PAPER

****

A Variable-Length-String Shuffler in RPL

by Tim Stryker

The article "Card Shuffle" by Bill Batcher in Vol.IV, No. 1 contained an interesting card-shuffling method attributed to Don Coscia. Coscia's method involves taking a string consisting of all of the card names, and pulling randomly placed cards out of this string, compressing the string in the process, until all of the cards are gone. I thought that was pretty nifty, because the other two shuffling methods that Batcher presented had definite problems: the first could (in principle) generate a truly random ordering but would take forever to execute, whereas the second could never be said to generate a truly random ordering, and would take quite some time to run even so. Coscia's method is easily seen to be capable of generating a truly random card ordering, and to run much faster than the other two. The only problem with his method is that it is incapable of handling character strings of variable length.

Thinking about this, I saw that all that would be needed to remove this restriction would be to apply his method to a set of fixed length pointers to strings, rather than the strings themselves. BASIC, however, makes dealing with pointers very difficult (in some instances, pointers can be simulated through array indices, but in this case would be clumsy at best). So, the question became: how best to implement Coscia's algorithm using pointers? Not in BASIC, certainly. Assembly language? Ugh. Pascal? I don't have it (who does, at 175-300 bucks a pop?), and even if I did, I wonder whether it is really set up for this kind of thing. Actually, since I have recently begun marketing a new stack-oriented language of my own called RPL, the conclusion was somewhat foregone. As it happens, RPL is specifically set up to facilitate pointer manipulation of all kinds. And it also happens to incorporate a very nice little primitive called "rotate" (courtesy of Ken Wasserman) which is ideal for the purpose of extracting just one element from a list and compressing it out of the list when done.

The accompanying listing shows an RPL program for shuffling and printing out the English-language names of the 52 usual playing cards. It times itself, finding that it takes just 1.36 seconds to shuffle and print out the whole deck. All but 210 milliseconds of this are taken by the printing operation itself (change what follows the dollar-sign on line 200 to ". NEXT . RETURN" to do the shuffle without printing the results and the process says "FINISHED IN 0.21 SECONDS"). This should be fast enough for most purposes.

Since most of you are not conversant with RPL, the listing will probably not make much sense to you. In general terms, the program mainline consists of nothing more than zeroing the PET jiffy timer ("0 142 !), calling the shuffle routine ("SHUFFLE &"), and reporting how long it took to execute (the rest of lines 110 and 120). The shuffle routine essentially just pushes the addresses of the 52 card-name strings onto its stack (line 190), and then proceeds to apply Coscia's algorithm to that list (lines 200 and 210). The heart of the shuffling process takes place in line 200 with the sequence "RND ; FN - (backslash) 2 + $", which extracts a random one of the pointers from the set remaining on the stack at that point. This sequence is enclosed between the first FOR in line 200 and the NEXT in line 210, which cause it to be executed just 52 times, each time extracting its new random card from a "pool" on the stack with one less card in it.

The above, of course, is hardly an "explanation" of how the program works. Hopefully, your appetite has been merely whetted and you now want to find out more about RPL. It really is a great language: you can get it to do a lot with very few keystrokes, but it doesn't cost you anything in execution time or object space to put in voluminous amounts of comments. It uses the ordinary PET screen editor for program entry, and it compiles very quickly. It can come with an integral "symbolic debugger", if you like, which makes program debugging a

breeze. It generates object code that is exceedingly fast and small, it's very easy to learn and use...and the list goes on. Drop me a line c/o Samurai Software, 14650 Bull Runn Rd 126, Miami Lakes FL 33014 if you are at all curious about it.

```
10 ********************************************************************************
20 *                                                                              *
30 *          VARIABLE-LENGTH-STRING SHUFFLING PROGRAM                            *
40 *      USING THE METHOD OF RANDON-DEPTH STACK ROTATIONS                        *
50 *  THE STACK ENTRIES BEING POINTERS TO THE STRINGS THEMSELVES                  *
60 *                                                                              *
70 *    BY TIM STRYKER 12/81 (WITH APPRECIATION TO DON COSCIA)                    *
80 *                                                                              *
90 ********************************************************************************
100 REM
110 TESTSHUF: 0 142 ! STRINGSEQ SHUFFLE & 142 @ INT "FINISHED IN " PRINT #
120 60 / STR$ PRINT "." PRINT 60 \ 5 * 3 / STR$ PRINT " SECONDS" PRINT STOP
130 REM
140    SHUFFLE IS THE SHUFFLE ROUTINE ITSELF. IT EXPECTS TO BE PASSED THE
150    ADDRESS OF THE ZERO-TERMINATED SEQUENCE OF STRINGS TO BE SHUFFLED,
160    AND IT THEN PRINTS OUT THESE STRINGS IN RANDOM ORDER, WITH COMMAS
170    BETWEEN THEM. IT POPS THE ADDRESS PASSED TO IT BEFORE RETURNING.
180 REM
190 SHUFFLE: 0 SETUP: 1 + ; PEEK IF ; # PEEK + 1 + % SETUP GOTO END
200 % . # 1 - 1 FOR RND ; FN - \ 2 + $ # PEEK 1 FOR # FN + PEEK 1 PRINT NEXT
210 . # FN - 1 > IF ", " THEN 13 1 END PRINT NEXT . RETURN
220 REM
230    STRINGSEQ IS THE SEQUENCE OF VARIABLE LENGTH STRINGS TO BE SHUFFLED,
240    IN THIS CASE THE NAMES OF THE CARDS IN A STANDARD PLAYING DECK.
250    THIS LIST MUST BE TERMINATED BY A ZERO.
260 REM
270 STRINGSEQ:("ACE OF SPADES" "KING OF SPADES" "QUEEN OF SPADES"
280 "JACK OF SPADES" "TEN OF SPADES" "NINE OF SPADES" "EIGHT OF SPADES"
290 "SEVEN OF SPADES" "SIX OF SPADES" "FIVE OF SPADES" "FOUR OF SPADES"
300 "THREE OF SPADES" "TWO OF SPADES"
310 "ACE OF HEARTS" "KING OF HEARTS" "QUEEN OF HEARTS"
320 "JACK OF HEARTS" "TEN OF HEARTS" "NINE OF HEARTS" "EIGHT OF HEARTS"
330 "SEVEN OF HEARTS" "SIX OF HEARTS" "FIVE OF HEARTS" "FOUR OF HEARTS"
340 "THREE OF HEARTS" "TWO OF HEARTS"
350 "ACE OF DIAMONDS" "KING OF DIAMONDS" "QUEEN OF DIAMONDS"
360 "JACK OF DIAMONDS" "TEN OF DIAMONDS" "NINE OF DIAMONDS"
370 "EIGHT OF DIAMONDS" "SEVEN OF DIAMONDS" "SIX OF DIAMONDS"
380 "FIVE OF DIAMONDS" "FOUR OF DIAMONDS" "THREE OF DIAMONDS" "TWO OF DIAMONDS"
390 "ACE OF CLUBS" "KING OF CLUBS" "QUEEN OF CLUBS"
400 "JACK OF CLUBS" "TEN OF CLUBS" "NINE OF CLUBS" "EIGHT OF CLUBS"
410 "SEVEN OF CLUBS" "SIX OF CLUBS" "FIVE OF CLUBS" "FOUR OF CLUBS"
420 "THREE OF CLUBS" "TWO OF CLUBS")
430 (0)
```

A Random Bit Generator

by Jim Fowler

Do you have any use for a really fast machine language routine that generates pseudo-random bits? It's not as good as a pseudo-random number generator, since it only outputs a bit at a time, but the bits are arranged in a sequence, so no pattern of bits predominates. That means 100 appears the same number of times as 101, 110, 111, and so forth. Eventually the sequence repeats. One nice feature is that you can assure the repeat length of the sequence is over some minimum value you select in advance. The longer the repeat length, the longer the routine, but even if the repeat length is long the routine is still quite short. I use a program with two constants, A and B. The values you choose for these will determine the repeat length up to 65,535 bits. If you want a longer sequence, let me know. I have the data to write one that will only repeat after more than an octillian bits!! Incidentally, there is a rigorous mathematical proof of the 'randomness' of these bits. The data for the constants (A and B below, but for longer sequences you may need more constants) comes from an article by E. J. Watson, "Primitive Polynomials (Mod 2)" in Mathematics of Computation, vol. 16, pp 368-369, 1962.

The program has a two byte shift register (X1,X2) you can locate anywhere convenient. You must seed this with a non-zero number. Then the sequence will be random once the leftmost bits are transformed. (This could take up to 9 runs of the routine). From that point on, it leaves the carry bit SET or CLEAR in a 'random' way. The accompanying table gives A and B values for the various repeat lengths, up to the maximum in a two byte register. Thus, seed the register, JSR to the routine and use the carry bit as random output.

```
033A             ; RANDOM BIT GENERATOR
033A             ; LEAVES CARRY FLAG 'SET' IN
033A             ; A RANDOM WAY AFTER EACH CALL
033A             ; SYS 916 FOR EACH USE
033A             ;
033A                       ORG $0392
0392             ;
0392             SHIFT1    BYT $05     ; SET UP TWO BYTE
0393             SHIFT2    BYT $22     ;  SHIFT REGISTER
0394  18                   CLC         ; CLEAR CARRY EACH TIME
0395  0E 93 03             ASL SHIFT2  ; SHIFT BYTE 2 OF REGISTER LEFT
0398  2E 92 03             ROL SHIFT1  ; ROTATE BYTE 1 OF REGISTER LEFT
039B  90 10                BCC END     ; IF CARRY CLEAR, BRANCH TO END
039D  AD 92 03             LDA SHIFT1  ; LOAD FROM REGISTER BYTE 1
03A0  49 08                EOR #$08    ; EXCLUSIVE OR W/ A FROM TABLE
03A2  8D 92 03             STA SHIFT1  ; STORE ACC AT REGISTER BYTE 1
03A5  AD 93 03             LDA SHIFT2  ; LOAD FROM REGISTER BYTE 2
03A8  49 80                EOR #$80    ; EXCLUSIVE OR W/ B FROM TABLE
03AA  8D 93 03             STA SHIFT2  ; STORE ACC AT REGISTER BYTE 2
03AD  60         END       RTS         ; RETURN TO CALLER
```

TABLE

Rep't length	A	B	Rep't length	A	B
255	1D	0	8191	0	D8
511	8	80	16383	0	AC
1023	2	40	32767	0	06
2047	0	A0	65535	0	2D
4095	5	30			