DO WE NEED A 5 LEVEL STACK?

The article bridging pages 2 and 3 of V4N9 presents a Y^X type problem and suggests it could be "easily done with minimal keystrokes, operators, if the stack were five high instead of four." The problem is:

$$2^{1.8^{1.6^{1.4^{1.2}}}}$$

The solution given for a four level stack was:

1.4 E↑ 1.2 Y^X 1.6 X↔Y Y^X 1.8 X↔Y Y^X 2 X↔Y Y^X

a 13 step solution counting number entries as a single step each. With a five level stack the following sequence would seem to be the most efficient:

2 E↑ 1.8 E↑ 1.6 E↑ 1.4 E↑ 1.2 Y^X Y^X Y^X Y^X

but this is also a 13 step solution and thus the five level stack gave no advantage over the four level stack.

Other keystroke solutions could be used for a four level stack, For example:

1.8 E↑ 1.6E↑ 1.4 E↑ 1.2 Y^X Y^X Y^X 2 X↔Y Y^X

1.4 E↑ 1.2 Y^X 2E↑ 1.8E↑ 1.6 R↑ Y^X Y^X Y^X

but these are also 13 step solutions and offer no advantage.

I believe the original (first above) is the best of the various solutions given to the problem as the method works no matter how many exponents are presented. The remaining solutions require careful analysis of stack capacity to avoid losing data out the top of the stack, and/or to roll the stack at an appropriate time.

No solution was seen, using a five level stack, which would have fewer than 13 steps and therefore I do not believe that this problem presents a good case for a five level stack.

However, I have personally wished for a five level stack. But if we had a five level stack someone would want six; and now that we have 224 steps of program and 26 registers who has not wished for more? Where should it stop?

The Pseudo Five Level Stack: Problem and Solution

In some programs I enter data into the stack and start the program running. The program operates on the stack data and/or stores it in registers. In one program I wanted to be able to enter five variables; but at first was limited by the four level stack. The solution was quite obvious, once discovered. It was to use the Last X register. Thus the five variables were entered as follows: (where V_1 stands for first variable etc).
CLX V_1 + (puts V_1 into Last X register) V_2 E↑ V_3 E↑ V_4 E↑ V_5 and then start the program running. Of course the program must be designed to use the stack data and recover the data in Last X before any data is lost from the stack or from the Last X by other manipulations or calculations. Obviously the data must be entered in strict sequence.

One may wonder why the data was not directly entered into registers. In my case the reason is compound. No available registers and/or the data is to be operated on and/or combined with other data already in the register and/or combined with data in more than one register. The whole thing would be just a little easier with a five level stack.

Has anyone faced this same problem with six variables? I would sure like to know what to do as I expect to face the problem eventually.

G W Killian 2038 R/S

SAMPLING WITHOUT REPLACEMENT

The ever increasing popularity of games and new uses for simulation sooner or later bring the programmer up against the need for a generator which draws numbers at random without replacement. The BINGO program by Mike Richter (V4N7P9) is one such example. After studying this particular program, however, one can only hope for a quick winner at 40 seconds per draw. The purpose of this article, though, is not to criticize an otherwise good program but to look at some of the different ways a routine can be written to select numbers at random without replacement. Some of the more obvious uses for such a routine include Bingo, Bridge hands, Bagels, Keno, Reverse, and divers card games such as Poker and Blackjack where true odds deals are desired.

The phrase "random draw without replacement" refers to a process whereby all of the elements in a given population at a given time are equally likely to be chosen on the next draw; once drawn, an element cannot be picked again. Although several possible schemes may suggest themselves, they will usually stand classification into one of four categories. It is only natural, however, that some techniques are better suited to one particular application or calculator than others and a clear understanding of the differences will permit a wise selection.

1. SIMPLE ELIMINATION. The first step in this method is to store all of the numbers in a list. A number is selected at random from the list and checked to determine if it has been used previously. If the number has been used, another is selected at random until a useable value is found. Once used, the number is flagged in some manner to prevent its further use. This can be done by changing its sign or setting the value equal to zero.

The major disadvantage of this technique is that once 90% of the values are used, there's only one chance in ten of getting a valid number on the subsequent tries. With ten items in the list this may not be particularly bothersome although half of the time you will require 27 or more attempts to exhaust the population. With 52 cards, however, the average number of draws required to exhaust the deck is 236. If a smaller subset of the population is required rather than the entire sample space, i.e. as in a Poker hand, the technique suffices well. A distinct advantage of this scheme is that flags can be used to indicate the status of each number in lieu of storing the actual numbers themselves. Initially all flags are set and then zeroed out one at a time as the corresponding number is drawn. An excellent discussion of the flag technique is presented in V3N9P11 and V4N1P18 under Jake's flags. Note that no more than 30 flags (0 to 29) can be stored in a register using the decimal coded binary (DCB) method described. Even with this limitation, the HP-67/97 can be programmed to draw from up to 750 numbers without replacement.

2. RANK ORDER. This scheme associates a random real number with each integer in the population. The program finds the largest random number and uses the integer associated with it. The number is then flagged so that it will not be used again. The next largest real number in the list is located for the next draw. Several variations of this technique are possible and include: Sorting the numbers at the beginning of the run to save time later, decreasing the list size after each draw to reduce the number of comparisons required to find the largest real, and assigning a new set of reals to the remaining integers on each draw.

The principal drawback of this technique is the number of comparisons required to sort out the largest number. As described, the Rank Order method requires (n-1)·(n-1) comparisons to exhaust the population if "n" is the number of elements in the population. Even if the list is shortened after each draw, it would still require (n)(n-1)/2 comparisons to exhaust the population (about half as many). Various types of sorts are available, however, which could minimize the time required to reorder the list if this approach is desired, cf. V4N8P8, 11 and 24.

3. LINEAR SEARCH. This technique was used by Mike in the BINGO program. A random number less than or equal to the number of elements remaining in the population is computed at the beginning of each draw. Starting at the top of the list, the program checks off the unused numbers until the count equals the random number. The last unused number checked is the one drawn. The number is flagged so that it connot be counted or drawn again and the size of the population is reduced by one.

With this particular method, about half of the numbers must be checked on each draw. The total number of checks required to exhaust the population is (n)(n+1)/2. An advantage, however, is that flags can be used to indicate what numbers have been used and this reduces the number of registers required. Mike uses a kind of binary coded decimal (BCD) scheme to keep track of the unused numbers that would be good up to about 250 numbers. Jake's flags could be used to extend the program to about 750 numbers.

It would be instructive to see what would happen if this technique were modified by eliminating the counting procedure in order to increase the speed. Rather than start at the top of the list each time a number is drawn, enter at a random point and search downward until an unused number is found. The choice of a random entry point assures that all of the remaining numbers are equally likely to be drawn. It would also be necessary to start again at the top of the list if no unused numbers are found below the random entry point. While a considerable number of checks are saved with this method, the process does not produce a random sample. Consider a situation where there are two numbers left--one at the top of the list and one at the bottom. The probability that the next number drawn is the bottom number is nine times as likely as the top number, assuming there were ten numbers in the list. Indeed the program would tend to produce the numbers in a preferred order. The point here is that one must be extremely careful in designing a program to avoid "stacking the deck."

4. STACK METHOD. In this system, all of the numbers are stored in a list and one is selected at random. The number at the top of the list is pulled off and stored in the place of the one selected. The list size is decreased by one to reflect

the current position of the top of the stack.

The greatest disadvantage of this scheme is that it is limited to about 100 numbers on the HP-67/97 since flags are not suitable for storing the numbers. This is because the numbers are continuously changing position in the list. The overriding advantage over other methods is the short execution time which does not vary from one draw to the next.

All of the techniques described have one feature in common: they incorporate a random number generator (RNG) of some type for the selection process. The ultimate guarantee of randomness in whichever method is used will be solely determined by the RNG routine employed. For most purposes, the linear congruential method should be adequate, cf. V4N8P1 through 6.

Of the different methods available, the one that is perhaps most useful as a general purpose routine is the Stack method. Such a program appears in this issue and may be used in any situation requiring up to 100 numbers in the population. A flow diagram in Figure 1 conveys the general implementation on the HP-67/97. The actual implementation, however, embodies a number of considerations which will be described in order for the user to improve the present version or tailor it to a specific problem. The most time consuming part of the program is the initialization which requires loading 100 numbers into storage, five to a register. Five of the six remaining registers are used for housekeeping. The loading is accomplished by storing 0.95 96 97 98 99 in the top secondary register. The constant 0.0505050505 is repeatedly subtracted from this number and the results stored in consecutive registers. If the highest number to be drawn is 99, the last two digits of R19 represent the top of the list. With fewer numbers, more registers could be freed and initialization might be simplified.

At the beginning of each draw, the number at the top of the list is placed in temporary storage, R_A. This number is extracted by using one less than the total remaining in the list and dividing by 5, e.g., (52-1)/5 = 10.2. Thus the number at the top of the list when there are 52 items left will be found in R10 after the 2nd digit. Multiply R10 by 10^2 and take the fractional part; multiply this by 100 and take the integer part. You now have the top number in the list. Randomly select a number less than 52 and extract the number at this position in the list in a similar fashion. Replace the extracted number with the contents of R_A. This is accomplished by subtracting the extracted number from the contents of R_A, shifting it an appropriate number of places to the right (four in this case) and algebraically adding this result to R10. This process cancels the number drawn and adds the number from the top of the list into the same position. The list size is decremented by one before another number is drawn.

The program does not terminate automatically after the population has been exhausted. Instead, the final number is repeated once and subsequent draws will cause a zero to be displayed. If DSZ were used to overcome this feature, the final number could be obtained by recalling R0 and multiplying by 100. The population size is easily adjusted by specifying the position of the top of the list before the first draw. Numbers above this point are not used. W.M.KOLB (265)

SAMPLING WITHOUT REPLACEMENT— USER INSTRUCTIONS

KEY A - N
KEY B - SEED
(STO B)

	INSTRUCTIONS	INPUT KEYS		OUTPUT
1	Load Program			
2	Enter a random number seed (decimal followed by 8 of 9 digits)	seed	STO B	
3	Enter number of elements in the population. The display will show the first random number drawn.	N	A	R.N.
4	Press R/S to obtain successive random numbers without replacement.		R/S	R.N.

Initialization requires about 15 seconds. Each draw takes about 4 seconds.

PROGRAM

001	*LBLA	21 11	013	*	-24	025	9	09	037	*LBL0	21 00
002	EEX	-23	014	2	02	026	7	07	038	RCLA	36 11
003	-	-45	015	+	-24	027	9	09	039	GSB1	23 01
004	STOA	35 11	016	CMS	-22	028	8	08	040	RCLB	36 12
005	1	01	017	ENT↑	-21	029	9	09	041	9	09
006	9	09	018	ENT↑	-21	030	9	09	042	9	09
007	STOI	35 46	019	ENT↑	-21	031	*LBLa	21 16 11	043	7	07
008	EEX	-23	020	.	-62	032	STOi	35 45	044	x	-35
009	1	01	021	9	09	033	+	-55	045	FRC	16 44
010	ENT↑	-21	022	5	05	034	BGZI	16 25 46	046	STO6	35 12
011	9	09	023	9	09	035	GTOa	22 16 11	047	RCLA	36 11
012	9	09	024	6	06	036	STO0	35 00	048	EEX	-23
									049	+	-55
									050	x	-35
									051	INT	16 34
									052	GSB1	23 01
									053	STOC	35 13
			054	-	-45	070	+	-24			
			055	RCLD	36 14	071	STOI	35 46			
			056	+	-24	072	FRC	16 44			
			057	EEX	-23	073	EEX	-23			
			058	2	02	074	1	01			
			059	+	-24	075	x	-35			
			060	ST+i	35-55 45	076	10^x	16 33			
			061	RCLA	36 11	077	STOD	35 14			
			062	EEX	-23	078	RCLi	36 45			
			063	-	-45	079	x	-35			
			064	STOA	35 11	080	FRC	16 44			
			065	RCLC	36 13	081	EEX	-23			
			066	PRTX	-14	082	2	02			
			067	GTO0	22 00	083	x	-35			
			068	*LBL1	21 01	084	INT	16 34			
			069	5	05	085	RTN	24			

Ed. Note. A seed of .123456789 was used and 100 used as the "N". Step 66 was replaced with PRINT X and the resultant list checked for omissions or duplicates. None were found. The distribution of the numbers 0 to 99 looked good. The results are shown below. After 15.5 seconds the first number printed with the last number finishing 7 minutes 53.5 seconds.

6.00	88.00	56.00	64.00	38.00	69.00	61.00	2.00	19.00	25.00
15.00	20.00	9.00	21.00	11.00	43.00	7.00	4.00	16.00	71.00
84.00	85.00	35.00	17.00	41.00	94.00	82.00	74.00	56.00	59.00
55.00	75.00	27.00	32.00	99.00	23.00	5.00	95.00	93.00	91.00
18.00	49.00	12.00	97.00	6.00	47.00	98.00	28.00	52.00	1.00
50.00	76.00	87.00	22.00	46.00	13.00	31.00	8.00	54.00	83.00
63.00	70.00	14.00	40.00	26.00	36.00	96.00	80.00	67.00	24.00
29.00	57.00	78.00	60.00	30.00	42.00	92.00	66.00	45.00	89.00
33.00	73.00	62.00	10.00	68.00	79.00	44.00	65.00	51.00	77.00
66.00	48.00	34.00	90.00	3.00	72.00	37.00	53.00	39.00	81.00

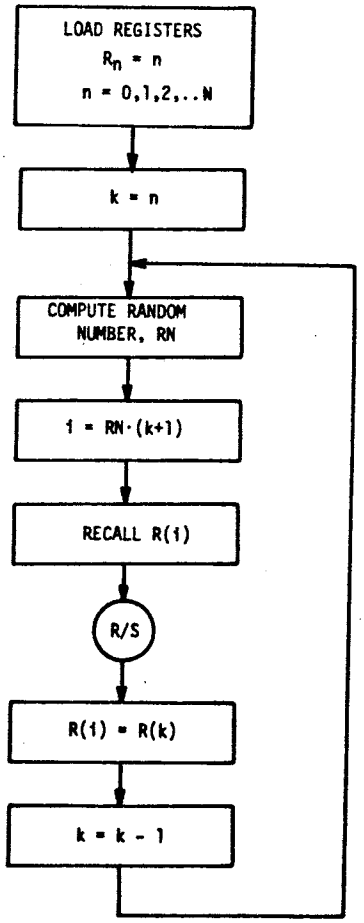

Figure 1. Sampling Without Replacement

R/S