```
 10 REM CHARLES A STEWART
 20 REM 3033 MARVIN DR.
 30 REM ADRIAN MI 49221
 40 REM 517-265-4798
 50 REM AUTOLOAD PROGRAM FOR OSI C1P
 80 POKE133,0:POKE134,8
 90 DIMA$(24),B$(24):POKE15,0
 100 FORX=0TO40:PRINT:NEXT:PRINT" AUTO L
OAD OF MACHINE LANG PROG"
 110 PRINT" IN PAGE 2 OR MEMORY LOC ABOV
E $0800
 120 PRINT"PROGRAM REQUIRES 2047 BYTES T
O OPERATE
 150 FORX=1TO10:PRINT:NEXT
 155 INPUT"START,END ADDRESS IN DECIMAL"
;A,B
 170 PRINT:PRINT:INPUT"SOURCE PROGRAM LI
NE # START & INC";D,E
 175 POKE15,255:SAVE
 180 PRINTD;"FORX="A"TO"B":READY:POKEX,Y
:NEXT"
 210 FORI=ATOBSTEP23:FORJ=0TO22
 220 A$(J)=STR$(PEEK(I+J))
 230 A$(J)=RIGHT$(A$(J),LEN(A$(J))-1):NE
XTJ
 240 D=D+E:PRINTD;"DATA";
 250 FORJ=0TO11:IFI+J>BTHEN280
 251 PRINTA$(J);CHR$(44);:NEXT:PRINTA$(J
);
 260 D=D+E:PRINTD;"DATA";
 270 FORJ=13TO21:IFI+J>BTHEN280
 271 PRINTA$(J);CHR$(44);:NEXT:PRINTA$(J
);
 275 NEXTI
 280 PRINT:PRINT"POKE515,1:RUN"
 290 POKE517,0
```

Program Listing

```
START,END ADDRESS IN DECIMAL? 0,222

SOURCE PROGRAM LINE # START & INC? 100,1
0
 100 FORX= 0 TO 222 :READY:POKEX,Y:NEXT
 110 DATA76,116,162,76,195,168,5,174,193
,175,76,136,174
 120 DATA0,0,255,56,17,0,49,48,48,44
 130 DATA49,48,0,0,69,0,49,55,44,48,32,0
,78
 140 DATA34,0,75,0,53,49,53,44,49,58
 150 DATA82,85,78,34,0,82,73,78,84,65,36
,40,74
 160 DATA41,58,32,0,84,34,32,0,82,84
 170 DATA32,38,32,73,78,67,34,59,68,44,6
9,32,0
 180 DATA71,34,0,177,128,128,11,96,171,3
4
 190 DATA58,0,0,0,0,0,0,0,0,104,101,0,1
 200 DATA249,6,165,143,174,225,141,32,8,
6
 210 DATA247,1,32,25,0,251,1,3,226,5,12,
6,226
 220 DATA6,115,7,106,7,0,8,220,0,155
 230 DATA0,236,4,164,237,0,3,25,0,74,0,8
,6
 240 DATA71,6,255,164,0,83,0,104,0,4
 250 DATA76,30,180,19,6,227,5,0,0,6,6,13
6,0
 260 DATA0,175,33,0,0,136,161,0,0,33
 270 DATA56,0,8,0,230,195,208,2,230,196,
173,255,4
 280 DATA201,58,176,10,201,32,240,239,56
,233
 290 DATA48,56,233,208,96,128,79,199,82,
47,140,164,171
 300 DATA5,229,231,
POKE515,1:RUN
```


Example 1

Part One Of Two
OSI C1P Newspaper Route Listing Program

Charles L. Stanford
Cinnaminson, NJ

This program, like most, started out as a very simple task to fulfill a stated need. And like too many, it got very, very complicated. My son, John, has a paper route. In a big city suburb, newspaper routes are very volatile; the customer list changes as the promotions of the various papers attract readers, and as the residents move on with their corporations. So the route list is hard to keep current. Each day off requires a new hand-written list for the sub (too often Dad). A Paper Route program seemed like a natural. And the program was easy to write. It started out in much the same form as listed here. The data save method is similar to the one in **COMPUTE!**, Issue 2, "Home Accounting" article, with the exception that I added $trings for the customer's names. All seemed to be fine. But then the bug showed up. The program wouldn't save $trings to data statements when new customers were added! Everybody ended up with the same name.

A week (and a lot of POKEing around in RAM) later, I knew one heck of a lot more about my C1P's method of storing variable arrays, and the program ran. I think that a quick review of what I learned, and how the computer can be "fooled" by some $tring manipulation tricks, will be useful to many readers.

Microsoft BASIC Source Code Storage

Much has been written on the method Microsoft

BASIC uses to store programs. I think one of the best explanations if found in Edward H. Carlson's book "OSI BASIC In ROM". To simplify, the source code is stored in RAM starting at Hex address $0300. The first byte is 00. The next two hold the address of the next line, in the standard notation of lo byte first, hi byte second. To convert to decimal, multiply the decimal value of the second byte by #256 and add the value of the first. The next two bytes are the line number, in the same form. For example, line 100 would read 64 00 (the Hex value of Dec 100 followed by 0 * 256). Now comes the line itself, with the BASIC commands in their token form and all other information represented by its ASCII value. See Table 1 for the representation of a typical line.

Each successive line is ended by a 00, and each new line starts as above. The last line is followed by three bytes of 00. Next comes the variable table, with the simple variables stored first. The numeric variables are stored in four-byte floating point binary. I won't go into that here, except to say that a decimal number is represented in a manner similar to a logorithm, with the characteristic (exponent) first and the mantissa (base value) next. The $tring variables are stored in a much different manner. The second byte of a $tring variable is the ASCII value of the second character of the variable plus $80 (Dec 128). Where the next four bytes of a numeric variable are the value for the numeric, they are, for a $tring, the length of the $tring; the address of the location of the actual $tring elsewhere in memory; and 00 to end the variable.

This latter characteristic is what brought me to not inconsiderable grief. The same difference exists for the storage of numeric and $tring arrays. Arrays start a bit differently, but the idea is the same. The first seven bytes define the array. For a string array, they are as shown in Table 1. The array used is dimensioned at two, which will give it three elements (remember that "0" is a place for a computer). In addition, you must remember that non dimensioned variables default to ten. Thus they have eleven elements, counting the 0th to the tenth. The third byte of each array is a pointer so the program can easily find the next array without searching through every element of each. It represents 7 + (No. of elements) * 4, which is the number of bytes to the next array. The fourth through seventh bytes contain $00, $01, $00, and (No. of elements + 1), respectively.

Next is the elements of the array, with four bytes each. They are, in order, the length of the $tring for that element; the address of the $tring elsewhere in memory; and 00 to end the element. If the $tring's value is established in the source code, whether in a DATA statement or as a $tring constant, its location stays with the source code. If the $tring is etablished during the run of the program, by keyboard input or through $tring manipulation, it is placed in high memory, working from the top of RAM down. However, you can fool the program. By concatenating a $tring with a zero length string, the BASIC routine thinks a new $tring has been established, and puts it at top of memory as well as in source code. A$ = A$ + "" does it. The disadvantage is that the $tring is now in two places, with attendant use of extra memory.

But why would you want to do this? One reason came to light during the creation of the Paper Route program. When a new customer is added, the routine at Line 525 of Listing 2 opens a space, and readdresses all of the Name $trings from the insertion location up. This means that the $tring in source code which used to be N$(X) is now N$(X + 1), and so on. Everything works fine, as the new N$(X) is INPUT and placed at top of memory. The problem arises when you try to save all the $trings, old and new, to DATA by the routines between Lines 800 and 995. I at first tried to save from N$(1) to N$(75) in all cases. It worked whenever customers were deleted. But if customers were added, everyone from the new one to the end had the new person's name. As it turned out, the program was trying to pick itself up by its own bootstraps!

If a customer is added at number 3, then old customer 3 becomes number 4. But the name is still stored in the third DATA statement. Now you start to rePOKE the DATA statements, from 1 to N. What happens? Old 3 is replaced by new 3. But now you try to read new 4 (which was old 3), and you, instead, get new 3. Sounds simple. But it sure was perplexing until I reached a fairly complete understanding of the $tring variable storage system discussed above.

Several solutions appeared possible, with the easiest to just concatenate each string. But that turned out to use up more than my 8K of RAM with 75 customers. The best answer seemed to be to reverse the order for adds and deletes. The disadvantage here is time; it takes about 35 seconds for the save routine, and it must be done once each for adds and deletes. On the other hand, more efficient solutions would involve machine language routines or complicated $tring manipulations. ©