SECOND FEATURE

SUBROUTINE MASTER

DOS 3.3

Now you can build a library of Applesoft subroutines that behave much like Pascal procedures. Features include two-way parameter passing, local variables, nesting, and recursion.

ProDOS

by H. Cem Kaner and John R. Vokey, 256C Calle Marguerita, Los Gatos, CA 95030

One of Applesoft's main drawbacks is the lack of "real" subroutines. With BASIC's weak subroutine capabilities, we find ourselves constantly rewriting the same sections of code, and tailoring them to each new program. Ideally, we should be writing fairly general subroutines, recording them on a subroutine library disk, and then merging them with any program that needs them. Subroutine libraries save time and reduce errors: once a general-purpose subroutine has been debugged, it can be used with confidence, and without error, in program after program.

Despite the advantages of library creation, we find that we don't do this in standard Applesoft, nor do we often swap subroutines or take subroutines from articles without modifying them, often extensively. In contrast, even in a language as antiquated and downright clumsy as FORTRAN II, developing subroutine libraries is a very natural programming activity. We also have problems translating programs written in the usual business and scientific languages into Applesoft. Most of the published programs worth translating rely heavily on subroutine facilities that do not exist in BASIC. It's not impossible to get around this, but it makes translation a frustrating, time-consuming, and error-ridden process.

Subroutine Master (**Listing 1**) adds "real" subroutine handling to Applesoft. It's still not perfect and program execution is slower than we would like. But using this program has saved us a great deal of programming and debugging time, which easily justifies the cost of some of the computer's (not our) execution time.

AN EXAMPLE

The program shown in **Example 1** is a very simple example of subroutine calls using our handler. This program uses the same subroutine, SRT, to sort the elements of two different arrays into ascending order. A normal subroutine (**line 150**) is used to print the unsorted X() array in **line 40** and the sorted values in **line 60**. We'll discuss the features and syntax of Subroutine Master's subroutines in detail shortly. For now, read through the remarks, which illustrate some of the handler's capabilities. Note the following features of Subroutine Master:

1. *Reference to a subroutine by name*. The subroutine named SRT starts at **line 90**. It wouldn't matter if this were moved. The computer will find the subroutine SRT wherever it appears in the program.
2. *Variable passing to the subroutine*. **Line 60** CALLs (GOSUBs) SRT and passes two pieces of information to it. As called from **line 60**, SRT sorts the ten elements of array X(). When called from **line 70**, SRT sorts the 20 elements of array Y() instead. In both cases, SRT thinks it's working with N elements of array S(). It has no idea that these are called by different names in the main (calling) program.
3. *Local variables*. In **line 100**, variables I, J and S are declared LOCAL to SRT. This means that they are created specifically for the SRT routine, and that they will only exist in memory while SRT is active. The value of the main program's variable, I, which was used in **lines 40 and 50**, is absolutely unaffected by any changes in the value of the local variable I. The two I's have the same name, but they are entirely different variables.

USING THE PROGRAM

Before you use any of the commands described below, SUBR.MASTER (**Listing 1**) must be installed and the beginning of the Applesoft program adjusted upward. See the two demonstration programs (**Listings 2 and 3**) for examples of how to do this from within an Applesoft program. The demonstration programs will run as they are, as long as a disk with SUBR.MASTER is in the current disk drive.

Near the beginning of the program, the variable EXIT should be set to 4058, and each subroutine name should be set to 3141. (Note that Applesoft only distinguishes variables by their first two characters. However, Subroutine Master can distinguish longer names. The two-character variable should not be changed from the initial 3141 setting.)

Five statements are included in the Subroutine Master system:

CALL ***name,parameter list*** — This calls a subroutine by *name*, passing the variables or expressions in the *parameter list* to the

EXAMPLE 1: A Program Using Subroutine Master

```
                                                          Remarks
10   IF  PEEK (1Ø4) <  > 17 THEN  POKE 1Ø3,44:            Reload program and load subroutine master
     POKE 1Ø4,17: POKE 4395,Ø: PRINT  CHR$ (
     4)"BLOAD SUBR.MASTER": PRINT  CHR$ (4)"R
     UN EXAMPLE1"
2Ø   SRT = 3141:EXIT = 4Ø58                               Handler address definitions
3Ø   DIM X(1Ø),Y(2Ø)
4Ø   FOR I = 1 TO 1Ø:X(I) =  RND (1): NEXT : GOSUB 15Ø    Starts with data in random order
5Ø   FOR I = 1 TO 2Ø:Y(I) =  RND (1): NEXT
6Ø   CALL SRT,X(Ø),1Ø: GOSUB 15Ø                          Sorts it
7Ø   CALL SRT,Y(Ø),2Ø                                     Uses CALL statement and variable list
8Ø   END
9Ø   DEF SRT,S(Ø),N                                       Subroutine DEFinition
1ØØ LOCAL,I,J,S                                           Declaration of LOCAV variables
11Ø FOR I = 1 TO N - 1: FOR J = I + 1 TO N                Actual subroutine starts here
12Ø IF S(I) > S(J) THEN S = S(I):S(I) = S(J)              Reorders the array elements from smallest to
    :S(J) = S                                             largest
13Ø NEXT J,I
14Ø CALL EXIT,SRT                                         EXIT (subroutine return) statement
15Ø FOR I = 1 TO 1Ø: PRINT X(I):NEXT:RETURN               Standard subroutine to print array X()
```

corresponding variables in the subroutine's DEF header. Floating point, integer, and string variables, as well as arrays of all three types, may be included in the *parameter list*. The CALL *name* statement may appear anywhere in an Applesoft program line. **Line 140** of **Listing 2** is an example of passing a string literal, while **line 350** demonstrates passing a floating point variable.

DEF *name,parameter list* — This marks the beginning of a named subroutine. The variables in the *parameter list* receive values from items in the corresponding *parameter list* of a CALL statement. The variables used in the *parameter list* are local variables, which are passed back to the corresponding CALL variables on return from the subroutine. The DEF statement must be the first statement on a program line. **Line 390** of **Listing 1** is a typical DEF statement.

CALL EXIT,*name* — This marks the end of a named subroutine. The *name* used in the DEF header must be included in the CALL EXIT statement. CALL EXIT must be the last statement on a program line. **Line 420** of **Listing 2** is the CALL EXIT statement that corresponds to **line 390.**

CALL DISP,*variable list* — DISP (short for DISPOSE) removes the variables named in the *variable list* from memory. If you need to use this command, you must also set DISP equal to 2304 at the beginning of the program.

LOCAL,*variable list* — This optional statement must be the next statement after the DEF statement of the subroutine. It declares the variables in the *variable list* as local variables — distinct from variables of the same name used in the main program or in other subroutines. **Line 780** of **Listing 2** makes the variable Z$ local to the RET subroutine.

Values are passed from the parameters listed in the CALL *name* statement to the variables listed in the DEF statement. If the CALL *name* statement uses variables, rather than expressions, the values of DEF variables are passed back to variables in the CALL statement.

An entire array may be passed as a parameter in a CALL *name* statement. This is indicated by simply placing a zero in place of each index. The corresponding array variable in the DEF statement should be indicated in the same way. You can pass an array element to a subroutine as a simple variable, but you can't send an array element or a simple variable to an element of a subroutine array. Arrays may be specified as LOCAL by specifying the array in the same way you would in a DIM statement. The LOCAL statement automatically dimensions the array.

The system does not allow you to create new global variables inside a subroutine. If you want to change a global variable from within a subroutine, make sure that it has already been created in the main program.

User-defined functions (DEF FN) may not be included in a CALL *name* statement, nor may they be used inside a subroutine. Other restrictions, idiosyncracies, and error messages are discussed later.

ENTERING THE PROGRAMS

To key in SUBR.MASTER, either use your assembler to enter the source code from **Listing 1**, or type CALL −151 <RETURN> and use the Monitor to enter the hex codes. The entire source file may be too long to fit into the memory available with some assemblers. In this case, you may have to split it into two parts, as we did with Apple's DOS Tool Kit Assembler. Be sure that the name of the second file and the name specified in the CHN (or equivalent) command at the end of the first file match. Then save the program with the command:

BSAVE SUBR.MASTER,A$900,L$82B

Key in **Listing 2** and save it with the command:

SAVE SUBR.MAST.DEMO1

Key in **Listing 3** and save it with the command:

SAVE SUBR.MAST.DEMO2

These programs relocate themselves in memory, so it is important that you save them before you run them. Also, be sure that SUBR.MASTER is on the disk in the current disk drive. For help in entering *Nibble* listings, see "A Welcome to New *Nibble* Readers" at the beginning of this issue.

DESIGN CRITERIA

What features should "real" subroutines have? We knew how we wanted the subroutine handler to interact with the user long before we figured out how to achieve this. We worked on a number of conceptually very different approaches to creating "real" subroutines before settling on the one presented here. In this section we describe our general goals and outline the approaches taken to meet them.

Non-Interference With GOSUB

Nothing in our program affects GOSUB, POP or RETURN in any way, and we do not store our return parameters in the stack, which is GOSUB and FOR territory. GOSUB subroutines can still be used freely and will come in handy in many programs.

One Entry, One Exit

You should always have to enter a subroutine at the same place, the beginning, and leave it at the same place, the end. This is a key restriction underlying the philosophies of modular and structured programming, mainly because it eliminates a regular source of programming errors.

Our program does not allow multiple entry points in a subroutine. However, multiple exit statements are possible within subroutines. Since multiple entry points and unexpected subroutine entries caused us much more grief than multiple exits, we were less worried about restricting the exits.

Space Efficiency

In passing variables from the main program to a subroutine, as we did with the

SRT routine, we rename X() to S(), calling it X() again on exit. This is fast and simple, unless there's already an array called S() in memory. In that case, the new S() (the old X()) must be moved down in memory so that the new S() will be used in SRT. Instead of costing us 5,000 or more bytes (as it would if we had chosen to copy the array), passing arrays costs us no memory beyond the length of the routines required to rename, check, and, if necessary, move the variables.

A second space-expensive trick is to set aside a reserved area of memory for local variables. This approach allows local variables to retain their values, but it adds the requirement of zeroing these variables and it consumes way too much memory. This is a luxury — if it is a luxury — that we simply cannot afford. Instead, we get rid of the locals on exit from subroutines, freeing up the memory they occupied for use by the rest of the program.

In both cases, we trade speed for space efficiency. We are much more concerned about handling lots of data, running large programs, and using high resolution graphics than we are (usually) about saving a few seconds. When speed is more important, we can always use the old standby, custom-tailored GOSUB subroutines, instead.

Recursion

In recursive programming languages (such as Pascal), subroutines can call themselves freely. In contrast, languages like FORTRAN never allow a subroutine to call itself. Applesoft is partially recursive. Up to the limits of available space in the stack (which disappears fairly quickly), subroutines can call themselves. We felt that our routines should be as fully recursive as possible, treating all of free memory as a stack.

In principle our subroutines allow extensive recursion. In practice, with CALL piled upon CALL, highly recursive programs run quite slowly, which limits the utility of this approach. Still, it will be important for some users, particularly students who wish to learn about recursion, that such programs execute correctly, if not promptly. **Listing 3** is a simple example of using recursion.

Variable Passing

In **Example 1**, we called SRT twice, once passing it array X(), the next time passing it Y(). To do this in standard BASIC, in **line 60** we would have had to resort to something like:

FOR I=1 TO 10: S(I)=X(I): NEXT: N=10: GOSUB 100

instead of:

CALL SRT,X(),10

Then at **line 70** we would have had to do the same thing again just to pass down Y() and 20. This is a tedious and error-prone method.

LISTING 1: SUBR.MASTER

```
ØØØØ:    ;**********************
ØØØØ:    ;*     SUBR.MASTER      *
ØØØØ:    ;*  BY CEM KANER AND    *
ØØØØ:    ;*     JOHN VOKEY       *
ØØØØ:    ;* COPYRIGHT (C) 1985   *
ØØØØ:    ;* BY MICROSPARC, INC   *
ØØØØ:    ;* CONCORD, MA  Ø1742   *
ØØØØ:    ;**********************
ØØØØ:    ;
ØØØØ:    ; DOS TOOLKIT ASSEMBLER
ØØØØ:    ;
ØØØØ:    ;      PROGRAM CONSTANTS
ØØØØ:    ;
ØØ24:    STRING   EQU  $24         ; $
ØØ25:    PERCENT  EQU  $25         ; %
ØØ28:    LPAREN   EQU  $28         ; (
ØØ29:    RPAREN   EQU  $29         ; )
ØØ2C:    COMMA    EQU  $2C         ; ,
ØØ3A:    COLON    EQU  $3A         ; :
ØØ41:    EH       EQU  $41         ; A
ØØ43:    CE       EQU  $43         ; C
ØØ4C:    EL       EQU  $4C         ; L
ØØ4F:    OH       EQU  $4F         ; O
ØØ8C:    CALL     EQU  $8C         ; CALL TOKEN
ØØB8:    DEF      EQU  $B8         ; DEF TOKEN
ØØØØ:    ;
ØØØØ:    ;      APPLESOFT ROUTINES
ØØØØ:    ;
ØØB1:    CHRGET   EQU  $B1         ; FETCH CHR AT TXTPTR
ØØB7:    CHRGOT   EQU  $B7         ; RECOVER LAST CHR. SET FLAGS
D393:    BLTU     EQU  $D393       ; BLOCK TRANSFER UP
D39A:    BLTUP    EQU  $D39A       ; BLTU AFTER REASON TEST
D412:    ERROR    EQU  $D412       ; DIE HORRIBLY
D41Ø:    OMERR    EQU  $D41Ø       ; OUT OF MEMORY
D697:    STXTPT   EQU  $D697       ; TXTPTR = START OF PROGRAM TEXT
D998:    ADDON    EQU  $D998       ; ADD Y TO TXTPTR
D9A6:    REMN     EQU  $D9A6       ; PUT OFFSET TO EOL IN REG Y
DA52:    LETCNT   EQU  $DA52       ; LATE ENTRY TO LET
DB97:    GETTXT   EQU  $DB97       ; TRANSFER OLDTXT TO TXTPTR
DD76:    MISMATCH EQU  $DD76       ; TYPE MISMATCH
DEBE:    CHKCOM   EQU  $DEBE       ; CRASH IF NOT COMMA
DEC9:    SYNERR   EQU  $DEC9       ; SYNTAX ERROR
DFD9:    DIM      EQU  $DFD9       ; DIMENSION COMMAND
DFE3:    PTRGET   EQU  $DFE3       ; FIND VARIABLE IN MEMORY
EØ7D:    ISLETC   EQU  $EØ7D       ; SET CARRY IF A HOLDS A LETTER
E199:    QUANTERR EQU  $E199       ; ILLEGAL QUANTITY ERROR
E1BC:    DATAERR  EQU  $E1BC       ; OUT OF DATA
E3Ø6:    ERRDIR   EQU  $E3Ø6       ; CRASH IF IN IMMEDIATE MODE
ØØØØ:    ;
ØØØØ:    ;      APPLESOFT ADDRESSES
ØØØØ:    ;
ØØ1Ø:    DIMFLAG  EQU  $1Ø         ; DIMENSION FLAG
ØØ11:    VALTYP   EQU  $11         ; FF IF STRING
ØØ12:    INTFLAG  EQU  $12         ; 8Ø IF INTEGER
ØØ14:    SUBFLAG  EQU  $14         ; 8Ø IF SUBSCRIPT OK
ØØ3C:    MOVESTART EQU $3C         ; MONITOR A1L, A1H
ØØ3E:    MOVEND   EQU  $3E         ; MONITOR A2L, A2H
ØØ42:    MOVETO   EQU  $42         ; MONITOR A4L, A4H
ØØ69:    VARTAB   EQU  $69         ; START VARIABLE STORAGE
ØØ6B:    ARYTAB   EQU  $6B         ; START ARRAY STORAGE
ØØ6D:    STREND   EQU  $6D         ; END VARIABLE STORAGE
ØØ75:    CURLIN   EQU  $75         ; CURRENT LINE #
ØØ79:    OLDTXT   EQU  $79         ; OLD TEXT POINTER
ØØ81:    LASTVAR  EQU  $81         ; LATEST VARIABLE NAME
ØØ83:    VARPNT   EQU  $83         ; POINTS LATEST VARIABLE VALUE
ØØ85:    FORPNT   EQU  $85         ; USED BY LET
ØØ94:    HIGHDS   EQU  $94         ; HIGH DESTINATION, BLTU
ØØ96:    HIGHTR   EQU  $96         ; HIGH TRANSFER, BLTU
ØØ9B:    LOWTR    EQU  $9B         ; PTR TO VAR NAME OR LOW TRANSFER
ØØB8:    TXTPTR   EQU  $B8         ; TEXT POINTER
Ø2ØØ:    BUFR     EQU  $2ØØ        ; INPUT BUFFER
ØØØØ:    ;
Ø8Ø1:    PO       EQU  $8Ø1        ; TRUE PROGRAM ORIGIN
----- NEXT OBJECT FILE NAME IS XX
Ø9ØØ:             ORG  PO+$FF      ; 1ST PAGE FOR DATA
Ø9ØØ:    ;
Ø9ØØ:    ;      PROGRAM VARIABLES
Ø9ØØ:    ;
Ø9ØØ:    ; $77-DF LOCS USED ARE WALKED ON BY APPLESOFT
Ø9ØØ:    ; ERROR, BUT DON'T INTERFERE WITH APPLESOFT
Ø9ØØ:    ; FUNCTIONING ITSELF, SO ARE SAFE TEMPS.
Ø9ØØ:    ;
ØØ77:    NUMCHR   EQU  $77         ; (OLDLIN) # CHRS TO MOVE
ØØ78:    NUMPAGE  EQU  $78         ; # PAGES TO MOVE
ØØDA:    BUFPTR   EQU  $DA         ; (ERRLIN) PTR FOR BUFR, SECBUF
ØØDB:    COUNTER  EQU  $DB         ; (ERRLIN,ERRPOS) 2ND PTR
ØØDD:    ARYFLAG  EQU  $DD         ; (ERRPOS) FF IF ARRAY OR ARY EXPR
ØØDE:    EXPRFLAG EQU  $DE         ; (ERRNUM) FF IF EXPR
ØØDF:    PARENCOUNT EQU $DF        ; (ERRSTK) # PARENS LEFT
ØØFA:    GENPTR   EQU  $FA         ; GENERAL POINTER
ØØFC:    CALLPTR  EQU  $FC         ; POINT TO CALL LIST
ØØFE:    DEFPTR   EQU  $FE         ; POINT TO DEF LIST
Ø8Ø1:    FATOFF   EQU  PO          ; STORE OLD $FA TO $FF
Ø8Ø7:    DEFLIST  EQU  PO+$Ø6      ; POINT TO START DEF VARLIST
Ø8Ø9:    CALLIST  EQU  PO+$Ø8      ; POINT TO START CALL VARLIST
```

```
Ø8ØB:            DEFLINE EQU  PO+$ØA     ; DEF LINE NUMBER
Ø8ØD:            PROCNAME EQU PO+$ØC     ; POINT TO PROC NAME
Ø8ØF:            OLDARYTAB EQU PO+$ØE    ; SAVED ARYTAB
Ø811:            OLDVARTAB EQU PO+$1Ø
Ø813:            OLDSTREND EQU PO+$12
Ø815:            OLDSIMPLE EQU PO+$14    ; START OF ORIGINAL SIMPLES
Ø817:            NEWARYTAB EQU PO+$16    ; FOR MOVE ROUTINE
Ø819:            HOLDCOMMA EQU PO+$18    ; SAVES CHR FROM PUTCOLON
Ø81A:            SECBUF  EQU  PO+$19     ; SECOND BUFFER
ØØDØ:            BUFMAX  EQU  $DØ        ; MAX # CHRS ALLOWED IN SECBUF
Ø9ØØ:                    CHN  SUBR.MAST.S2
Ø9ØØ:            ;
Ø9ØØ:            ;      SUBROUTINES
Ø9ØØ:            ;
Ø9ØØ:            ************************************
Ø9ØØ:            *    CALL THIS ADDRESS TO DISPOSE   *
Ø9ØØ:            *    OF A VARIABLE.  FOR DOCUMEN-   *
Ø9ØØ:            *    TATION SEE VOKEY & KANER, 1982 *
Ø9ØØ:            ************************************
Ø9ØØ:            ; NOTE: THIS ROUTINE IS MODIFIED FROM THE
Ø9ØØ:            ; BYTE PAPER AS FOLLOWS: JSR MOVE ($FE2C)
Ø9ØØ:            ; IS CHANGED TO JSR NEWMOVE ($Ø96F).  THIS
Ø9ØØ:            ; CUTS EXECUTION TIME FOR LARGE MOVES BY UP
Ø9ØØ:            ; TO 7Ø% BUT ELIMINATES RELOCATABILITY
Ø9ØØ:            ; OF THE CODE.
Ø9ØØ:            ;
Ø9ØØ:2Ø B1 ØØ    DISPOSE JSR  CHRGET     ; MOVE PAST COMMA
Ø9Ø3:2Ø B7 ØØ    CLEAR   DFB  $2Ø,$B7,$ØØ,$DØ,$Ø3,$4C,$6C,$D6
Ø9Ø6:DØ Ø3 4C
Ø9Ø9:6C D6
Ø9ØB:2Ø E3 DF            DFB  $2Ø,$E3,$DF,$C4,$6C,$DØ,$Ø2,$C5
Ø9ØE:C4 6C DØ
Ø911:Ø2 C5
Ø913:6B AØ Ø2            DFB  $6B,$AØ,$Ø2,$Ø8,$BØ,$ØA,$A9,$ØØ
Ø916:Ø8 BØ ØA
Ø919:A9 ØØ
Ø91B:C8 91 9B            DFB  $C8,$91,$9B,$88,$A9,$Ø7,$91,$9B
Ø91E:88 A9 Ø7
Ø921:91 9B
Ø923:18 B1 9B            DFB  $18,$B1,$9B,$85,$44,$A5,$9B,$85
Ø926:85 44 A5
Ø929:9B 85
Ø92B:42 65 44            DFB  $42,$65,$44,$85,$3C,$A5,$9C,$85
Ø92E:85 3C A5
Ø931:9C 85
Ø933:43 C8 71            DFB  $43,$C8,$71,$9B,$85,$3D,$B1,$9B
Ø936:9B 85 3D
Ø939:B1 9B
Ø93B:85 45 AØ            DFB  $85,$45,$AØ,$ØØ,$A5,$6D,$85,$3E
Ø93E:ØØ A5 6D
Ø941:85 3E
Ø943:A5 6E 85            DFB  $A5,$6E,$85,$3F,$2Ø,$6F,$Ø9,$A5
Ø946:3F 2Ø 6F
Ø949:Ø9 A5
Ø94B:6D E5 44            DFB  $6D,$E5,$44,$85,$6D,$A5,$6E,$E5
Ø94E:85 6D A5
Ø951:6E E5
Ø953:45 85 6E            DFB  $45,$85,$6E,$28,$BØ,$ØA,$A5,$6B
Ø956:28 BØ ØA
Ø959:A5 6B
Ø95B:E9 Ø6 85            DFB  $E9,$Ø6,$85,$6B,$BØ,$Ø2,$C6,$6C
Ø95E:6B BØ Ø2
Ø961:C6 6C
Ø963:2Ø B7 ØØ            DFB  $2Ø,$B7,$ØØ,$DØ,$Ø1,$6Ø,$2Ø,$BE
Ø966:DØ Ø1 6Ø
Ø969:2Ø BE
Ø96B:DE 38 BØ            DFB  $DE,$38,$BØ,$9C
Ø96E:9C
Ø96F:38          NEWMOVE SEC              ; SET UP IN SAME WAY AS
Ø97Ø:A5 3E               LDA  MOVEND      ; MONITOR MOVE BUT EXECUTION
Ø972:E5 3C               SBC  MOVESTART   ; IS MUCH FASTER FOR
Ø974:85 77               STA  NUMCHR      ; NON-TRIVAL MOVES.  MOVEND
Ø976:A5 3F               LDA  MOVEND+1    ; MUST BE STRICTLY GREATER
Ø978:E5 3D               SBC  MOVESTART+1 ; THAN MOVESTART.
Ø97A:85 78               STA  NUMPAGE     ; # FULL PAGES TO MOVE
Ø97C:BØ Ø3               BCS  ADD1        ; DO A RANGE CHECK. CRASH
Ø97E:4C 99 E1            JMP  QUANTERR    ; IF MOVEND <= MOVESTART.
Ø981:E6 77       ADD1    INC  NUMCHR      ; TOTAL # BYTES IS 1 SHY
Ø983:DØ Ø2               BNE  PAGECHECK   ; FROM SUBTRACTION. SO ADD
Ø985:E6 78               INC  NUMPAGE     ; IT BACK IN
Ø987:A5 78       PAGECHECK LDA NUMPAGE    ; ANY FULL PAGES TO MOVE?
Ø989:FØ 11               BEQ  PARTMOVE    ; IF NOT, DO PARTIAL PAGE
Ø98B:AØ ØØ               LDY  #Ø          ; START OF FULL PAGE
Ø98D:B1 3C       PAGEMOVE LDA (MOVESTART),Y ; MOVES.
Ø98F:91 42               STA  (MOVETO),Y
Ø991:C8                  INY              ; PAGE DONE?
Ø992:DØ F9               BNE  PAGEMOVE    ; LEAVES WITH Y=Ø
Ø994:E6 3D               INC  MOVESTART+1 ; ADJUST FOR NEXT PAGE
Ø996:E6 43               INC  MOVETO+1
Ø998:C6 78               DEC  NUMPAGE     ; ANOTHER LEFT?
Ø99A:DØ F1               BNE  PAGEMOVE    ; DO TILL DONE LAST.
Ø99C:A5 77       PARTMOVE LDA NUMCHR      ; ANY LEFT?
Ø99E:FØ Ø9               BEQ  MOVEDONE    ; CARRY SET FROM BCS ADD1
Ø9AØ:B1 3C       PARTMOVE1 LDA (MOVESTART),Y ; MOVE LAST NUMCHR BYTES
Ø9A2:91 42               STA  (MOVETO),Y ; Y STARTS Ø FROM ABOVE
Ø9A4:C8                  INY              ; RUNS TO NUMCHR-1
Ø9A5:C4 77               CPY  NUMCHR      ; FOR TOTAL OF NUMCHR BYTES
```

Other languages solve the problem of passing variables by forcing the programmer to specify which variables a subroutine is to act on, each time the routine is called. So we allow variable lists as in FORTRAN, Pascal, COBOL, and many other languages. There is one big difference, of course. If you want to use a GOSUB with no explicit variable passing, you can use a GOSUB. But now you don't have to.

Ideally, variable passing should be as unrestricted as is variable assignment in Applesoft. You should be able to pass reals to integers and vice versa; to pass array elements (like X(10)) back and forth; and to pass expressions to simple variables. We achieved most, but not all of this.

Portability

If you can use a friend's subroutine correctly in your own program after spending less than five minutes examining it, the subroutine is "portable" — it moves easily from person to person and from program to program. Making it easy to write portable subroutines is the main goal of this program. A variety of factors increase portability. We've looked at one already: the less a subroutine is tied to specific variable names, the more general, and the more portable, it will be.

Named Subroutines — People understand names much better than they understand line numbers when they are trying to figure out the function of some section of a program. Accordingly, it should be possible to refer to the separate sections of a program by name, rather than location.

Our program searches for actual names, without ever using line numbers, by scanning the start of each line for a DEF token. When it finds a DEF, it compares what follows to the name of the subroutine it's looking for. When it finds a matching name, it has found the right routine.

Reusable Variable Names — When was the last time you wrote something like:

FOR I = 1 TO 10: GOSUB 1000

only to discover later that the subroutine at **line 1000** changes the value of I? This kind of bug is as annoying as it is common. You should be able to write a subroutine without worrying about what variable names will be used in any of the programs that call it. The subroutine's variables should not affect those of the main program unless you want them to.

Our first step in eliminating conflicts between main and subprogram variable names was to create local variables. Declare a variable LOCAL in a subroutine and a brand new variable (any type, simple or array) of this name is created in memory. Reference to this variable has no effect on any of the main program's data. Further, because the locals are cleared out of memory on exit from the subroutine, that routine gives back as much free memory as it got, so there's no conflict with future main program variable storage requirements either.

As a second step, we added variable passing. The variables passed to a subroutine are renamed to the names in the DEF list. We make sure that variables passed to the subroutine are stored lower in memory than variables of the main program, which have the same names as those found in the DEF lists. Because of this, Applesoft always operates, in the subroutine, on the subroutine's variables. This protects the main program's variables from being changed in the subroutine accidentally. Thus, you can send variables to the subroutine without knowing or caring what it will call them there; you can call them whatever you want in the subroutine. You will affect the variables you think you are working with, and no others.

A third level of protection against unexpected reference to variables in a subroutine was built in to allow natural use of "global" variables. A global variable is defined in the main program but can be used in a subroutine without appearing in the DEF or LOCAL parameter lists. Some (not many) variables can and should be safely made global. Think of D$=CHR$(4), for example. DOS requires the programmer to define this, or something like it, in every program that uses disk access. It's tedious enough doing DOS's housekeeping for it once per program, so you shouldn't have to worry about passing it or redefining it for each subroutine. Globals should be made and unmade in the main program; subroutines that tinker with global storage are not portable. This system detects the initial definition of a global variable within a subroutine, and signals it with a MEMORY ERROR.

Explicit Subroutine Interface — To use a subroutine correctly, you need to understand its inputs (what gets passed down), its outputs (what variables it can and does affect) and its function. These three aspects (the "subroutine interface" with the main program) are all that you need to know about the subroutine. You do not need to know the details (the "subroutine quagmire") of how the subroutine does what it's supposed to do. As long as it does it correctly, don't worry about how it does it.

If the subroutine interface is laid out clearly, correctly and briefly, you should be able to use that routine in your program within minutes. If the interface has to be fished out of the quagmire, you may as well, and probably will, rewrite the beast instead.

For a subroutine to be portable, it must be well documented, meaning that its interface must be easy to find and understand. The DEF statement's variable list tells you automatically what types of variables the subroutine expects as input. If no globals are used, the DEF statement describes the types of all inputs. The CALL statement's variable list identifies the inputs themselves, tell-

LISTING 1: SUBR.MASTER *(continued)*

```
Ø9A7:9Ø F7                BCC   PARTMOVE1
Ø9A9:6Ø         MOVEDONE RTS                   ; LEAVES WITH CARRY SET
Ø9AA:           ;
Ø9AA:           ;      HOUSEKEEPING
Ø9AA:           ;
Ø9AA:2Ø Ø6 E3   IN       JSR   ERRDIR      ; USES 2ØØ+. CRASH IN IMM MODE
Ø9AD:A2 Ø5               LDX   #5          ; TRANSFER FA TO FF
Ø9AF:B5 FA      SAVEPØ   LDA   $FA,X       ; TO A SAFE PLACE
Ø9B1:9D Ø1 Ø8            STA   FATOFF,X    ; WILL PUT THEM BACK
Ø9B4:CA                  DEX               ; ON EXIT
Ø9B5:1Ø F8               BPL   SAVEPØ      ; LOOP TILL DONE
Ø9B7:6Ø                  RTS
Ø9B8:A2 Ø5      OUT      LDX   #5          ; RESTORE FA TO FF
Ø9BA:BD Ø1 Ø8   BACKPØ   LDA   FATOFF,X    ; TO THEIR
Ø9BD:95 FA               STA   $FA,X       ; OLD HOME
Ø9BF:CA                  DEX
Ø9CØ:1Ø F8               BPL   BACKPØ
Ø9C2:6Ø                  RTS
Ø9C3:A5 B8      SAVETXT  LDA   TXTPTR      ; SAVE TEXT POINTER
Ø9C5:85 79               STA   OLDTXT      ; IN APPLESOFT'S
Ø9C7:A5 B9               LDA   TXTPTR+1    ; USUAL
Ø9C9:85 7A               STA   OLDTXT+1    ; HIDEYHOLE
Ø9CB:6Ø                  RTS
Ø9CC:A5 FE      POINTDEF LDA   DEFPTR      ; POINT TEXT POINTER
Ø9CE:85 B8               STA   TXTPTR      ; AT THE DEF LIST
Ø9DØ:A5 FF               LDA   DEFPTR+1
Ø9D2:85 B9               STA   TXTPTR+1
Ø9D4:6Ø                  RTS
Ø9D5:A5 FC      POINTCALL LDA CALLPTR      ; POINT TXTPTR
Ø9D7:85 B8               STA   TXTPTR      ; AT THE CALL LIST
Ø9D9:A5 FD               LDA   CALLPTR+1
Ø9DB:85 B9               STA   TXTPTR+1
Ø9DD:6Ø                  RTS
Ø9DE:A5 B8      TXTTODEF LDA   TXTPTR      ; UPDATE
Ø9EØ:85 FE               STA   DEFPTR      ; DEF POINTER
Ø9E2:A5 B9               LDA   TXTPTR+1
Ø9E4:85 FF               STA   DEFPTR+1
Ø9E6:6Ø                  RTS
Ø9E7:A5 B8      TXTTOCALL LDA TXTPTR       ; UPDATE
Ø9E9:85 FC               STA   CALLPTR     ; THE CALL PTR
Ø9EB:A5 B9               LDA   TXTPTR+1
Ø9ED:85 FD               STA   CALLPTR+1
Ø9EF:6Ø                  RTS
Ø9FØ:AD Ø9 Ø8   STARTLIST LDA CALLIST      ; POINT THE
Ø9F3:85 FC               STA   CALLPTR     ; CALL AND DEF PTRS
Ø9F5:AD ØA Ø8            LDA   CALLIST+1   ; AT THE START OF
Ø9F8:85 FD               STA   CALLPTR+1   ; THEIR VARIABLE
Ø9FA:AD Ø7 Ø8            LDA   DEFLIST     ; LISTS
Ø9FD:85 FE               STA   DEFPTR
Ø9FF:AD Ø8 Ø8            LDA   DEFLIST+1
ØAØ2:85 FF               STA   DEFPTR+1
ØAØ4:6Ø                  RTS
ØAØ5:AD ØD Ø8   POINTNAME LDA PROCNAME     ; POINT
ØAØ8:85 B8               STA   TXTPTR      ; TO PROC NAME
ØAØA:AD ØE Ø8            LDA   PROCNAME+1
ØAØD:85 B9               STA   TXTPTR+1
ØAØF:6Ø                  RTS
ØA1Ø:2Ø Ø5 ØA   GETNAME  JSR   POINTNAME   ; POINT TO PROC NAME
ØA13:2Ø B1 ØØ            JSR   CHRGET      ; ADVANCE PAST LEADING COMMA
ØA16:4C E3 DF            JMP   PTRGET      ; FIND IT IN MEMORY
ØA19:A5 B8      DECTXT   LDA   TXTPTR      ; MOVE TXTPTR
ØA1B:DØ Ø2               BNE   DECTXTLOW   ; BACK 1
ØA1D:C6 B9               DEC   TXTPTR+1    ; FROM HIGH BYTE IF NEEDED
ØA1F:C6 B8      DECTXTLOW DEC TXTPTR       ; FROM LOW BYTE ALWAYS
ØA21:6Ø                  RTS
ØA22:A5 69      SAVEVARTAB LDA VARTAB      ; THESE ROUTINES
ØA24:8D 11 Ø8            STA   OLDVARTAB   ; USED TO STASH
ØA27:A5 6A               LDA   VARTAB+1    ; APPLESOFT POINTERS
ØA29:8D 12 Ø8            STA   OLDVARTAB+1 ; WHILE THE ROUTINE
ØA2C:6Ø                  RTS               ; PLAYS WITH THEM
ØA2D:AD 11 Ø8   GETVARTAB LDA OLDVARTAB    ; ALL ARE MADE
ØA3Ø:85 69               STA   VARTAB      ; SUBROUTINES
ØA32:AD 12 Ø8            LDA   OLDVARTAB+1 ; NO MATTER HOW
ØA35:85 6A               STA   VARTAB+1    ; OFTEN CALLED
ØA37:6Ø                  RTS
ØA38:A5 6B      SAVEARYTAB LDA ARYTAB      ; BECAUSE THEY
ØA3A:8D ØF Ø8            STA   OLDARYTAB   ; ARE OF GENERAL USE
ØA3D:A5 6C               LDA   ARYTAB+1    ; FOR MANY DIFFERENT
ØA3F:8D 1Ø Ø8            STA   OLDARYTAB+1 ; UTILITY PROGRAMS
ØA42:6Ø                  RTS
ØA43:AD ØF Ø8   GETARYTAB LDA OLDARYTAB    ; SINCE THIS
ØA46:85 6B               STA   ARYTAB      ; PROGRAM WILL ALMOST
ØA48:AD 1Ø Ø8            LDA   OLDARYTAB+1 ; ALWAYS BE IN CORE
ØA4B:85 6C               STA   ARYTAB+1    ; (FOR US ANYWAY)
ØA4D:6Ø                  RTS               ; THIS GIVES US A
ØA4E:A5 6D      SAVESTREND LDA STREND      ; STANDARD PLACE
ØA5Ø:8D 13 Ø8            STA   OLDSTREND   ; TO FIND THEM
ØA53:A5 6E               LDA   STREND+1    ; INSTEAD OF
ØA55:8D 14 Ø8            STA   OLDSTREND+1 ; REWRITING THEM
ØA58:6Ø                  RTS               ; DOZENS OF TIMES
ØA59:AD 13 Ø8   GETSTREND LDA OLDSTREND
ØA5C:85 6D               STA   STREND
ØA5E:AD 14 Ø8            LDA   OLDSTREND+1
ØA61:85 6E               STA   STREND+1
ØA63:6Ø                  RTS
ØA64:2Ø 22 ØA   SIMPTOVAR JSR SAVEVARTAB   ; PUT OLDSIMPLE
```

ing you automatically, for each call of the subroutine, which variables are inputs and which variables (the whole list) are possible outputs.

If no globals are used, this list covers all possible outputs of the subroutine. No other variables can be changed by it. Add a few REMs on function and variable use at the top of the subroutine, and your documentation of the interface is complete.

Even better, the most important part of this documentation, the statement of inputs and outputs, is always correct. Because the variable lists of the DEF and CALL statements are part of the code, they can never misrepresent the code, as REMs sometimes do. There is no better (or easier) form of documentation than self-documentation, automatically generated by writing the code.

THE FINER POINTS

Passing Simple Variables

When a simple variable is encountered in the DEF list, a new local variable is automatically created with this name and is set equal to the variable or expression in the CALL list. Anything that Applesoft considers legal to do for an equal sign (=) is legal for a pass. You can't pass strings to integers, or commands to anything (X= GOTO?), but otherwise passing is quite flexible. Each simple variable in the DEF list adds seven bytes of overhead to variable storage while the subroutine is in effect. DEF simple variables are erased from memory on exit from the subroutine, after their values have been passed back to the CALL variables. When a large amount of data is stored in memory, creating and clearing these local variables can take noticeable amounts of time.

User-Defined Functions

Functions such as FN A(X) should never be passed to or used within a subroutine. This is the sole exception to the '=' rule for simple variables. Applesoft's internal handling of functions makes use of the absolute memory location of the function, but this location is changed when local variables are created and can be changed when variables are erased. Subroutine calls always involve local variable creation, for storing return pointers (we have to put them somewhere, and as we don't use the stack, we stash them in local variables having the same name as the subroutine). Functions are almost always mishandled as a result. Subroutines exit with simple variable storage as they found it, so functions defined in the main program can be used in the main program at any time.

Passing Arrays

Array variables are always passed via renaming. The array named in the CALL list is given the DEF list name on the way to the subroutine, and it is given its old name back on exit. Typically, this requires only a few microseconds, no matter how large

LISTING 1: SUBR.MASTER *(continued)*

```
ØA67:AD 15 Ø8           LDA  OLDSIMPLE ; IN THE SIMPLE PTR
ØA6A:85 69              STA  VARTAB    ; WITHOUT LOSING
ØA6C:AD 16 Ø8           LDA  OLDSIMPLE+1 ; VARTAB'S OLD
ØA6F:85 6A              STA  VARTAB+1  ; VALUE
ØA71:6Ø                 RTS
ØA72:AD ØB Ø8  DEFTOCUR LDA  DEFLINE   ; PUT THE DEF STATEMENT
ØA75:85 75              STA  CURLIN    ; LINE NUMBER IN CURLIN
ØA77:AD ØC Ø8           LDA  DEFLINE+1
ØA7A:85 76              STA  CURLIN+1
ØA7C:6Ø                 RTS
ØA7D:          ;
ØA7D:          ;      VARIABLE HANDLING SUBROUTINES
ØA7D:          ;
ØA7D:AØ ØØ     SKIPVAR  LDY  #Ø        ; BYPASS THIS VAR OR EXPRN
ØA7F:84 DF              STY  PARENCOUNT ; COUNTS PARENTHESES
ØA81:C8        SKIP1    INY            ; ON ENTRY TXTPTR POINTS
ØA82:B1 B8              LDA  (TXTPTR),Y ; AT LEADING COMMA
ØA84:FØ 21              BEQ  SKIPPED   ; LEAVE ON END OF LINE
ØA86:C9 3A              CMP  #COLON    ; SAME AS END OF LINE
ØA88:FØ 1D              BEQ  SKIPPED
ØA8A:C9 2C              CMP  #COMMA    ; END OF VARIABLE?
ØA8C:DØ Ø6              BNE  SKIP2     ; IF NO, CHECK FOR PARENS
ØA8E:A5 DF              LDA  PARENCOUNT ; GOT COMMA. ANY PARENS?
ØA9Ø:FØ 15              BEQ  SKIPPED   ; IF NOT, DONE.
ØA92:DØ ED              BNE  SKIP1     ; SUBSCRIPT COMMA, NEXT CHR
ØA94:C9 28     SKIP2    CMP  #LPAREN   ; LEFT PAREN?
ØA96:DØ Ø4              BNE  SKIP3     ; IF NO, CHECK FOR RIGHT
ØA98:E6 DF              INC  PARENCOUNT ; IF YES, ADD IT IN
ØA9A:DØ E5              BNE  SKIP1     ; NEXT CHR, BRANCH ALWAYS
ØA9C:C9 29     SKIP3    CMP  #RPAREN   ; ENDING PAREN?
ØA9E:DØ E1              BNE  SKIP1     ; NO, JUST A CHR. GET NEXT
ØAAØ:C6 DF              DEC  PARENCOUNT ; YES, SUBTRACT IT
ØAA2:1Ø DD              BPL  SKIP1     ; GET NEXT CHR
ØAA4:4C C9 DE           JMP  SYNERR    ; CRASH ON EXTRA LPAREN
ØAA7:2Ø 98 D9  SKIPPED  JSR  ADDON     ; UPDATE TXTPTR
ØAAA:18                 CLC            ; FORCE LEAVE WITH CARRY CLEAR
ØAAB:6Ø                 RTS
ØAAC:2Ø C3 Ø9  PUTCOLON JSR  SAVETXT   ; PUT A COLON
ØAAF:2Ø 7D ØA           JSR  SKIPVAR   ; AFTER THIS VARIABLE
ØAB2:A5 B8              LDA  TXTPTR    ; SAVE ITS PLACE
ØAB4:85 FA              STA  GENPTR    ; HERE FOR LATER
ØAB6:A5 B9              LDA  TXTPTR+1
ØAB8:85 FB              STA  GENPTR+1
ØABA:AØ ØØ              LDY  #Ø        ; FIND OUT WHAT WAS THERE
ØABC:B1 B8              LDA  (TXTPTR),Y
ØABE:8D 19 Ø8           STA  HOLDCOMMA ; SAVE IT
ØAC1:FØ Ø4              BEQ  COLONPUT  ; LEAVE ALONE IF EOL
ØAC3:A9 3A              LDA  #COLON
ØAC5:91 B8              STA  (TXTPTR),Y ; REPLACE THE COMMA (OR COLON)
ØAC7:4C 97 DB  COLONPUT JMP  GETTXT    ; RECOVER TXTPTR AND RTS
ØACA:AØ ØØ     REPCOLON LDY  #Ø        ; UNDO PUTCOLON
ØACC:AD 19 Ø8           LDA  HOLDCOMMA ; RECOVER COMMA, COLON, OR EOL
ØACF:91 FA              STA  (GENPTR),Y ; PUT IT BACK
ØAD1:6Ø                 RTS
ØAD2:2Ø CC Ø9  ADVANCEPTRS JSR POINTDEF ; MOVE THE DEF
ØAD5:2Ø 7D ØA           JSR  SKIPVAR   ; DEF AND CALL PTRS
ØAD8:2Ø DE Ø9           JSR  TXTTODEF  ; PAST THE CURRENT
ØADB:2Ø D5 Ø9           JSR  POINTCALL ; ENTRY IN EACH LIST
ØADE:2Ø 7D ØA           JSR  SKIPVAR   ; LEAVES WITH CARRY CLEAR
ØAE1:4C E7 Ø9           JMP  TXTTOCALL ; RTS FROM THERE
ØAE4:2Ø C3 Ø9  GETVARNAM JSR SAVETXT   ; EXIT WITH THIS TXTPTR
ØAE7:A2 ØØ              LDX  #Ø        ; MIMIC START OF PTRGET
ØAE9:86 1Ø              STX  DIMFLAG   ; GET THE VARIABLE NAME
ØAEB:86 11              STX  VALTYP    ; WITHOUT GETTING THE VARIABLE
ØAED:86 12              STX  INTFLAG   ; POINTER, AND THUS
ØAEF:86 14              STX  SUBFLAG   ; WITHOUT MAKING A NEW
ØAF1:86 DD              STX  ARYFLAG   ; ONE IF DOESN'T EXIST
ØAF3:86 DE              STX  EXPRFLAG  ; ALSO FLAG EXPRS AND ARRAYS
ØAF5:2Ø B1 ØØ           JSR  CHRGET    ; TXTPTR STARTS AT LEADING COMMA
ØAF8:2Ø 7D EØ           JSR  ISLETC    ; A LETTER?
ØAFB:9Ø 4C              BCC  GVEXPRSN  ; NO, LEAVE
ØAFD:85 81              STA  LASTVAR   ; YES, SAVE 1ST CHR
ØAFF:2Ø B1 ØØ           JSR  CHRGET    ; GET SECOND
ØBØ2:9Ø Ø5              BCC  NAME1     ; IF NUMBER, SAVE IT
ØBØ4:2Ø 7D EØ           JSR  ISLETC    ; LETTER?
ØBØ7:9Ø ØB              BCC  NAME3     ; NO, CHECK IF STRING, ETC
ØBØ9:AA        NAME1    TAX            ; SAVE 2ND CHR OF NAME
ØBØA:2Ø B1 ØØ  NAME2    JSR  CHRGET    ; SKIP REST OF LETTERS
ØBØD:9Ø FB              BCC  NAME2     ; AND NUMBERS. ALL EXCESS
ØBØF:2Ø 7D EØ           JSR  ISLETC    ; CHRS IN VAR NAME
ØB12:BØ F6              BCS  NAME2     ; SET IF LETTER
ØB14:C9 24     NAME3    CMP  #STRING   ; GOT A "$"?
ØB16:DØ Ø6              BNE  NAME4     ; IF NO, CHECK %
ØB18:A9 FF              LDA  #$FF      ; YES, FLAG IT
ØB1A:85 11              STA  VALTYP
ØB1C:DØ ØC              BNE  NAME5     ; ALWAYS TAKEN
ØB1E:C9 25     NAME4    CMP  #PERCENT  ; GOT AN INTEGER?
ØB2Ø:DØ ØF              BNE  NAME6     ; NO, MAYBE PAREN
ØB22:A9 8Ø              LDA  #$8Ø      ; YES, FLAG IT
ØB24:85 12              STA  INTFLAG
ØB26:Ø5 81              ORA  LASTVAR   ; AND CHANGE VARNAME
ØB28:85 81              STA  LASTVAR   ; TO REFLECT IT
ØB2A:8A        NAME5    TXA            ; NOW SET HIGH BIT OF
ØB2B:Ø9 8Ø              ORA  #$8Ø      ; 2ND CHR TO REFLECT
ØB2D:AA                 TAX            ; INTEGER OR STRING
ØB2E:          ;     NOTE ERROR, APPLESOFT REF MANUAL PAGE 137
ØB2E:          ;     STRINGS ARE + ON 1ST BYTE, - ON SECOND
```

LISTING 1: SUBR.MASTER *(continued)*

```
ØB2E:           ;      NOT THE REVERSE AS STATED THERE.
ØB2E:20 B1 ØØ              JSR  CHRGET     ; GET CHR AFTER % OR $
ØB31:86 82      NAME6      STX  LASTVAR+1  ; SAVE 2ND CHR OF NAME
ØB33:C9 28                 CMP  #LPAREN    ; GOT AN ARRAY?
ØB35:DØ Ø6                 BNE  NAME7      ; NO, CHECK IF EXPRSN
ØB37:A9 FF                 LDA  #$FF       ; ARRAY, SO FLAG IT
ØB39:85 DD                 STA  ARYFLAG
ØB3B:DØ 1Ø                 BNE  GOTVNAME   ; AND DONE
ØB3D:C9 2C      NAME7      CMP  #COMMA     ; END OF VAR?
ØB3F:FØ ØC                 BEQ  GOTVNAME   ; YES, DONE
ØB41:C9 ØØ                 CMP  #Ø         ; END OF LINE?
ØB43:FØ Ø8                 BEQ  GOTVNAME
ØB45:C9 3A                 CMP  #COLON
ØB47:FØ Ø4                 BEQ  GOTVNAME
ØB49:A9 FF      GVEXPRSN LDA #$FF          ; GOT EXPRESSION
ØB4B:85 DE                 STA  EXPRFLAG
ØB4D:4C 97 DB   GOTVNAME JMP GETTXT        ; RECOVER TXTPTR & RTS
ØB5Ø:A6 6B      FINDARY LDX ARYTAB         ; CHECK IF ARRAY
ØB52:A5 6C                 LDA  ARYTAB+1   ; OF NAME IN LASTVAR
ØB54:86 9B      FINDA1     STX  LOWTR      ; EXISTS IN MEMORY
ØB56:85 9C                 STA  LOWTR+1    ; IF SO, POINT TO IT
ØB58:C5 6E                 CMP  STREND+1   ; IN LOWTR
ØB5A:9Ø Ø7                 BCC  FINDA2     ; KEEP SEARCHING IF
ØB5C:E4 6D                 CPX  STREND     ; HAVEN'T PASSED END
ØB5E:9Ø Ø3                 BCC  FINDA2     ; OF MEMORY
ØB6Ø:AØ ØØ                 LDY  #Ø         ; FLAG NO ARRAY
ØB62:6Ø         FINDRTS RTS
ØB63:AØ ØØ      FINDA2     LDY  #Ø
ØB65:B1 9B                 LDA  (LOWTR),Y  ; NAME HERE MATCH
ØB67:C8                    INY             ; Y=1
ØB68:C5 81                 CMP  LASTVAR    ; ARRAY NAME?
ØB6A:DØ Ø6                 BNE  FINDA3     ; IF NO, NEXT ARRAY
ØB6C:B1 9B                 LDA  (LOWTR),Y  ; CHECK 2ND CHR
ØB6E:C5 82                 CMP  LASTVAR+1  ; OF NAMES
ØB7Ø:FØ FØ                 BEQ  FINDRTS    ; LEAVE ON MATCH
ØB72:C8         FINDA3     INY             ; (Y=2) GET OFFSET TO NEXT
ØB73:B1 9B                 LDA  (LOWTR),Y  ; ARRAY
ØB75:18                    CLC             ; ADD IT TO LOWTR
ØB76:65 9B                 ADC  LOWTR      ; TO POINT TO NEXT
ØB78:AA                    TAX             ; GOT LOW BYTE
ØB79:C8                    INY             ; (Y=3) GET HIGH BYTE
ØB7A:B1 9B                 LDA  (LOWTR),Y
ØB7C:65 9C                 ADC  LOWTR+1
ØB7E:9Ø D4                 BCC  FINDA1     ; BRANCH ALWAYS TAKEN
ØB8Ø:2Ø E4 ØA   MAKEVAR JSR GETVARNAM      ; WHAT TYPE OF VAR?
ØB83:A5 DD                 LDA  ARYFLAG    ; IF ARRAY
ØB85:DØ 35                 BNE  MAKEARRAY  ; DO BELOW
ØB87:A5 DE                 LDA  EXPRFLAG   ; IF EXPRESSION
ØB89:FØ Ø3                 BEQ  MAKESIMPLE ; THEN
ØB8B:4C C9 DE              JMP  SYNERR     ; CRASH
ØB8E:2Ø 38 ØA   MAKESIMPLE JSR SAVEARYTAB ; MAKE A LOCAL
ØB91:A5 69                 LDA  VARTAB     ; SIMPLE BY HIDING
ØB93:85 6B                 STA  ARYTAB     ; ALL THE OLD ONES
ØB95:A5 6A                 LDA  VARTAB+1   ; THUS CREATING A
ØB97:85 6C                 STA  ARYTAB+1   ; NEW ONE AT THE
ØB99:2Ø B1 ØØ              JSR  CHRGET     ; BOTTOM OF STORAGE. PASS
ØB9C:2Ø E3 DF              JSR  PTRGET     ; LEADING COMMA.
ØB9F:AD ØF Ø8              LDA  OLDARYTAB  ; RECOVER ARRAYS
ØBA2:18                    CLC             ; TO REFLECT NEW
ØBA3:69 Ø7                 ADC  #7         ; SIMPLE IN MEMORY.
ØBA5:85 6B                 STA  ARYTAB     ; UPDATE ARRAY PTR
ØBA7:AD 1Ø Ø8              LDA  OLDARYTAB+1
ØBAA:69 ØØ                 ADC  #Ø         ; ADD IN CARRY
ØBAC:85 6C                 STA  ARYTAB+1
ØBAE:AD 15 Ø8              LDA  OLDSIMPLE  ; UPDATE POINTER
ØBB1:69 Ø7                 ADC  #7         ; (CARRY CLEAR) TO START
ØBB3:8D 15 Ø8              STA  OLDSIMPLE  ; OF ORIGINAL
ØBB6:9Ø Ø3                 BCC  SIMPLERTS  ; SIMPLE VARIABLES
ØBB8:EE 16 Ø8              INC  OLDSIMPLE+1
ØBBB:6Ø         SIMPLERTS RTS
ØBBC:2Ø AC ØA   MAKEARRAY JSR PUTCOLON     ; ONLY DIM THIS ARRAY
ØBBF:2Ø 5Ø ØB              JSR  FINDARY    ; NOW DOES THIS ARRAY EXIST?
ØBC2:98                    TYA             ; IF SO, Y NOT Ø. IF EXISTS,
ØBC3:DØ Ø9                 BNE  MAKEAR1    ; MAKE NEW ARRAY, SAME NAME
ØBC5:2Ø B1 ØØ              JSR  CHRGET     ; NO ARRAY OF THIS NAME EXISTS
ØBC8:2Ø D9 DF              JSR  DIM        ; MAKE ONE NOW
ØBCB:4C CA ØA              JMP  REPCOLON   ; FIX LINE AND LEAVE
ØBCE:2Ø 38 ØA   MAKEAR1    JSR  SAVEARYTAB ; ARRAY WITH THIS NAME
ØBD1:2Ø 4E ØA              JSR  SAVESTREND ; EXISTS. TO MAKE A NEW ONE,
ØBD4:85 6C                 STA  ARYTAB+1   ; HIDE ALL OLD ARRAYS BY
ØBD6:A5 6D                 LDA  STREND     ; CALLING THEM SIMPLE VARS.
ØBD8:85 6B                 STA  ARYTAB     ; THEN MAKE NEW ARRAY AT THE
ØBDA:2Ø B1 ØØ              JSR  CHRGET     ; (NEEDED FOR DIM)
ØBDD:2Ø D9 DF              JSR  DIM        ; TOP OF MEMORY.
ØBEØ:2Ø 97 DB              JSR  GETTXT     ; RECOVER TXTPTR
ØBE3:AD ØF Ø8              LDA  OLDARYTAB  ; SET UP BLTU
ØBE6:85 9B                 STA  LOWTR      ; TO TRANSFER ARRAYS.
ØBE8:AD 1Ø Ø8              LDA  OLDARYTAB+1 ; TRANSFER THE OLD ARRAYS
ØBEB:85 9C                 STA  LOWTR+1    ; FROM ARYTAB THROUGH OLDSTREND
ØBED:AD 13 Ø8              LDA  OLDSTREND  ; UP IN MEMORY ENDING AT THE
ØBFØ:18                    CLC             ; NEW STREND, OVERWRITING THE
ØBF1:69 Ø1                 ADC  #1         ; NEW ARRAY.
ØBF3:85 96                 STA  HIGHTR     ; USE BLTUP NOT BLTU
ØBF5:AD 14 Ø8              LDA  OLDSTREND+1 ; TO DO THE MOVE AS WE
ØBF8:69 ØØ                 ADC  #Ø         ; KNOW STREND IS OK (NO NEED FOR
ØBFA:85 97                 STA  HIGHTR+1   ; 'REASON' ROUTINE) AND DON'T
```

the array, but it can take longer. If another array with the DEF name already exists and is located lower in memory than the CALL array to be renamed, the CALL array is moved below it. No variable space is used when renaming or moving the arrays.

Integer and real array data takes different amounts of memory. If array R() is renamed as I%() and you reference I%(30), you will not get the value of R(30). We considered adding the code needed to define a local R() for the subroutine, to convert I%()'s values to real, and to pass them to R(), but this would be slow and would waste the space taken up by R(). We were concerned that automating the practice would encourage it, and decided not to. Instead, the program flags an attempted pass of this sort with a TYPE MISMATCH error.

The CALL array's subscripts must be legal. If the subscript is too large, given the DIMension of the array, or if there are too many dimensions, the program halts with the usual BAD SUBSCRIPT message. Otherwise, the values of the subscripts are immaterial. The whole array is passed to the DEF list array. The subscripts of the DEF list arrays are not checked in any way. Subscripts are not even necessary — D() will do the job in the DEF statement. The parentheses specify that we're dealing with an array, and the D specifies that it is a real array named D. The dimension assigned to D() is the actual dimension of the CALL list array. If the subroutine tries to use D() with a bad subscript for the passed CALL array, BASIC will flag this.

The requirement that the CALL array must have been dimensioned uses extra code, but it adds protection against various types of errors. If the CALL array was not DIMensioned before our handler tries to pass it to the DEF array, the program halts with an ARRAY ERROR message. Our main concern in adding this reflects our feeling that arrays should always be DIMensioned explicitly. Traditional BASIC does the programmer a "favor" in allowing various types of sloppy coding practices, including this one. We'd rather be spared the favor, and the errors we've missed finding because of it.

Preferences aside, this provides protection against nesting problems that can arise if you accidentally have one too many NEXTs or RETURNs in your subroutine. If there is an active FOR or GOSUB outside the subroutine, one of these may pull you back to it, without an error message, but also without properly exiting the called subroutine. (We'll describe the problem in more detail below.) Here is the basic idea: suppose that you do somehow get back to the calling program without exiting from this subroutine. If you passed an array down, it is renamed for the subroutine. Without the EXIT, it is not renamed back. The next time you call the routine, no array with the name in the CALL statement will exist, forcing a program halt with an error message. This tech-

LISTING 1: SUBR.MASTER *(continued)*

```
ØBFC:A5 6D              LDA  STREND      ; WANT ANY PTRS TINKERED WITH.
ØBFE:69 Ø1              ADC  #1
ØCØØ:85 94              STA  HIGHDS
ØCØ2:2Ø 4E ØA           JSR  SAVESTREND  ; RETURNS HOLDING STREND+1
ØCØ5:69 ØØ              ADC  #Ø          ; WITH CARRY UNAFFECTED
ØCØ7:85 95              STA  HIGHDS+1
ØCØ9:2Ø 9A D3           JSR  BLTUP       ; ACTUAL MOVE HERE.
ØCØC:2Ø 43 ØA           JSR  GETARYTAB   ; NOW HIDE THE ARRAYS AGAIN
ØCØF:85 6E              STA  STREND+1    ; (=ARYTAB+1). THIS TIME THE
ØC11:A5 6B              LDA  ARYTAB      ; DIM CREATES THE ARRAY AT THE
ØC13:85 6D              STA  STREND      ; BOTTOM OF ARRAY STORAGE, SO
ØC15:2Ø B1 ØØ           JSR  CHRGET      ; IT IS ALWAYS FOUND FIRST.
ØC18:2Ø D9 DF           JSR  DIM         ; THE MOVE ABOVE MADE ROOM HERE
ØC1B:2Ø 59 ØA           JSR  GETSTREND   ; FOR THIS.  RECOVER CORRECT
ØC1E:2Ø CA ØA           JSR  REPCOLON    ; STREND, FIX LINE,
ØC21:6Ø         RTS1    RTS              ; AND DONE.
ØC22:A2 Ø2      RENAME  LDX  #2          ; PUT NEW NAMES IN OLD ARRAYS,
ØC24:E4 DA      RENAME1 CPX  BUFPTR      ; WHILE ARRAYS LEFT TO DO.
ØC26:BØ F9              BCS  RTS1        ; THEN EXIT
ØC28:BD 1A Ø8           LDA  SECBUF,X    ; LOCATION OLD ARRAY
ØC2B:85 9B              STA  LOWTR       ; STASH IT
ØC2D:BD 1B Ø8           LDA  SECBUF+1,X
ØC3Ø:85 9C              STA  LOWTR+1
ØC32:AØ ØØ              LDY  #Ø          ; INDIRECT ADDRESSING
ØC34:BD 18 Ø8           LDA  SECBUF-2,X  ; NEW NAME
ØC37:91 9B              STA  (LOWTR),Y   ; FOR THE ARRAY
ØC39:BD 19 Ø8           LDA  SECBUF-1,X
ØC3C:C8                 INY
ØC3D:91 9B              STA  (LOWTR),Y
ØC3F:8A                 TXA
ØC4Ø:69 Ø4              ADC  #4          ; CLEAR CARRY FROM BCS ABOVE
ØC42:AA                 TAX
ØC43:9Ø DF              BCC  RENAME1     ; MUST STILL BE CLEAR
ØC45:           ************************************
ØC45:           *       CALL THIS ADDRESS          *
ØC45:           *       TO ENTER A PROC            *
ØC45:           ************************************
ØC45:2Ø AA Ø9   PROC    JSR  IN          ; CHECK MODE. SAVE FA-FF.
ØC48:A5 B8              LDA  TXTPTR      ; CALL VARLIST STARTS
ØC4A:8D Ø9 Ø8           STA  CALLIST     ; AT COMMA (OR EOL IF NO LIST)
ØC4D:A5 B9              LDA  TXTPTR+1    ; AFTER PROC NAME, WHICH
ØC4F:8D ØA Ø8           STA  CALLIST+1   ; IS WHERE TXTPTR IS AT.
ØC52:AØ ØØ              LDY  #Ø          ; FOR INDIRECT ADDRESS
ØC54:2Ø 19 ØA   FINDNAME JSR DECTXT      ; MOVE TXTPTR BACK
ØC57:B1 B8              LDA  (TXTPTR),Y  ; TO FIND "CALL"
ØC59:C9 8C              CMP  #CALL       ; NAME OF THE PROC
ØC5B:DØ F7              BNE  FINDNAME    ; STARTS THERE.
ØC5D:2Ø E7 Ø9           JSR  TXTTOCALL   ; POINT TO IT IN CALLPTR
ØC6Ø:8D ØE Ø8           STA  PROCNAME+1  ; AND IN PTR TO
ØC63:A5 B8              LDA  TXTPTR      ; THE NAME ITSELF.
ØC65:8D ØD Ø8           STA  PROCNAME
ØC68:2Ø 97 D6           JSR  STXTPT      ; TXTPTR AT START OF PROG
ØC6B:AØ Ø2      FINDDEF LDY  #2          ; HIGH BYTE OF
ØC6D:B1 B8              LDA  (TXTPTR),Y  ; NEXT LINE'S ADDRESS
ØC6F:DØ Ø5              BNE  FINDDEF1    ; IF Ø, NO PROG LEFT
ØC71:A2 5A      UNDEF   LDX  #9Ø         ; IN WHICH CASE MAKE
ØC73:4C 12 D4           JMP  ERROR       ; UNDEFINED STATEMENT ERR.
ØC76:C8         FINDDEF1 INY             ; Y=3. LOW BYTE OF
ØC77:B1 B8              LDA  (TXTPTR),Y  ; NEW LINE #
ØC79:8D ØB Ø8           STA  DEFLINE     ; SAVE IT
ØC7C:C8                 INY              ; Y=4
ØC7D:B1 B8              LDA  (TXTPTR),Y  ; HIGH BYTE
ØC7F:8D ØC Ø8           STA  DEFLINE+1   ; GOT IT
ØC82:C8                 INY              ; FIRST CHR OF TEXT
ØC83:B1 B8              LDA  (TXTPTR),Y  ; IS IT "DEF"?
ØC85:C9 B8              CMP  #DEF
ØC87:DØ 16              BNE  NEXTLINE    ; IF NOT, TRY NEXT LINE
ØC89:2Ø 98 D9           JSR  ADDON       ; POINT TO IT WITH TXTPTR
ØC8C:AØ ØØ              LDY  #Ø          ; Y INDEXES DEF AND CALL NAMES
ØC8E:C8         NEXTCHAR INY             ; PAST "DEF" & "CALL"
ØC8F:B1 FC              LDA  (CALLPTR),Y ; GET NEXT CHAR OF NAME
ØC91:FØ 17              BEQ  CNAMDONE    ; CALL NAME ENDS ON Ø
ØC93:C9 3A              CMP  #COLON      ; OR ":" OR ","
ØC95:FØ 13              BEQ  CNAMDONE    ; IF END, CHECK IF
ØC97:C9 2C              CMP  #COMMA      ; DEF NAME DONE TOO.
ØC99:FØ ØF              BEQ  CNAMDONE    ; IF GET PAST HERE, THEN
ØC9B:D1 B8              CMP  (TXTPTR),Y  ; STILL IN NAME. CHECK DEF.
ØC9D:FØ EF              BEQ  NEXTCHAR    ; BOTH MATCH, CHECK NEXT CHR
ØC9F:2Ø 98 D9   NEXTLINE JSR ADDON       ; DEF & CALL MISMATCH. MOVE
ØCA2:2Ø A6 D9           JSR  REMN        ; PAST LINE #, THEN PAST LINE
ØCA5:2Ø 98 D9           JSR  ADDON       ; SET TXTPTR TO NEXT LINE
ØCA8:DØ C1              BNE  FINDDEF     ; ALWAYS TAKEN. TRY AGAIN
ØCAA:D1 B8      CNAMDONE CMP (TXTPTR),Y  ; DEF NAME DONE TOO?
ØCAC:FØ ØC              BEQ  FOUNDIT     ; YES. DONE SEARCHING
ØCAE:C9 2C              CMP  #COMMA      ; IF COMMA & DEF NOT COMMA
ØCBØ:FØ ED              BEQ  NEXTLINE    ; MISMATCH OR BAD CALL
ØCB2:B1 B8              LDA  (TXTPTR),Y  ; CALL IS END OF LINE
ØCB4:FØ Ø4              BEQ  FOUNDIT     ; MATCH IF DEF IS TOO
ØCB6:C9 3A              CMP  #COLON
ØCB8:DØ E5              BNE  NEXTLINE
ØCBA:2Ø 98 D9   FOUNDIT JSR  ADDON       ; TXTPTR POINTS TO COMMA
ØCBD:A5 B8              LDA  TXTPTR      ; AT START OF DEF LIST
ØCBF:8D Ø7 Ø8           STA  DEFLIST     ; SAVE THE POINTER
ØCC2:A5 B9              LDA  TXTPTR+1    ; IN PTR TO START OF
ØCC4:8D Ø8 Ø8           STA  DEFLIST+1   ; DEF VAR LIST
ØCC7:2Ø FØ Ø9           JSR  STARTLIST   ; MOVE PTRS TO
```

nique will not always catch bad nesting, but it is another level of safeguarding.

Nesting

You can "nest" subroutines to your heart's content or until the Apple runs out of memory, whichever comes first. If one subroutine calls a second, the second is "nested" within the first. If the second calls a third, you've added another level of nesting. Each level of nesting has a memory cost associated with it, which disappears on exit. There is a basic cost of 21 bytes per level of nesting (for pointers, etc.), plus seven bytes per simple variable passed, plus however much is required for variables declared LOCAL. Note that locals from a calling subroutine are globals to the called one, as in Pascal.

Speed

Calling and exiting subroutines takes a variable length of time, depending on how many variables are passed, how many locals are created, and how many variables are already in memory. The dominant factor is the number of bytes taken up by simple and array storage. You can estimate how long a CALL or EXIT will take by determining how many bytes are taken up by variable storage (subtract VARTAB, in $69,$6A, from STREND, in $6D,$6E, for this) and how many local simple variables are created during the CALL. Each byte takes about 17.5 microseconds to move, and each is moved whenever a simple variable is created or cleared. A minimum of three simple variables are created per CALL and cleared per EXIT, for housekeeping. The move is a bit faster when lots of data is transferred, and a bit slower per byte when very few bytes are moved, but this is a good ballpark figure, even though it ignores local array handling. In practical terms, if there is very little data, a CALL–EXIT pair takes about 0.2 seconds to execute. If memory is nearly full (say, 25,000 bytes of data), each CALL–EXIT pair takes a minimum of 2.5 seconds.

ERROR HANDLING

Detected Errors

Whenever possible, we rely on Applesoft to detect errors, either when our handler uses Applesoft internal routines, or within the subroutine itself. For example, we don't check if a DEF array has the right number of dimensions. If it has the wrong number for the CALL array being passed to it, Applesoft will halt the program as soon as that DEF array is used in the subroutine.

A number of further errors, which we have to catch, can arise relating to our subroutines themselves. For these we either use standard Applesoft error messages or, in two cases, parts of them: ARRAY ERROR and MEMORY ERROR. If you get either of these, you know the program crashed while executing a CALL or an EXIT.

An ARRAY ERROR occurs in response

to various errors involving array passing. For example, if a simple variable is passed to a DEF array, if an attempt to pass a previously undefined array to a subroutine is detected, or if a DEF array is part of an expression, you get an ARRAY ERROR message.

A MEMORY ERROR occurs when the pointer to the end of variable storage (STREND) doesn't have the value at a certain point during the EXIT that it had at a comparable point during the CALL. Usually this means that you created a new global or cleared an old one within the subroutine. It also signals crossed subroutines and, generally, an EXIT from a subroutine with a different name from the one called.

An UNDEF'D STATEMENT error indicates that the handler can't find a subroutine of the name you called. This happens most frequently when you miss a comma after the subroutine name following the CALL or DEF. CALL SRTX(0),10 will not lead you to DEF SRT,S(0),N.

CALL and DEF parameter lists that have a different number of items may be flagged in a number of ways. If the CALL or the DEF statement is followed by no list (and no comma), while the other is followed by a parameter list, you get an UNDEF'D STATEMENT error. If both statements have a parameter list but one list has more items than the other, you get a SYNTAX ERROR, either on entry to the routine (CALL list short) or on exit (DEF list short) — if you don't get a MEMORY ERROR first.

Undetected Errors

Some errors are not trapped, since we consider the cure worse than the disease. Most of these errors are very unlikely to occur, or generally harmless. Further, many of them are eventually caught when BASIC or the handler doesn't understand something later in the program. However, we'll describe these errors as if they are never caught, and consider the "worst case" behavior of the program. Our intention is to clearly discuss the error handling problems you may run into, and what to do about them if you do. (Don't be scared off, though — in practice, we've found this program to be extremely reliable.)

Expressions in the DEF Variable List — If you have an expression in the DEF list that starts with an array and ends with a parenthesis (but didn't start with a parenthesis), it must start with the array name itself. If the corresponding CALL variable is also an array, then and only then, the handler will not crash on an expression in the list. Instead it will pass the CALL array to the DEF array and back to EXIT, completely ignoring the expression code between the array name and the next comma.

LISTING 1: SUBR.MASTER *(continued)*

```
ØCCA:A5 69               LDA  VARTAB     ; START OF VAR LISTS.
ØCCC:8D 15 Ø8            STA  OLDSIMPLE  ; THEN SAVE PTR TO
ØCCF:A5 6A               LDA  VARTAB+1   ; START OF ENTERING
ØCD1:8D 16 Ø8            STA  OLDSIMPLE+1 ; SIMPLE VARIABLES
ØCD4:20 CC Ø9  PASSIMPLE JSR POINTDEF    ; PNT TO NEXT DEF LIST VAR
ØCD7:20 B7 ØØ            JSR  CHRGOT     ; WHILE NOT EOL PASS SIMPLES.
ØCDA:FØ 30               BEQ  PASSARY    ; WHEN DONE, PASS THE ARRAYS
ØCDC:20 E4 ØA            JSR  GETVARNAM  ; SIMPLE VAR?
ØCDF:A5 DE               LDA  EXPRFLAG   ; NO EXPRS IN DEFS
ØCE1:FØ Ø6               BEQ  SIMPLE1    ; CHECK IF ARRAY BELOW
ØCE3:20 72 ØA            JSR  DEFTOCUR   ; CRASH ON EXPR
ØCE6:4C C9 DE            JMP  SYNERR     ; IN DEF STATEMENT
ØCE9:A5 DD     SIMPLE1   LDA  ARYFLAG    ; ARRAY?
ØCEB:DØ 1A               BNE  NEXTSIMPLE ; IF SO, SKIP IT
ØCED:20 8E ØB            JSR  MAKESIMPLE ; CREATE LOCAL WITH
ØCFØ:20 D5 Ø9            JSR  POINTCALL  ; FIND VAR IN CALL LIST
ØCF3:20 B1 ØØ            JSR  CHRGET     ; MOVE PAST LEADING COMMA
ØCF6:20 64 ØA            JSR  SIMPTOVAR  ; LOOK PAST LOCAL SIMPLES
ØCF9:A5 83               LDA  VARPNT     ; SET UP THESE PTRS FOR "LET"
ØCFB:85 85               STA  FORPNT     ; THEY WERE SET UP BY THE PTRGET
ØCFD:A5 84               LDA  VARPNT+1   ; CALL IN MAKESIMPLE.
ØCFF:85 86               STA  FORPNT+1   ; GO PARTWAY INTO LET TO
ØDØ1:20 52 DA            JSR  LETCNT     ; SKIP THE "=" TEST THERE.
ØDØ4:20 2D ØA            JSR  GETVARTAB  ; RECOVER TRUE VARTAB
ØDØ7:20 D2 ØA  NEXTSIMPLE JSR ADVANCEPTRS ; UPDATE CALL & DEF PTRS
ØDØA:90 C8               BCC  PASSIMPLE  ; ALWAYS CLEAR
ØDØC:20 FØ Ø9  PASSARY   JSR  STARTLIST  ; RECOVER LIST PTRS
ØDØF:A9 ØØ               LDA  #Ø         ; INITIALIZE COUNTERS
ØD11:85 DA               STA  BUFPTR
ØD13:85 DB               STA  COUNTER
ØD15:20 CC Ø9  ARRAY1    JSR  POINTDEF   ; GET 1ST VAR IN DEF
ØD18:20 B7 ØØ            JSR  CHRGOT     ; WHILE NOT EOL, PASS ARRAYS
ØD1B:DØ Ø3               BNE  ARRAY2
ØD1D:4C AF ØD            JMP  FIND       ; WHEN DONE, RENAME & FIND THEM
ØD2Ø:20 E4 ØA  ARRAY2    JSR  GETVARNAM  ; SIMPLE VAR?
ØD23:A5 DD               LDA  ARYFLAG    ; NOT IF THIS NOT ZERO
ØD25:DØ Ø5               BNE  ARRAY3     ; SO PASS IT
ØD27:20 D2 ØA            JSR  ADVANCEPTRS ; SIMPLE, SO SKIP IT
ØD2A:90 E9               BCC  ARRAY1     ; LOOK AT NEXT VAR
ØD2C:A5 81     ARRAY3    LDA  LASTVAR    ; STORE NAME
ØD2E:A6 DA               LDX  BUFPTR     ; IN SECBUF
ØD3Ø:9D 1A Ø8            STA  SECBUF,X
ØD33:29 8Ø               AND  #$8Ø       ; STORE TYPE
ØD35:9D 1C Ø8            STA  SECBUF+2,X ; HERE TEMPORARILY
ØD38:A5 82               LDA  LASTVAR+1  ; NAME HIGH BYTE
ØD3A:9D 1B Ø8            STA  SECBUF+1,X
ØD3D:29 8Ø               AND  #$8Ø
ØD3F:9D 1D Ø8            STA  SECBUF+3,X
ØD42:20 D5 Ø9            JSR  POINTCALL  ; FIND CALL ARRAY
ØD45:20 E4 ØA            JSR  GETVARNAM  ; GOT ARRAY?
ØD48:A5 DD               LDA  ARYFLAG    ; MUST BE FF
ØD4A:DØ Ø5               BNE  ARYCHK     ; OR CRASH WITH
ØD4C:A2 8Ø     ARRAYERR  LDX  #128       ; "ARRAY ERROR"
ØD4E:4C 12 D4            JMP  ERROR      ; AS PASSING SIMPLE TO ARRAY
ØD51:20 5Ø ØB  ARYCHK    JSR  FINDARY    ; WHERE IS IT?
ØD54:98                  TYA             ; REQUIRE THAT IT EXISTS ALREADY
ØD55:FØ F5               BEQ  ARRAYERR   ; REFUSE TO PASS UNDIMENSIONED ARRAY.
ØD57:20 D5 Ø9            JSR  POINTCALL  ; NOW SEE IF ARRAY EXPRESSION
ØD5A:20 B1 ØØ            JSR  CHRGET     ; MUST HAVE COMMA OR EOL AFTER
ØD5D:20 E3 DF            JSR  PTRGET     ; ARRAY NAME.
ØD6Ø:20 B7 ØØ            JSR  CHRGOT     ; DO WE?
ØD63:FØ Ø4               BEQ  ITISARRAY
ØD65:C9 2C               CMP  #COMMA
ØD67:DØ E3               BNE  ARRAYERR
ØD69:20 D2 ØA  ITISARRAY JSR ADVANCEPTRS ; UPDATE THE LIST PTRS
ØD6C:20 CC Ø9            JSR  POINTDEF   ; NOW CHECK THAT DEF ARRAY
ØD6F:20 19 ØA            JSR  DECTXT     ; WAS NOT AN EXPRESSION WITH
ØD72:20 B7 ØØ            JSR  CHRGOT     ; A LEADING ARRAY IN IT
ØD75:C9 29               CMP  #RPAREN    ; SUCH AS D(1)*5
ØD77:FØ Ø6               BEQ  BOTHOK     ; THIS CHECKS THAT LAST CHAR
ØD79:20 72 ØA            JSR  DEFTOCUR   ; OF DEF VAR IS ')'.
ØD7C:4C 4C ØD            JMP  ARRAYERR
ØD7F:A6 DA     BOTHOK    LDX  BUFPTR     ; WHERE WERE WE?
ØD81:A5 81               LDA  LASTVAR    ; COMPARE TYPES OF VARS
ØD83:29 8Ø               AND  #$8Ø       ; MASK ALL BUT TYPE FLAG
ØD85:DD 1C Ø8            CMP  SECBUF+2,X ; STORED HERE FOR DEF VAR
ØD88:DØ ØE               BNE  BADTYPE    ; CRASH IF NOT SAME
ØD8A:A5 9B               LDA  LOWTR      ; OVERWRITE TYPE WITH
ØD8C:9D 1C Ø8            STA  SECBUF+2,X ; ADDRESS OF CALL VAR
ØD8F:A5 82               LDA  LASTVAR+1  ; DO SAME FOR HIGH BYTE
ØD91:29 8Ø               AND  #$8Ø
ØD93:DD 1D Ø8            CMP  SECBUF+3,X
ØD96:FØ Ø3               BEQ  ARRAY4
ØD98:4C 76 DD  BADTYPE   JMP  MISMATCH
ØD9B:A5 9C     ARRAY4    LDA  LOWTR+1
ØD9D:9D 1D Ø8            STA  SECBUF+3,X
ØDAØ:8A                  TXA             ; UPDATE BUFPTR
ØDA1:69 Ø3               ADC  #3         ; CARRY SET, THIS ADDS 4
ØDA3:85 DA               STA  BUFPTR
ØDA5:C9 DØ               CMP  #BUFMAX    ; PAST THE MAX # VARS?
ØDA7:BØ Ø3               BCS  TOOMANY    ; IF SO, CRASH OUT OF MEMORY
ØDA9:4C 15 ØD            JMP  ARRAY1     ; ELSE, DO NEXT ARRAY
ØDAC:4C 1Ø D4  TOOMANY   JMP  OMERR
ØDAF:20 22 ØC  FIND      JSR  RENAME     ; RENAME THE ARRAYS
ØDB2:A2 Ø2               LDX  #2         ; SEARCH FOR NEWLY RENAMED ARRAYS
ØDB4:E4 DA     FIND1     CPX  BUFPTR     ; WHILE ARRAYS LEFT TO CHECK
```

Missed Commas in the CALL and DEF Lists — If you miss the comma after the subroutine name in both lists, and if the first variable in each list is simple and both have the same name, the handler will think that that variable is part of the name and won't realize you missed the comma. Since the variable passed is an existing global, the subroutine will execute correctly. If you don't miss the comma in your next CALL, however, you may wonder why the program didn't crash on the first one.

Wrong EXIT Subroutine Name — If you CALL SRT and try to exit from STR, the program will crash, as it should. But suppose you CALL SUB2 and CALL EXIT, SUB1. In this case, since SUB1 and SUB2 are the same variable to BASIC, you will not get an error message. Instead, you will exit from SUB2 normally. This is no problem unless you compound the error. If subroutine SUB1 includes a CALL to SUB2, which in turn tries to EXIT SUB1 (lots of GOTOs in a program could put you in this position), then EXIT will behave just like a RETURN would and take you out of the last subroutine called, i.e., SUB2. Again, this can only happen if two routines have the same first two letters, and if one calls the other directly, without a third one between them.

Invalid User-Defined Functions — If you DEF FN A(X) in the main program and try to do anything with it in the subroutine, the computer will respond with the wrong answer but no error message.

Wrong Value for Exit or the Subroutine Name — If you CALL SRT, and SRT is not 3141; or CALL EXIT, SRT, and EXIT is not 4058, BASIC will transfer control to the wrong location in memory. The typical case is that you forget to define one of these variables before calling it. In this case, you CALL 0, which works like an END statement: a halt with no error message. If the variable is non-zero but wrong, you'll likely crash on an error test in our routine, or crash with a Monitor break on a stored zero. But anything is possible.

Immediate Mode GOSUB — You can only call subroutines in program execution mode because we use the keyboard input buffer at $200 to move arrays. Immediate mode GOSUBs may cause data to be scrambled without any error message. Therefore, you should only use Subroutine Master from a running program.

Crossing GOSUB or FOR With Called Subroutines — If you call a subroutine from within a GOSUB subroutine, and the called subroutine contains one too many RETURN statements, then you will return to the outer GOSUB without getting a RETURN WITHOUT GOSUB error message and

LISTING 1: SUBR.MASTER *(continued)*

```
ØDB6:BØ 38              BCS  SORT       ; THEN SORT THOSE TO MOVE
ØDB8:BD 18 Ø8           LDA  SECBUF-2,X ; GET THE NAME
ØDBB:85 81              STA  LASTVAR    ; PUT IT HERE
ØDBD:BD 19 Ø8           LDA  SECBUF-1,X ; TO FIND IT
ØDCØ:85 82              STA  LASTVAR+1
ØDC2:86 DC              STX  COUNTER+1  ; AUXILIARY COUNTER
ØDC4:2Ø 5Ø ØB           JSR  FINDARY    ; WHERE'S THE ARRAY?
ØDC7:A6 DC              LDX  COUNTER+1  ; RECOVER X
ØDC9:A5 9C              LDA  LOWTR+1    ; ARRAY FOUND STARTS HERE
ØDCB:DD 1B Ø8           CMP  SECBUF+1,X ; SAME AS RENAMED ONE?
ØDCE:DØ Ø7              BNE  MUSTMOVE   ; IF NOT, HAVE TO MOVE
ØDDØ:A5 9B              LDA  LOWTR      ; THE NEW ONE DOWN IN
ØDD2:DD 1A Ø8           CMP  SECBUF,X   ; MEMORY, TO HIDE THIS ONE.
ØDD5:FØ 12              BEQ  FIND3      ; SAME ARRAY, CHECK NEXT
ØDD7:A4 DB     MUSTMOVE LDY  COUNTER    ; INDEX BUFR
ØDD9:BD 1A Ø8           LDA  SECBUF,X   ; RECOVER ADDRESS OF
ØDDC:99 ØØ Ø2           STA  BUFR,Y     ; RENAMED ARRAY
ØDDF:BD 1B Ø8           LDA  SECBUF+1,X ; AND SAVE IT FOR SORTING
ØDE2:99 Ø1 Ø2           STA  BUFR+1,Y
ØDE5:C8                 INY
ØDE6:C8                 INY
ØDE7:84 DB              STY  COUNTER    ; POINT TO NEXT FREE SPOT
ØDE9:8A        FIND3    TXA             ; UPDATE BUFFER PTR
ØDEA:18                 CLC
ØDEB:69 Ø4              ADC  #4
ØDED:AA                 TAX
ØDEE:DØ C4              BNE  FIND1      ; ALWAYS TAKEN
ØDFØ:A4 DB     SORT     LDY  COUNTER    ; WHILE ARRAYS TO SORT, DO
ØDF2:DØ Ø3              BNE  SORT1      ; ELSE DONE PASSING.  NOTE NON-Ø
ØDF4:4C FA ØE           JMP  LOCAL      ; COUNTER PTS 1 PAST END OF LIST
ØDF7:          ;
ØDF7:          ;      WE WILL MOVE THE ARRAYS IN ORDER FROM
ØDF7:          ;      LOWEST IN MEMORY TO HIGHEST.  THIS IS FASTER
ØDF7:          ;      AND DOES NOT INTERFERE WITH THE ADDRESSES
ØDF7:          ;      STORED FOR THE VARIABLES TO BE MOVED.
ØDF7:          ;
ØDF7:AØ FE     SORT1    LDY  #$FE       ; START AT Ø AFTER INY'S
ØDF9:C8        NEXTY    INY             ; FOR Y=FIRST TO LAST-1 ARRAY
ØDFA:C8                 INY             ; (ARRAY PTRS TAKE 2 BYTES)
ØDFB:C4 DB              CPY  COUNTER    ; WHILE ARRAYS LEFT, SORT
ØDFD:BØ 41              BCS  SORTED     ; THEN MOVE THEM
ØDFF:98                 TYA             ; FOR X = Y+1TH TO LAST ARRAY
ØEØØ:AA                 TAX
ØEØ1:E8        NEXTX    INX             ; IF ADDRESS(Y) > ADDRESS(X)
ØEØ2:E8                 INX             ; THEN SWITCH ORDER OF ADDRESSES
ØEØ3:E4 DB              CPX  COUNTER    ; ANY LEFT IN LIST?
ØEØ5:BØ F2              BCS  NEXTY      ; IF NO, DONE INSIDE LOOP
ØEØ7:B9 Ø1 Ø2           LDA  BUFR+1,Y   ; COMPARE HIGH BYTES
ØEØA:DD Ø1 Ø2           CMP  BUFR+1,X
ØEØD:9Ø F2              BCC  NEXTX      ; ADDRESS(Y) < ADDRESS(X)
ØEØF:DØ ØD              BNE  SWITCH     ; ADDRESS(Y) > ADDRESS(X)
ØE11:B9 ØØ Ø2           LDA  BUFR,Y     ; CHECK LOW BYTES
ØE14:DD ØØ Ø2           CMP  BUFR,X
ØE17:9Ø E8              BCC  NEXTX
ØE19:DØ Ø3              BNE  SWITCH
ØE1B:4C 4C ØD           JMP  ARRAYERR   ; FLAG ALIASED ARRAYS WHEN SPOTTED.
ØE1E:B9 ØØ Ø2  SWITCH   LDA  BUFR,Y     ; STASH ADDRESS(Y)
ØE21:85 FA              STA  GENPTR     ; HERE, TEMPORARILY
ØE23:B9 Ø1 Ø2           LDA  BUFR+1,Y
ØE26:85 FB              STA  GENPTR+1
ØE28:BD ØØ Ø2           LDA  BUFR,X     ; NOW SET ADDRESS(Y)
ØE2B:99 ØØ Ø2           STA  BUFR,Y     ; TO ADDRESS(X)
ØE2E:BD Ø1 Ø2           LDA  BUFR+1,X
ØE31:99 Ø1 Ø2           STA  BUFR+1,Y
ØE34:A5 FA              LDA  GENPTR     ; AND SET ADDRESS(X)
ØE36:9D ØØ Ø2           STA  BUFR,X     ; TO OLD ADDRESS(Y)
ØE39:A5 FB              LDA  GENPTR+1
ØE3B:9D Ø1 Ø2           STA  BUFR+1,X
ØE3E:BØ C1              BCS  NEXTX      ; SET ENTERING SWITCH
ØE4Ø:BD ØØ Ø2  SORTED   LDA  BUFR,X     ; NOW MOVE SORTED ADDRESSES
ØE43:9D 1A Ø8           STA  SECBUF,X   ; BACK UP TO SECBUF. SINCE
ØE46:CA                 DEX             ; THIS LIST IS HALF OLD SECBUF
ØE47:1Ø F7              BPL  SORTED     ; LENGTH, CAN'T EXCEED $7F
ØE49:E8                 INX             ; SET X BACK TO Ø
ØE4A:86 DA              STX  BUFPTR     ; INITIALIZE BUFFER POINTER
ØE4C:2Ø 38 ØA           JSR  SAVEARYTAB ; WILL OVERWRITE THIS
ØE4F:8D 18 Ø8           STA  NEWARYTAB+1 ; TO AVOID MOVING
ØE52:A5 6B              LDA  ARYTAB     ; MOVED ARRAYS TWICE
ØE54:8D 17 Ø8           STA  NEWARYTAB
ØE57:          ;
ØE57:          ;      THE MOVE ROUTINE:  TWO ARRAYS HAVE SAME NAME. THE
ØE57:          ;      WRONG ONE IS LOWER IN MEMORY.  MOVE RENAMED ARRAY
ØE57:          ;      DOWN TO THE START OF ARRAY STORAGE, VIA THE INPUT
ØE57:          ;      BUFFER AT $2ØØ, IN SEGMENTS NO LONGER THAN 1 PAGE.
ØE57:          ;      MAKE ROOM FOR IT BY MOVING OLD ARRAYS UP UNTIL THEY
ØE57:          ;      OVERWRITE THE SEGMENT OF THE ARRAY MOVED DOWN.  NO
ØE57:          ;      VARIABLE STORAGE SPACE IS USED BY THIS ROUTINE, SO
ØE57:          ;      IF GRAPHICS ARE STORED HIGHER, THEY ARE LEFT ALONE.
ØE57:          ;
ØE57:AD 17 Ø8  MOVEARY  LDA  NEWARYTAB  ; POINTS TO FIRST ARRAY
ØE5A:85 6B              STA  ARYTAB     ; FOLLOWING THE LAST ONE
ØE5C:AD 18 Ø8           LDA  NEWARYTAB+1 ; THAT WAS MOVED DOWN
ØE5F:85 6C              STA  ARYTAB+1
ØE61:BD 1A Ø8           LDA  SECBUF,X   ; FIND NEXT ARRAY PTR
ØE64:85 FA              STA  GENPTR     ; SAVE IT
ØE66:E8                 INX             ; GET THE HIGH BYTE
```

LISTING 1: SUBR.MASTER *(continued)*

```
ØE67:BD 1A Ø8             LDA  SECBUF,X
ØE6A:85 FB                STA  GENPTR+1
ØE6C:E8                   INX              ; POINT TO NEXT ARRAY
ØE6D:86 DA                STX  BUFPTR      ; SAVE IT
ØE6F:AØ Ø2                LDY  #2          ; FETCH OFFSET OF
ØE71:38                   SEC              ; THIS ARRAY TO NEXT
ØE72:B1 FA                LDA  (GENPTR),Y ; THIS POINTS TO
ØE74:AA                   TAX              ; START OF NEXT ARRAY.
ØE75:E9 Ø1                SBC  #1          ; THIS POINTS TO END OF
ØE77:85 77                STA  NUMCHR      ; THIS ONE.
ØE79:C8                   INY              ; FETCH HIGH BYTE
ØE7A:B1 FA                LDA  (GENPTR),Y ; # FULL PAGES TO MOVE
ØE7C:48                   PHA              ; SAVE TO ADJUST ARYTAB
ØE7D:E9 ØØ                SBC  #Ø          ; CARRY FROM -1 ABOVE
ØE7F:85 78                STA  NUMPAGE     ; NOW ADJUST THIS UP
ØE81:E6 78                INC  NUMPAGE     ; BY 1 FOR THE PARTIAL PAGE
ØE83:18                   CLC              ; ADD OFFSET OF ARRAY
ØE84:8A                   TXA              ; TO ARYTAB TO GET FIRST
ØE85:65 6B                ADC  ARYTAB      ; LOC OF NEXT ARRAY
ØE87:8D 17 Ø8             STA  NEWARYTAB ; ONCE THIS ONE
ØE8A:68                   PLA              ; IS MOVED TO THE BOTTOM
ØE8B:65 6C                ADC  ARYTAB+1    ; OF ARRAY STORAGE
ØE8D:8D 18 Ø8             STA  NEWARYTAB+1
ØE9Ø:A4 77                LDY  NUMCHR      ; IS THERE A PARTIAL PAGE?
ØE92:DØ Ø4                BNE  MOVE1       ; YES, MOVE IT
ØE94:C6 78                DEC  NUMPAGE     ; NO, THEN DONE PARTIAL PAGE
ØE96:C6 77                DEC  NUMCHR      ; AND SET THIS TO #$FF
ØE98:A5 6B      MOVE1     LDA  ARYTAB      ; SET UP 1ST BLTU
ØE9A:85 9B                STA  LOWTR       ; MOVE FROM BOTTOM OF
ØE9C:A5 6C                LDA  ARYTAB+1    ; OLD ARRAYS TO OVERWRITE
ØE9E:85 9C                STA  LOWTR+1     ; ARRAY SEGMENT BEING MOVED
ØEAØ:B1 FA      MOVEDOWN LDA  (GENPTR),Y ; TO THE BUFFER
ØEA2:99 ØØ Ø2             STA  BUFR,Y      ; MOVE THIS SEGMENT DOWN
ØEA5:88                   DEY              ; TO PAGE $2ØØ-$2FF, IE
ØEA6:DØ F8                BNE  MOVEDOWN    ; THE INPUT BUFFER
ØEA8:B1 FA                LDA  (GENPTR),Y ; COVERS Y=Ø
ØEAA:8D ØØ Ø2             STA  BUFR        ; NOW MOVE BUFFER UP.
ØEAD:38                   SEC              ; ADDS 1 TO # OF BYTES, AS
ØEAE:A5 FA                LDA  GENPTR      ; REQUIRED FOR BLTU. MOVE
ØEBØ:85 96                STA  HIGHTR      ; FROM LOW TRANSFER ADDRESS
ØEB2:65 77                ADC  NUMCHR      ; (LOWTR) UP, WITH LAST BYTE
ØEB4:85 94                STA  HIGHDS      ; STORED IN HIGH DESTINATION
ØEB6:A5 FB                LDA  GENPTR+1    ; WHICH IS HIGHDS -1
ØEB8:85 97                STA  HIGHTR+1    ; MOVE ONLY A PAGE.  DON'T ADD 1
ØEBA:69 ØØ                ADC  #Ø          ; FOR HIGHTR, OR WILL MOVE
ØEBC:85 95                STA  HIGHDS+1    ; TOO MUCH.
ØEBE:2Ø 9A D3             JSR  BLTUP       ; LATE ENTRY AVOIDS REASON CHECK
ØEC1:A4 77                LDY  NUMCHR      ; NOW MOVE BUFR UP
ØEC3:B9 ØØ Ø2   MOVEUP    LDA  BUFR,Y      ; TO EMPTY AREA
ØEC6:91 9B                STA  (LOWTR),Y  ; ABOVE OLD LOW
ØEC8:88                   DEY              ; TRANSFER ADDRESS
ØEC9:DØ F8                BNE  MOVEUP      ; AGAIN, MISSES Y=Ø
ØECB:AD ØØ Ø2             LDA  BUFR        ; SO DO IT HERE
ØECE:91 9B                STA  (LOWTR),Y  ; SINCE Ø BYTE MOVED
ØEDØ:38                   SEC              ; ACTUALLY MOVED NUMCHR+1 BYTES
ØED1:A5 9B                LDA  LOWTR       ; CARRY SET ADDS 1 MORE
ØED3:65 77                ADC  NUMCHR      ; THIS CALCULATES THE NEW ADDRESS
ØED5:85 9B                STA  LOWTR       ; OF THE BOTTOM OF THE
ØED7:9Ø Ø2                BCC  MOVE2       ; OLD ARRAYS TO MOVE UP.
ØED9:E6 9C                INC  LOWTR+1     ; IF NEEDED.
ØEDB:38         MOVE2     SEC              ; FOR NUMCHR+1
ØEDC:A5 FA                LDA  GENPTR      ; UPDATE PTR TO REMAINDER
ØEDE:65 77                ADC  NUMCHR      ; OF ARRAY TO GO DOWN
ØEEØ:85 FA                STA  GENPTR
ØEE2:9Ø Ø2                BCC  MOVE3
ØEE4:E6 FB                INC  GENPTR+1
ØEE6:AØ FF      MOVE3     LDY  #$FF        ; FULL PAGE MOVES FROM
ØEE8:84 77                STY  NUMCHR      ; HERE. Y SET FOR MOVEDOWN.
ØEEA:C6 78                DEC  NUMPAGE     ; MORE PAGES TO MOVE?
ØEEC:DØ B2                BNE  MOVEDOWN    ; YES, DO.
ØEEE:A6 DA                LDX  BUFPTR      ; NO. MORE VARS TO MOVE?
ØEFØ:E4 DB                CPX  COUNTER
ØEF2:BØ Ø3                BCS  MOVED       ; NO, DONE ARRAY PASS.
ØEF4:4C 57 ØE             JMP  MOVEARY     ; YES, DO NEXT.
ØEF7:2Ø 43 ØA   MOVED     JSR  GETARYTAB ; RECOVER ARYTAB
ØEFA:2Ø 1Ø ØA   LOCAL     JSR  GETNAME     ; VARS ALL PASSED. PROC
ØEFD:AØ Ø4                LDY  #4          ; NAMED VAR HOLDS ADDRESS OF THIS
ØEFF:B1 83      SAVEPROC LDA  (VARPNT),Y ; PROGRAM. SAVE THE
ØFØ1:99 1A Ø8             STA  SECBUF,Y    ; STORED REPRESENTATION, AND
ØFØ4:88                   DEY              ; LOAD IT INTO A NEW PROCNAME VAR
ØFØ5:1Ø F8                BPL  SAVEPROC    ; LATER.  THIS ALLOWS RECURSION.
ØFØ7:C8                   INY              ; BACK TO Ø
ØFØ8:AD 15 Ø8             LDA  OLDSIMPLE ; STORE START ADDRESS OF
ØFØB:91 83                STA  (VARPNT),Y ; (Y=Ø) MAIN'S SIMPLES
ØFØD:C8                   INY              ; (Y=1) IN PROCNAME VAR
ØFØE:AD 16 Ø8             LDA  OLDSIMPLE+1
ØF11:91 83                STA  (VARPNT),Y
ØF13:2Ø Ø5 ØA             JSR  POINTNAME ; MAKE A NEW VARIABLE
ØF16:2Ø 8E ØB             JSR  MAKESIMPLE ; WITH PROC'S NAME.
ØF19:AØ ØØ                LDY  #Ø          ; PUT FURTHER RETURN DATA
ØF1B:AD Ø7 Ø8             LDA  DEFLIST     ; AWAY IN THE NEW
ØF1E:91 83                STA  (VARPNT),Y ; PROC VAR.
ØF2Ø:C8                   INY              ; Y=1
ØF21:AD Ø8 Ø8             LDA  DEFLIST+1
ØF24:91 83                STA  (VARPNT),Y ; START OF DEFLIST SAVED
```

without properly exiting from the called routine. Local variables will now be global, and renamed variables will stay renamed.

Similarly, you might call a subroutine within a FOR loop and have one too many NEXTs inside it. You get back to the FOR without executing the EXIT. In the case of FOR, you can protect yourself easily. Use NEXT with the proper index variable (e.g., NEXT I). When the index is encountered, Applesoft checks it against the name it should find (saved on the stack) in the location it expects to find the index (also on the stack). The index variable has been moved by the creation of locals (at least three) during the CALL, so the test fails, and BASIC crashes the program with a NEXT WITHOUT FOR error — just as it does with crossed FORs and GOSUBs. However, if you don't specify the index variable, and you do cross the routines, you are in trouble. If you encounter this kind of problem, pressing <RESET> followed by FP reinitializes everything from DOS 3.3. Reload the handler and your program, fix the program and try again. From ProDOS you must reboot.

Crossing FOR and DISP — DISP is a subroutine of the handler that can be used in its own right to clear variables out of memory. If you clear a variable that occurred lower in memory than the index of a FOR loop, while the loop is active, and don't specify the index in the NEXT, the computer will create and clear the variables in the DISP list until you stop the program with <CTRL>C or <RESET>. If you specify the index (NEXT I instead of NEXT), the problem will be flagged with an error message. Further, the problem is rare because typical index variables, like I, are usually defined very early in the program, before variables that you would want to clear out later, so their location is not affected by the clear.

ONERR-GOTO, ONERR-GOODBYE — ONERR should be used with extreme caution, or not at all, when calling or exiting a subroutine. Within the CALL and EXIT, Applesoft's internal pointers are readjusted in unconventional ways when making and clearing local variables. If a programming error is detected at this stage (very rare since most errors are detected before new local variables are made), your pointers on return are not valid.

Modified Program Code — In a few places in our program, we have to look at the Applesoft program's variables one at a time. A comma after a variable is usually an adequate separator, but not for a DIM or a DISP, which expect variable lists. In these cases, we temporarily trade the comma for a colon. If an array in the LOCAL list is syntactically mis-specified (e.g., D(−2)), BASIC crashes while DIMming it before we can put the comma back. Thus, the program now has a colon where we used to have a comma. This is useful for pinpointing the error: if there is a new colon, you know that the variable preceding it is the bad one. Further, it's harmless. If you miss it, you get a SYNTAX ERROR next time. Still, it modifies the code, which we didn't intend.

USEFUL HANDLER SUBROUTINES

Dispose

At last, we have come to dispose! Dispose is a Pascal command (Jensen and Wirth Standard, i.e., the original Pascal) that some microcomputer software distributors (like Apple) did not include in their versions of Pascal. If you set DISP=2304 (DISPOSE is parsed by BASIC into DIS POS E), and CALL DISP,*variable list*, all of the variables in the list will be erased from memory.

DISP can also do strange things to DEF FN functions, just as it can do them to FOR loops, as noted previously. If you DEF an FN after declaring a variable which you then dispose of, the FN will be moved down, and its internal pointers will be incorrect. Disposing of array variables never affects FNs, nor does clearing of simple variables that first appeared in the program after the DEF FN statement. If you use FNs in your main program, be cautious with DISP.

The remaining routines are only of interest to assembly language programmers. We assume that you have the premier issue of *Apple Orchard*, with Crossley's documentation of Applesoft pointers and subroutine, and that you have one of Apple's reference manuals.

NEWMOVE

The NEWMOVE routine starting at $96F mimics the Monitor MOVE routine. Its inputs are the same and it leaves Carry Set on exit, as does MOVE. It may repeat sequences if used to move data upward in memory, just as MOVE does. To move data up without worrying about this, use BLTU (Block Transfer Up) at $D393 or BLTUP at $D39A, which doesn't check or change STREND (so use this cautiously).

The Monitor MOVE ($FE2C) is documented on pages 44-46 and 55-56 of the *Apple II Reference Manual*. It moves data starting at the address pointed to by $3C,$3D through $3E,$3F, into the memory range starting at the location held in $42,$43. Our program does the same.

There are four differences between our routine and the Monitor's. If you specify a move starting location that is greater than or equal to the move ending location, our routine assumes you didn't mean it and gives an ILLEGAL QUANTITY error. The Monitor's MOVE moves nothing instead, or one byte, without flagging the error. Second, our program is longer than MOVE. Third, it's much faster for moves of more than a page (255 bytes) of data. For very small moves, MOVE is faster, but these take such a short

LISTING 1: SUBR.MASTER *(continued)*

```
ØF26:C8                  INY               ; Y=2
ØF27:AD Ø9 Ø8            LDA  CALLIST      ; SAVE START OF CALL
ØF2A:91 83               STA  (VARPNT),Y   ; VARIABLE LIST.
ØF2C:C8                  INY               ; Y-3
ØF2D:AD ØA Ø8            LDA  CALLIST+1
ØF30:91 83               STA  (VARPNT),Y
ØF32:C8                  INY               ; Y=4. LAST THIS VARIABLE.
ØF33:A5 75               LDA  CURLIN       ; CALL STATEMENT
ØF35:91 83               STA  (VARPNT),Y   ; LINE NUMBER.
ØF37:2Ø Ø5 ØA            JSR  POINTNAME    ; MAKE A NEW PROCNAME
ØF3A:2Ø 8E ØB            JSR  MAKESIMPLE   ; TO HOLD THE REST.
ØF3D:AØ Ø4               LDY  #4
ØF3F:A5 76               LDA  CURLIN+1     ; HIGH BYTE
ØF41:91 83               STA  (VARPNT),Y
ØF43:88                  DEY               ; Y=3. GO DOWN TO Ø
ØF44:A5 6D               LDA  STREND       ; SNEAKY TRICK.
ØF46:91 83               STA  (VARPNT),Y   ; CHECK STREND WHEN
ØF48:88                  DEY               ; (Y=2) EXIT ATTEMPTED. IF THAT
ØF49:A5 6E               LDA  STREND+1     ; STREND DOESN'T MATCH
ØF4B:91 83               STA  (VARPNT),Y   ; THIS ONE, GLOBAL VARS
ØF4D:88                  DEY               ; (Y=1) WERE TINKERED WITH. SCREAM.
ØF4E:A9 ØØ               LDA  #Ø           ; PUT Ø TO FLAG NO LOCAL LIST.
ØF5Ø:91 83               STA  (VARPNT),Y   ; CHANGE IF IS ONE.
ØF52:2Ø CC Ø9            JSR  POINTDEF     ; POINTS TO END OF LIST
ØF55:2Ø 72 ØA            JSR  DEFTOCUR     ; PUT DEFLINE IN CURLIN
ØF58:2Ø B7 ØØ            JSR  CHRGOT       ; WHAT ENDS THE LIST?
ØF5B:FØ.Ø3               BEQ  LOCALØ       ; MUST BE COLON OR EOL
ØF5D:4C BC E1            JMP  DATAERR      ; OR LIST LONGER THAN CALL LIST.
ØF6Ø:C9 ØØ      LOCALØ   CMP  #Ø           ; TRUE END OF LINE?
ØF62:DØ 16               BNE  LOCAL2       ; NO, WHAT FOLLOWS?
ØF64:AØ Ø2               LDY  #2           ; YES, FIND NEXT LINE
ØF66:B1 B8               LDA  (TXTPTR),Y   ; HIGH BYTE OF PTR
ØF68:DØ Ø3               BNE  LOCAL1       ; IF Ø, END OF PROGRAM
ØF6A:4C 71 ØC            JMP  UNDEF        ; WHICH IS CRAZY.
ØF6D:C8         LOCAL1   INY               ; POINT TO NEW LINE #
ØF6E:B1 B8               LDA  (TXTPTR),Y   ; SAVE IT AS
ØF7Ø:85 75               STA  CURLIN       ; CURRENT LINE
ØF72:C8                  INY
ØF73:B1 B8               LDA  (TXTPTR),Y
ØF75:85 76               STA  CURLIN+1
ØF77:2Ø 98 D9            JSR  ADDON        ; TXTPTR TO LINE'S TEXT
ØF7A:AØ Ø1      LOCAL2   LDY  #1           ; HAVE WE A "LOCAL' STATEMENT?
ØF7C:B1 B8               LDA  (TXTPTR),Y   ; LET'S SEE...
ØF7E:C9 4C               CMP  #EL          ; L
ØF8Ø:DØ 1C               BNE  NOTLOCAL
ØF82:C8                  INY
ØF83:B1 B8               LDA  (TXTPTR),Y
ØF85:C9 4F               CMP  #OH          ; O
ØF87:DØ 15               BNE  NOTLOCAL
ØF89:C8                  INY
ØF8A:B1 B8               LDA  (TXTPTR),Y
ØF8C:C9 43               CMP  #CE          ; C
ØF8E:DØ ØE               BNE  NOTLOCAL
ØF9Ø:C8                  INY
ØF91:B1 B8               LDA  (TXTPTR),Y
ØF93:C9 41               CMP  #EH          ; A
ØF95:DØ Ø7               BNE  NOTLOCAL
ØF97:C8                  INY
ØF98:B1 B8               LDA  (TXTPTR),Y
ØF9A:C9 4C               CMP  #EL          ; L
ØF9C:FØ Ø6               BEQ  LOCAL3       ; L.O.C.A.L. YUP! DO IT.
ØF9E:2Ø 72 ØA   NOTLOCAL JSR  DEFTOCUR     ; NO. RESTORE DEF LINE
ØFA1:4C C4 ØF            JMP  CALLDONE     ; AND LEAVE.
ØFA4:C8         LOCAL3   INY               ; TXTPTR TO CHR AFTER "LOCAL"
ØFA5:2Ø 98 D9            JSR  ADDON        ; ONCE THIS DONE
ØFA8:2Ø BE DE            JSR  CHKCOM       ; MUST BE COMMA THERE.
ØFAB:A5 B8               LDA  TXTPTR       ; SAVE TRUE START OF LOCAL
ØFAD:AØ ØØ               LDY  #Ø           ; VARIABLE LIST.
ØFAF:91 83               STA  (VARPNT),Y   ; IN THE FIRST TWO BYTES
ØFB1:C8                  INY               ; OF THE LATEST PROCNAME VAR.
ØFB2:A5 B9               LDA  TXTPTR+1     ; NEVER Ø
ØFB4:91 83               STA  (VARPNT),Y
ØFB6:2Ø 19 ØA            JSR  DECTXT       ; MOVE BACK TO COMMA
ØFB9:2Ø 8Ø ØB   LOCAL4   JSR  MAKEVAR      ; MAKE A LOCAL
ØFBC:2Ø DE Ø9            JSR  TXTTODEF     ; UPDATE DEF LIST PTR
ØFBF:2Ø B7 ØØ            JSR  CHRGOT       ; END OF LIST?
ØFC2:DØ F5               BNE  LOCAL4       ; NO, MAKE NEXT LOCAL
ØFC4:2Ø Ø5 ØA   CALLDONE JSR  POINTNAME    ; ONE LAST PROCNAME
ØFC7:2Ø 8E ØB            JSR  MAKESIMPLE   ; COMING UP.
ØFCA:AØ Ø4               LDY  #4           ; PUT THE CALL HANDLER
ØFCC:B9 1A Ø8   LASTPROC LDA  SECBUF.Y     ; ADDRESS IN IT.
ØFCF:91 83               STA  (VARPNT),Y
ØFD1:88                  DEY
ØFD2:1Ø F8               BPL  LASTPROC
ØFD4:2Ø CC Ø9            JSR  POINTDEF     ; POINT TO END OF DEF
ØFD7:4C B8 Ø9            JMP  OUT          ; RESTORE $FA-FF & EXIT
ØFDA:           *********************************
ØFDA:           *         CALL THIS ADDRESS     *
ØFDA:           *         TO EXIT FROM PROC     *
ØFDA:           *********************************
ØFDA:2Ø AA Ø9   EXIT     JSR  IN           ; FREE UP $FA-FF
ØFDD:A5 B8               LDA  TXTPTR       ; POINTS AT COMMA
ØFDF:8D ØD Ø8            STA  PROCNAME     ; PRECEDING THE PROC
ØFE2:A5 B9               LDA  TXTPTR+1     ; NAME FOLLOWING "EXIT"
ØFE4:8D ØE Ø8            STA  PROCNAME+1
ØFE7:2Ø BE DE            JSR  CHKCOM       ; BETTER BE A COMMA
```

time that it doesn't matter which routine you use. Finally, our routine sets Y internally. You need not set Y=0 before entering it.

Variable Finding Routines

Applesoft uses two subroutines to find the name and address of variables in memory: PTRGET ($DFE3) and GETARYPTR ($F7D9), which uses part of PTRGET. PTRGET finds the name of the variable (of any type), sets various flags in the process, and creates the variable if it didn't previously exist. It is this routine that DIMs previously undefined arrays. GETARYPTR finds the name of arrays only, sets flags, and looks for an array. It will not create a new array, but it will crash if one doesn't exist.

We needed to separate these functions. The subroutine GETVARNAM ($AE4) stores the name of your variable in LASTVAR ($81,$82) in the same way that PTRGET does, and it sets the same flags. It also flags simple versus array variables in ARYFLAG ($DD, an alias of ERRPOS), and expressions versus simple variables in EXPRFLAG ($DE, alias ERRNUM). Both error locations are used only when BASIC crashes, so you're not hurting anything by using them in the meantime. In the event of an Applesoft error, ARYFLAG and EXPRFLAG are overwritten. If you work with an array expression, ARYFLAG is set (holds $FF) if the first part of the expression is an array name. Otherwise, EXPRFLAG is set (holds $FF). GETVARNAM assumes that TXTPTR ($B8,$B9) points to the first character preceding the variable or expression. TXTPTR is unaffected by the routine.

> The DEF statement's variable list tells you automatically what types of variables the subroutine expects as input.

Subroutine FINDARY assumes that GETVARNAM has just been run. It searches memory for the array whose name is stored in LASTVAR. If the array exists, then, like GETARYPTR, it returns the first location of the array in LOWTR ($9B,$9C). Y is not zero in this case. If the array is not found, then rather than crashing OUT OF DATA (GETARYPTR) or DIMming an array (PTRGET), FINDARY returns with Y=0 and lets you decide what to do from here. TXTPTR is unaffected by the routine as is VARPNT ($83,$84), which PTRGET, but not GETARYPTR, changes.

Subroutine SKIPVAR ($A7D) will bypass an expression or variable in a list, with variables separated by commas or terminated by colons or the end of the line. On entry,

LISTING 1: SUBR.MASTER *(continued)*

```
ØFEA:2Ø E3 DF                JSR  PTRGET       ; WHERE IS IT?
ØFED:AØ ØØ                   LDY  #Ø           ; MUST SAVE THE
ØFEF:B1 83                   LDA  (VARPNT),Y   ; FIRST TWO BYTES
ØFF1:48                      PHA               ; OF THE ADDRESS OF
ØFF2:C8                      INY               ; PROC AS THESE WERE
ØFF3:B1 83                   LDA  (VARPNT),Y   ; OVERWRITTEN
ØFF5:48                      PHA               ; MAKING ROOM FOR OLDSIMPLE.
ØFF6:2Ø Ø5 ØA                JSR  POINTNAME    ; NOW GET RID OF
ØFF9:2Ø ØØ Ø9                JSR  DISPOSE      ; THIS VARIABLE.
ØFFC:2Ø 1Ø ØA                JSR  GETNAME      ; AND GET NEXT WITH THIS NAME
ØFFF:AØ Ø1                   LDY  #1           ; DO WE HAVE A
1ØØ1:B1 83                   LDA  (VARPNT),Y   ; LOCAL VARLIST?
1ØØ3:FØ ØD                   BEQ  EXIT1        ; NOT IF THIS IS Ø
1ØØ5:85 B9                   STA  TXTPTR+1     ; IF YES, POINT TO IT
1ØØ7:88                      DEY               ; POINTS TO TRUE START OF
1ØØ8:B1 83                   LDA  (VARPNT),Y   ; LOCAL LIST, NOT TO
1ØØA:85 B8                   STA  TXTPTR       ; LEADING COMMA. USE CLEAR.
1ØØC:2Ø Ø3 Ø9                JSR  CLEAR        ; AND BYE, BYE LOCALS.
1ØØF:2Ø 1Ø ØA                JSR  GETNAME      ; RECOVER PROCNAM LOC
1Ø12:AØ Ø2        EXIT1      LDY  #2           ; NOW CHECK STREND
1Ø14:B1 83                   LDA  (VARPNT),Y   ; HIGH BYTE
1Ø16:C5 6E                   CMP  STREND+1     ; MUST MATCH
1Ø18:DØ Ø7                   BNE  MEMORYERR
1Ø1A:C8                      INY
1Ø1B:B1 83                   LDA  (VARPNT),Y   ; Y=3
1Ø1D:C5 6D                   CMP  STREND
1Ø1F:FØ Ø5                   BEQ  EXIT3
1Ø21:A2 54        MEMORYERR LDX #84            ; GLOBALS WERE TINKERED WITH.
1Ø23:4C 12 D4                JMP  ERROR        ; "MEMORY ERROR"
1Ø26:C8           EXIT3      INY               ; Y=4
1Ø27:B1 83                   LDA  (VARPNT),Y   ; FETCH CALL LINE #
1Ø29:85 76                   STA  CURLIN+1     ; AND FLAG THIS AS CURRENT.
1Ø2B:2Ø Ø5 ØA                JSR  POINTNAME    ; THIS PROCNAME DONE
1Ø2E:2Ø ØØ Ø9                JSR  DISPOSE      ; GOODBYE
1Ø31:2Ø 1Ø ØA                JSR  GETNAME      ; NEXT PROCNAME, PLEASE.
1Ø34:AØ Ø4                   LDY  #4           ; FETCH CURLIN LOW BYTE
1Ø36:B1 83                   LDA  (VARPNT),Y
1Ø38:85 75                   STA  CURLIN
1Ø3A:88                      DEY               ; Y=3
1Ø3B:B1 83                   LDA  (VARPNT),Y   ; PTR TO
1Ø3D:8D ØA Ø8                STA  CALLIST+1    ; CALL VARLIST
1Ø4Ø:88                      DEY               ; Y=2
1Ø41:B1 83                   LDA  (VARPNT),Y
1Ø43:8D Ø9 Ø8                STA  CALLIST
1Ø46:88                      DEY               ; Y-1
1Ø47:B1 83                   LDA  (VARPNT),Y   ; PTR TO
1Ø49:8D Ø8 Ø8                STA  DEFLIST+1    ; DEF VARLIST
1Ø4C:88                      DEY               ; Y=Ø
1Ø4D:B1 83                   LDA  (VARPNT),Y
1Ø4F:8D Ø7 Ø8                STA  DEFLIST
1Ø52:2Ø Ø5 ØA                JSR  POINTNAME    ; ALL INFO FROM THIS
1Ø55:2Ø ØØ Ø9                JSR  DISPOSE      ; VAR USED. CLEAR IT.
1Ø58:2Ø 1Ø ØA                JSR  GETNAME      ; FIND LAST ONE.
1Ø5B:AØ Ø1                   LDY  #1           ; AND RESTORE
1Ø5D:B1 83                   LDA  (VARPNT),Y   ; TOP TWO BYTES.
1Ø5F:8D 16 Ø8                STA  OLDSIMPLE+1
1Ø62:68                      PLA               ; ADDRESS RECOVERY
1Ø63:91 83                   STA  (VARPNT),Y
1Ø65:88                      DEY               ; Y=Ø
1Ø66:B1 83                   LDA  (VARPNT),Y   ; THEN START PASSING
1Ø68:8D 15 Ø8                STA  OLDSIMPLE    ; THE DATA BACK.
1Ø6B:68                      PLA
1Ø6C:91 83                   STA  (VARPNT),Y
1Ø6E:84 DA                   STY  BUFPTR       ; INITIALIZE
1Ø7Ø:84 DB                   STY  COUNTER      ; THE
1Ø72:2Ø FØ Ø9                JSR  STARTLIST    ; POINTERS
1Ø75:2Ø CC Ø9     ARYBACK    JSR  POINTDEF     ; START BY PASSING
1Ø78:2Ø B7 ØØ                JSR  CHRGOT       ; ARRAYS, WHILE ARRAYS TO PASS.
1Ø7B:FØ 33                   BEQ  NAMEBACK     ; LIST THEM, THEN RENAME.
1Ø7D:2Ø E4 ØA                JSR  GETVARNAM    ; GET NAME & TYPE
1Ø8Ø:A5 DD                   LDA  ARYFLAG      ; GOT AN ARRAY?
1Ø82:FØ 27                   BEQ  ABACK1       ; IF NOT, SKIP AND DO NEXT
1Ø84:2Ø 5Ø ØB                JSR  FINDARY      ; GET ARRAY'S ADDRESS
1Ø87:A6 DA                   LDX  BUFPTR       ; SAVE NEW NAME, OLD ADDRESS.
1Ø89:A5 9B                   LDA  LOWTR        ; OLD ADDRESS
1Ø8B:9D 1C Ø8                STA  SECBUF+2,X   ; THIS ORDER FOR
1Ø8E:A5 9C                   LDA  LOWTR+1      ; THE RENAME ROUTINE
1Ø9Ø:9D 1D Ø8                STA  SECBUF+3,X
1Ø93:2Ø D5 Ø9                JSR  POINTCALL    ; GET THE NAME
1Ø96:2Ø E4 ØA                JSR  GETVARNAM    ; OF CALL LIST ARRAY
1Ø99:A5 81                   LDA  LASTVAR      ; NOW SAVE NEW NAME
1Ø9B:A6 DA                   LDX  BUFPTR
1Ø9D:9D 1A Ø8                STA  SECBUF,X
1ØAØ:A5 82                   LDA  LASTVAR+1
1ØA2:9D 1B Ø8                STA  SECBUF+1,X
1ØA5:8A                      TXA
1ØA6:18                      CLC
1ØA7:69 Ø4                   ADC  #4
1ØA9:85 DA                   STA  BUFPTR       ; UPDATE LIST POINTER
1ØAB:2Ø D2 ØA     ABACK1     JSR  ADVANCEPTRS  ; UPDATE VAR PTRS
1ØAE:9Ø C5                   BCC  ARYBACK      ; ALWAYS TAKEN
1ØBØ:2Ø 22 ØC     NAMEBACK  JSR RENAME
1ØB3:A9 ØØ                   LDA  #Ø
1ØB5:85 DA                   STA  BUFPTR       ; INITIALIZE AGAIN
1ØB7:2Ø FØ Ø9                JSR  STARTLIST    ; RESET CALL & DEF PTRS
```

TXTPTR points to the first character preceding the expression or variable. On exit, TXTPTR points to the first character (e.g., a comma) following it. This is the routine to use to skip variables or expressions after using GETVARNAM to find out what they are.

Local Variables

Subroutine MAKEVAR ($B80) creates all local variables. On entry, TXTPTR points to the character preceding the simple or array variable. If this is an expression that does not start with an array, you get a SYNTAX ERROR. If it is an array-started expression, whatever routine you call next will probably give you a SYNTAX ERROR, but don't bet on it. On exit, TXTPTR points to the character following the variable name. Simple variables are always created starting at the bottom of variable storage (VARTAB, $69,$6A). Arrays are created at the top of array storage (above what used to be storage end, $6D,$6E) if an array with that name does not exist. This is faster than creating one at the bottom of array storage (ARYTAB, $6B,$6C), which is only done if an old array has the same name as the new one.

Page Zero Save Routines

Many of these and other short utilities for moving page zero values are found at the start of the program ($9AA to $AAC). Specific locations from $801 up are used to store page zero variables while you use the page zero locations for some purpose of your own. The SECBUF region of page 8 (from $81A to $8FF) is used as temporary storage by various routines in this program but not by any you would be likely to use outside of this program. Thus, $81A to $8FF are free for temporary storage by your other assembly language subroutines. The subroutine handler does not rely on any values from this address range when the routine is entered, either at PROC (for CALLs) or at EXIT, so use these freely.

REFERENCES

Kaner, H.C. and J.R. Vokey. "Modifying Apple's Floating Point BASIC: An & Interpreter Without the &." *Compute!*, May 1982, pp. 146-152.

Mossberg, S. "LAMP — Part II." *Nibble*, Vol. 3/No. 3, 1982, pp. 33-39.

Mottola, R.M. "Amper-Interpreter." *Nibble*, Vol. 1/ No. 6, 1980, pp. 27-44.

Smith, M. "Using Named GOSUB and GOTO Statements in Applesoft BASIC." *Compute!*, May 1981, p. 64.

Worth, D., and P. Lechner. *Beneath Apple DOS*. Quality Software, 1981.

Yourdon, E. *Techniques of Program Structure and Design*. Prentice-Hall, 1975.

LISTING 1: SUBR.MASTER *(continued)*

```
10BA:20 CC 09  923 SIMPLEBACK JSR POINTDEF  ; WHILE SIMPLES TO PASS
10BD:20 B7 00  924            JSR  CHRGOT    ; PASS THEM
10C0:F0 63     925            BEQ  LEAVE     ; THEN DONE
10C2:20 E4 0A  926            JSR  GETVARNAM ; WHAT HAVE WE?
10C5:A5 DD     927            LDA  ARYFLAG   ; IF 0, IS SIMPLE
10C7:D0 57     928            BNE  SBACK3    ; ELSE SKIP VARS
10C9:20 D5 09  929            JSR  POINTCALL ; FIND OUT WHAT CALL
10CC:20 E4 0A  930            JSR  GETVARNAM ; VAR IS BEFORE PASS BACK
10CF:A5 DE     931            LDA  EXPRFLAG  ; DON'T WANT PASS TO EXPR
10D1:D0 33     932            BNE  SBACK2    ; IF EXPRESSION, PASS NOTHING
10D3:A5 DD     933            LDA  ARYFLAG   ; NOT 0 MEANS ARRAY OR ARRAY EXPR
10D5:F0 0F     934            BEQ  SBACK1    ; 0 IS SIMPLE, DO IT.
10D7:20 B1 00  935            JSR  CHRGET    ; ARRAY FLAG SET
10DA:20 E3 DF  936            JSR  PTRGET    ; MOVES TO END OF ARRAY
10DD:20 B7 00  937            JSR  CHRGOT    ; ARRAY IF FOLLOWED BY COMMA OR EOL
10E0:F0 04     938            BEQ  SBACK1    ; IN WHICH CASE PASS BACK
10E2:C9 2C     939            CMP  #COMMA    ; TO IT. ELSE ARRAY EXPR
10E4:D0 20     940            BNE  SBACK2    ; DON'T PASS IT
10E6:20 64 0A  941 SBACK1     JSR  SIMPTOVAR ; WILL PASS TO OLD SIMPLE
10E9:20 D5 09  942            JSR  POINTCALL ; CALL VAR OR TO CALL ARRAY
10EC:20 B1 00  943            JSR  CHRGET    ; PAST COMMA
10EF:20 E3 DF  944            JSR  PTRGET    ; FIND IT
10F2:A5 83     945            LDA  VARPNT    ; NOW PREPARE FOR LET
10F4:85 85     946            STA  FORPNT
10F6:A5 84     947            LDA  VARPNT+1
10F8:85 86     948            STA  FORPNT+1
10FA:20 2D 0A  949            JSR  GETVARTAB ; ALLOW FIND OF DEF SIMPLES
10FD:20 CC 09  950            JSR  POINTDEF  ; SET CALL VAR = DEF VAR
1100:20 B1 00  951            JSR  CHRGET    ; SKIP COMMA
1103:20 52 DA  952            JSR  LETCNT    ; DO THE LET.
1106:20 CC 09  953 SBACK2     JSR  POINTDEF  ; CLEAR OUT DEF SIMPLE.
1109:20 AC 0A  954            JSR  PUTCOLON  ; ONLY CLEAR THE ONE VAR.
110C:20 00 09  955            JSR  DISPOSE
110F:20 CA 0A  956            JSR  REPCOLON  ; RETURN THE CHR AFTER THE VAR
1112:38        957            SEC            ; AND UPDATE
1113:AD 15 08  958            LDA  OLDSIMPLE ; PTR TO THE
1116:E9 07     959            SBC  #7        ; OLD SIMPLE VARS
1118:8D 15 08  960            STA  OLDSIMPLE ; REFERRED TO
111B:B0 03     961            BCS  SBACK3    ; IN THE CALL LIST
111D:CE 16 08  962            DEC  OLDSIMPLE+1
1120:20 D2 0A  963 SBACK3     JSR  ADVANCEPTRS ; MOVE TO NEXT VAR
1123:90 95     964            BCC  SIMPLEBACK ; AND DO IT.
1125:20 D5 09  965 LEAVE      JSR  POINTCALL ; POINTS TO END OF CALL
1128:4C B8 09  966            JMP  OUT       ; RESTORE FA-FF & BACK TO BASIC
```

END OF LISTING 1

```
           KEY  PERFECT                  1DBD82C5   0C70 - 0CBF        22AC
             RUN ON                      6A9ADDF7   0CC0 - 0D0F        286A
           SUBR.MASTER                   5AFE6098   0D10 - 0D5F        2B34
=====================================    BB647EAD   0D60 - 0DAF        2410
CODE-5.0   ADDR# - ADDR#      CODE-4.0   A673FEB7   0DB0 - 0DFF        28B6
--------   -------------      --------   AD353C4E   0E00 - 0E4F        2873
448FE39B    0900 - 094F         2866     624771CA   0E50 - 0E9F        2B1A
B643E5E9    0950 - 099F         261A     790312FB   0EA0 - 0EEF        2AB7
F24E6F74    09A0 - 09EF         2942     2326E17C   0EF0 - 0F3F        2609
A3A23CE3    09F0 - 0A3F         2909     8F729F86   0F40 - 0F8F        286C
55D08203    0A40 - 0A8F         24DA     28EC56AB   0F90 - 0FDF        25B1
3211176C    0A90 - 0ADF         2707     31302CF4   0FE0 - 102F        25F6
F8DFD3E3    0AE0 - 0B2F         28B5     5170D012   1030 - 107F        2680
F69D94B0    0B30 - 0B7F         2A05     132F20D9   1080 - 10CF        260D
967B0F51    0B80 - 0BCF         264A     1453E41A   10D0 - 111F        27E2
E8C00353    0BD0 - 0C1F         2802     61ABAF59   1120 - 112A        0635
47C956F4    0C20 - 0C6F         2B0C     5BFD2671 = PROGRAM TOTAL =    082B
```

LISTING 2: SUBR.MAST.DEMO1

```
10  REM **********************
20  REM *  SUBR.MAST.DEMO1   *
30  REM * COPYRIGHT (C) 1985 *
40  REM * BY MICROSPARC, INC *
50  REM * CONCORD, MA  01742 *
60  REM **********************
70  IF  PEEK (104) <  > 17 THEN  POKE 103,44:
      POKE 104,17: POKE 4395,0: PRINT  CHR$ (
     4)"RUN SUBR.MAST.DEMO1"
80  IF  PEEK (2304) <  > 32 THEN  PRINT  CHR$
     (4)"BLOAD SUBR.MASTER"
90 SUB1 = 3141:RET = SUB1:EXIT = 4058
100  HOME : VTAB 12: HTAB 3: PRINT "DEMONSTRA
     TION OF SUBROUTINE MASTER": HTAB 6: PRINT
     "BY CEM KANER AND JOHN VOKEY": PRINT " C
     OPYRIGHT (C) 1985 BY MICROSPARC, INC.": CALL
     RET,"TO CONTINUE": HOME
110  HOME : INVERSE : PRINT "DEMONSTRATION OF
      PARAMETER PASSING": NORMAL
```

LISTING 2: SUBR.MAST.DEMO1 *(continued)*

```
12Ø  PRINT : PRINT "THE VALUES OF THE VARIABL
     ES IN THE": PRINT "CALL STATEMENT ARE PA
     SSED TO THE": PRINT "CORRESPONDING VARIA
     BLES IN THE": PRINT "DEF STATEMENT:": LIST
     35Ø: LIST 39Ø
13Ø  PRINT : PRINT "THE VALUES OF THE VARIABL
     ES IN THE": PRINT "DEF STATEMENT ARE PAS
     SED BACK TO THE": PRINT "CORRESPONDING V
     ARIABLES IN THE": PRINT "CALL STATEMENT.
     "
14Ø  CALL RET,"FOR LISTING": HOME : INVERSE :
      PRINT "LISTING OF PARAMETER PASSING DEM
     O:": NORMAL : PRINT : LIST 33Ø,43Ø
15Ø  CALL RET,"TO RUN PROGRAM": HOME : INVERSE
     : PRINT "PARAMETER PASSING DEMO": NORMAL
     : GOSUB 33Ø: CALL RET,"FOR NEXT DEMO"
16Ø  HOME : INVERSE : PRINT "DEMONSTRATION OF
      LOCAL VARIABLES": NORMAL
17Ø  PRINT : PRINT "EACH VARIABLE IN THE DEF
     STATEMENT": PRINT "IS A LOCAL VARIABLE,
     DISTINCT FROM": PRINT "VARIABLES OF THE
     SAME NAME IN THE MAIN": PRINT "PROGRAM."
18Ø  PRINT : PRINT "THE LOCAL STATEMENT CREAT
     ES ADDITIONAL": PRINT "LOCAL VARIABLES T
     HAT ARE DISTINCT": PRINT "FROM MAIN PROG
     RAM VARIABLES."
19Ø  LIST 51Ø
2ØØ  CALL RET,"TO LIST PROGRAM": HOME : INVERSE
     : PRINT "LISTING OF DEMO2": NORMAL : LIST
     44Ø,54Ø
21Ø  CALL RET,"TO RUN PROGRAM": HOME : INVERSE
     : PRINT "LOCAL VARIABLE DEMO": NORMAL : GOSUB
     44Ø
22Ø  CALL RET,"FOR NEXT DEMO"
23Ø  HOME : INVERSE : PRINT "EXPRESSION PASSI
     NG DEMO": NORMAL : PRINT : PRINT "EXPRES
     SIONS MAY BE USED IN THE": PRINT "CALL S
     TATEMENT:": LIST 57Ø
24Ø  PRINT "VARIABLES INCLUDED IN EXPRESSIONS
     ": PRINT "ARE NOT AFFECTED, EVEN IF THE"
     : PRINT "SUBROUTINE CHANGES THE VALUE OF
      THE": PRINT "CORRESPONDING VARIABLE IN
     THE": PRINT "DEF STATEMENT."
25Ø  CALL RET,"TO LIST PROGRAM": HOME : INVERSE
     : PRINT "LISTING OF DEMO3": NORMAL : LIST
     55Ø,65Ø
26Ø  CALL RET,"TO RUN PROGRAM": HOME : INVERSE
     : PRINT "EXPRESSION PASSING DEMO": NORMAL
     : GOSUB 55Ø
27Ø  CALL RET,"FOR NEXT DEMO"
28Ø  HOME : INVERSE : PRINT "DEMONSTRATION OF
      PASSING STRINGS": NORMAL : PRINT : PRINT
     "STRING VARIABLES AND STRING LITERALS": PRINT
     "ARE HANDLED IN THE SAME WAY AS": PRINT
     "NUMERICS.": LIST 68Ø: LIST 72Ø
29Ø  CALL RET,"FOR LISTING": HOME : INVERSE :
      PRINT "LISTING OF DEMO4": NORMAL : LIST
     66Ø,76Ø
3ØØ  CALL RET,"TO RUN PROGRAM": HOME : INVERSE
     : PRINT "STRING PASSING DEMO": NORMAL : GOSUB
     66Ø
31Ø  CALL RET,"TO QUIT": HOME
32Ø  END
33Ø  REM  PARAMETER PASSING DEMO
34Ø A = 5: PRINT : PRINT "A="A" BEFORE."
35Ø  CALL SUB1,A
36Ø  PRINT : PRINT "A="A" AFTER."
37Ø  RETURN : REM *** RETURN FROM THIS DEMO
38Ø  REM *** BEGINNING OF SUB1
39Ø  DEF SUB1,N
4ØØ  PRINT : PRINT "N="N" (VALUE RECEIVED FRO
     M A)"
41Ø N = N * 1Ø: PRINT : PRINT "VALUE OF N CHA
     NGED TO "N"."
42Ø  CALL EXIT.SUB1
43Ø  REM  *** END OF SUB1
44Ø  REM  LOCAL VARIABLE DEMO
45Ø A = 14:B = 34: PRINT : PRINT "BEFORE:": PRINT
     "A="A" AND B="B"  (GLOBAL VARIABLES)
46Ø  CALL SUB2,A
47Ø  PRINT : PRINT "AFTER:": PRINT "A="A" AND
      B="B"  (GLOBAL VARIABLES)
48Ø  RETURN : REM *** RETURN FROM THIS DEMO
49Ø  REM *** BEGINNING OF SUB2
5ØØ  DEF SUB2,B
51Ø LOCAL,A
52Ø A = 4.52: PRINT : PRINT "DURING:": PRINT
     "A="A" AND B="B"  (LOCAL VARIABLES)
53Ø  CALL EXIT,SUB2
54Ø  REM *** END OF SUB2
55Ø  REM  EXPRESSION DEMO
56Ø A = 2: PRINT : PRINT "BEFORE:": PRINT "A=
     "A
57Ø  CALL SUB3,A + 5
58Ø  PRINT : PRINT "AFTER:": PRINT "A="A
59Ø  RETURN : REM  RETURN FROM THIS DEMO
6ØØ  REM *** BEGINNING OF SUB3
61Ø  DEF SUB3,N
62Ø  PRINT : PRINT "N="N"  (RECEIVED FROM MAI
     N)
63Ø N = N * 1Ø: PRINT "N="N"  (CHANGED IN SUB
     3)
64Ø  CALL EXIT,SUB3
65Ø  REM *** END OF SUB3
66Ø  REM  STRING DEMO
67Ø A$ = "ABC": PRINT : PRINT "BEFORE:": PRINT
     "A$=" CHR$ (34)A$ CHR$ (34)
68Ø  CALL SUB4,A$
69Ø  PRINT : PRINT "AFTER:": PRINT "A$=" CHR$
     (34)A$ CHR$ (34)
7ØØ  RETURN
71Ø  REM *** BEGINNING OF SUB4
72Ø  DEF SUB4,X$
73Ø  PRINT : PRINT "X$=" CHR$ (34)X$ CHR$ (34
     )"  (RECEIVED BY SUB4) "
74Ø X$ = "-*-" + X$ + "-*-"
75Ø  CALL EXIT,SUB4
76Ø  REM *** END OF SUB4
77Ø  DEF RET,MS$
78Ø LOCAL,Z$
79Ø  VTAB 23: HTAB 1: PRINT "PRESS <RETURN> "
     MS$;: GET Z$: CALL EXIT,RET
```

END OF LISTING 2

```
                  KEY  PERFECT
                     RUN ON
                 SUBR.MAST.DEMO1
  ========================================
     CODE-5.Ø    LINE# - LINE#   CODE-4.Ø
     --------    -------------   --------
     1AF77823      1Ø -   1ØØ       C8D1
     5C9EØCBE     11Ø -   2ØØ     Ø175ØE
     917D3558     21Ø -   3ØØ     Ø14BF5
     7155AC8C     31Ø -   4ØØ       5A15
     BØ8A5Ø34     41Ø -   5ØØ       76CB
     EB1918CF     51Ø -   6ØØ       6458
     F2F1ADFB     61Ø -   7ØØ       64CA
     FDAØF4CB     71Ø -   79Ø       57A7
     4769Ø8AØ = PROGRAM TOTAL =     ØBA9
```

LISTING 3: SUBR.MAST.DEMO2

```
1Ø   REM **********************
2Ø   REM *   SUBR.MAST.DEMO2    *
3Ø   REM * COPYRIGHT (C) 1985 *
4Ø   REM * BY MICROSPARC, INC *
5Ø   REM * CONCORD, MA  Ø1742 *
6Ø   REM **********************
7Ø   IF  PEEK (1Ø4) <  > 17 THEN  POKE 1Ø3,44:
      POKE 1Ø4,17: POKE 4395,Ø: PRINT  CHR$ (
     4)"RUN SUBR.MAST.DEMO2"
8Ø   IF  PEEK (23Ø4) <  > 32 THEN  PRINT  CHR$
     (4)"BLOAD SUBR.MASTER"
9Ø FACT = 3141:EXIT = 4Ø58: HOME : VTAB 12: PRINT
    "FACTORIAL CALCULATIONS USING RECURSION"
    : PRINT : PRINT "* COPYRIGHT (C) 1985 BY
     MICROSPARC, INC*": VTAB 21: PRINT "PRES
    S <RETURN> TO START";: GET Z$: PRINT : HOME

1ØØ  INPUT "INPUT INTEGER (Ø TO 33): ";A
11Ø RS = 1: REM INITIALIZE RESULT TO 1
12Ø  CALL FACT,A
13Ø  PRINT RS
14Ø  GOTO 1ØØ
15Ø  REM *** BEGINNING OF FACT ROUTINE
16Ø  DEF FACT,N
17Ø  IF N > 1 THEN RS = RS * N: CALL FACT,N -
     1
18Ø  CALL EXIT,FACT
19Ø  REM *** END OF FACT ROUTINE
```

END OF LISTING 3