The Sourceror's Apprentice

The Assembly Language Journal of Merlin Programmers Vol. 1 No. 5 May, 1989

News, Views, and Much Ado About 8-bit Text Editing

Packing up the Ariel igloo

I'll sure be glad when Ross's Great Cross Country Moving Adventure gets finished. What a pain in the circuitry. Let's see, I'd best remind you that we have a new address:

Ariel Publishing
P.O. Box 398
Pateros, WA 98846

(509) 624-3161

I'll be unavailable from May 28th - June 10th, too. I apologize in advance to those who find that an inconvenience.

Furthermore, I promise that we'll get caught up on back ordered stuff (mostly back issues) ASAP. I'll have an authentic, flesh and blood, full time secretary beginning June 10th, so we should really start to resemble a professional operation soon (knock on wood). Don't fret too much about us losing your orders or correspondence. They're all tucked away in my overstuffed briefcase. Isn't that reassuring?

If it sounds like we're busy here at the Ariel igloo, that's 'cuz we are. Things are really going pretty well. Though our progress is modest by most standards, our overhead is low, too, so things are moving steadily forward.

I've received a jillion suggestions about topics to cover in ***The Apprentice,*** all of them good. We've got article fodder for the next few years, I think. Feel free to contribute a suggestion or two - I read 'em all. This month's coverage of text editing routines is brought to you courtesy of intense popular demand (and Prof. Robert Moore, who had no idea how timely his submission was!) You saved my skin again, Bob, and provided a truly outstanding set of routines for the readership. I've never seen anything this comprehensive published anywhere. I am proud to bring it to y'all in its entirety this month.

A GS Update

This is really old news, but... I've been known to wax preachy regarding the *Apple IIGS Toolbox References.* They're pretty close to indispensable for GS work. At present there are two volumes, but Apple recently released the *Toolbox Reference Update.* The *Update,* too, is finding its way onto my "can't do without" list.

First, it corrects outright errors in the *References.* Although there really aren't that many, some of the existing errors can drive you nuts. The QuickDraw chapter, for example, says that calls such as `_LineTo` and `_MoveTo` take global coordinates. It probably didn't take many of you GS types too long to figure out that they don't.

The *Update* also lists several new calls added to the toolboxes since the manuals went to press. We've already examined one of these within our pages, `_AlertWindow`. Another useful new routine is called `_RealFreeMem`, and it's worth a quick "once over" here.

As you've probably already discovered, the Memory Manager function `_FreeMem` only returns the amount of memory *not currently in use.* This is sensible, of course, except when we need to know how much *more* memory would be available if purgable blocks were evicted from the joint.

Thence cometh `_RealFreeMem`. This new call will dutifully report the amount of memory available *after* purgeable blocks are removed. As the *Update* suggests, it gives a much more accurate picture of the state of the silicon. Note that it does *not* actually execute a purge, it just reports what would things would be like if one happened.

The following snippet shows how to use the call:

```
*  _RealFreeMem call

PushLong #0         ;result space
_RealFreeMem
PullLong FreeBytes

* to convert to kilobytes

lda FreeBytes+1
lsr
lsr
sta FreeKilobytes
```

The conversion to kilobytes code looks odd at first blush, but stop and consider that converting from bytes to kilobytes entails a division by 1024. If you're thinking in terms of binary shifts to the right, each of which is a division by two, dividing by 1024 means ten shifts to the right (LSRs). The lowest byte, then, is lost completely. It would be shifted into nothingness.

By leaving the lowest byte out of the process altogether and starting to work on `FreeBytes+1`, we save a few bytes, a few instructions, and a few cycles. This is never a bad idea when possible, even on the memory rich GS.

Note, too, that the high byte of the four byte variable `FreeBytes` is ignored, this because it must always be equal to zero on the GS (at least when we're talking about the range of memory locations).

By the way, if you want to add a macro for this call to your MEM.MACS library on the Merlin disk, make it look like this:

```
~RealFreeMem MAC
        PHS  2
_RealFreeMem MAC
        Tool $2F02
        <<<
```

(This macro is already in later version of Merlin 8/16 and in the new Merlin 16+. I've been asked to remember those who don't have the "latest and greatest" versions of Merlin. The above macro is in their honor.)

Back at the ranch, I've only scratched the surface. The entire *Update* is packed with goodies that make 16 bit life easier. It is available for $30 from APDA (800/282-2732). Yes, $30 is a bit much for looseleaf material. But that is a debate for another day (a day that is coming all too quickly, it appears).

Another product I recommend is RavenWare's *DesignMaster*. Author Chris Haun has put together a neat code generating utility which lets you literally draw your windows, dialogs, menus, etc. Priced at $30, the package is a genuine d-e-a-l. You draw it, and *DesignMaster* produces the code and definition data in either APW or Merlin format (for assembly language junkies), or C or Forth for you high level types. (RavenWare, 23930 Ocean Avenue, #201, Torrance, CA 90505).

AppleFest attendees were also wowed by another code generating product due out in September. GENESYS supposedly does everything except press keys for you. It had better, with a price tag of $125. Seriously, though, my 'Fest spies say it looks *very* impressive.

The GS marketplace is warming. That alone is neat, but Apple's literal "pre-announcement" of System Disk 5.0 at AppleFest bodes well for the II, too. The Apple II is *never* going to get the support *I* think it merits, but I'll devour any bones I'm thrown (and continue yapping for more).

Enough news and views. On with Professor Moore's show... I think you'll like it. And there are no commercial interruptions!

&Input, &Print, and &Get
or
More Bang for Your Text Bytes

by Robert C. Moore
1204 Marton Street
Laurel, MD 20707

Editor: These routines put advanced and powerful text editing routines right at your fingertips. It's the best and most comprehensive program of its kind that I've ever seen.

Bob chose to connect his program to Applesoft, but it is possible to take the ampersand and variable passing routines out if you want to operate in a "pure" assembly environment. It would be a tad trickier, though, if you wanted to switch out the Applesoft ROMs altogether.

I hope you enjoy Bob's code as much as I have.

This article documents an Applesoft extension program which I have called INPUT.PRINT.GET. The program adds three ampersand commands to Applesoft:

&INPUT x$,
&PRINT x$, and
&GET x$.

The commands behave much as the similar commands in AppleWorks' SU2.OBJ do.

The source code is in a format that is compatible with most 6502 assemblers, including Merlin; it needs very few modifications to be used with most other popular assemblers. The source code is very heavily commented. This is to facilitate customization by readers of ***The Sourceror's Apprentice*** who choose to modify the program for their own special uses.

The comments in the source code carefully document the program's use. They also should help you to understand how various portions of the program work. Specifically, the source code illustrates how to install machine language routines above HIMEM in both DOS3.3 and ProDOS 8, how to chain into the ampersand hook, how to read the value of an Applesoft real variable from machine language, how to set the value of an Applesoft string or real variable from machine language, and how to use software "switches" and "signatures" to obtain multiple functions using a single module of code.

The three ampersand commands are installed simply by BRUNning INPUT.PRINT.GET prior to assigning any string variables. (Under ProDOS 8 and BZSIC.SYSTEM you may use the smart run [dash] command.) Once installed, the object code uses only 1024 bytes of memory. During installation, locations $2096 - $24FF are used temporarily. The source code explains how this temporary workspace may be relocated, if the location I have chosen conflicts with any of your previously installed programs.

Zero-page locations $3C through $47 are used temporarily by INPUT.PRINT.GET. Their original contents are destroyed. (This should not be a problem, because these are scratchpad locations for ProDOS 8 and the system monitor.) All other zero-page locations that are normally available to assembly language programs remain accessible.

I have attempted to make this program easy to use and as compatible as possible with other enhancements to Applesoft. The program has been tested on an Apple //c, a "regular" IIe, an enhanced IIe, and a IIGS. It assumes you have Applesoft in ROM, and that you are using text page 1 in either 40- or 80-column mode.

&INPUT x$

&INPUT x$ prints the current (default) value of the specified string variable x$ to the current text screen window (40- or 80-column display) and then permits you to edit the string from the keyboard.

The powerful string editing features of the "&INPUT x$" command are particularly useful:

ARROW KEYS move the blinking underscore "insert" cursor. If the edit string occupies more than one line in the text window then the up- and down-arrow keys will work. This gives you full-screen editing of the string.

DELETE deletes the character to the left of the cursor and closes up the resulting gap in the edit string.

CTRL-D deletes the character under the cursor and closes up the resulting gap in the edit string.

CTRL-X ("cross out") erases the entire edit string.

CTRL-Y erases from the cursor to the end of the edit string.

CTRL-B moves the cursor to the beginning of the edit string.

CTRL-N moves the cursor to the end of the edit string.

CTRL-C toggles the case of the character under the cursor, if it is a letter (alphabetic character), then advances the cursor to the right. Upper case letters are converted to lower case; lower case letters are converted to upper case.

RETURN accepts the current edit string, strips off any trailing spaces, and assigns the resulting string as the new value for the specified string variable, x$.

ESCape aborts the &INPUT x$. The value of the specified string variable, x$, remains at the default. The Applesoft real variable ES is set to 1. (If ESCape is not used to abort an &INPUT x$, the value of variable ES will be set to 0.) The abort may be detected following &INPUT x$ by using ON ES GOTO.

OPEN-APPLE (when used to modify another key) aborts &INPUT x$ and sets the Applesoft variable OA to 128 plus the ASCII value for the key that was pressed (i.e., high-ASCII). (If OPEN-APPLE-key is not used to abort &INPUT x$, the value of variable OA will be set to zero.) For example, OPEN-APPLE-A will abort &INPUT x$ (the value of x$ will remain at the default) and set the value of variable OA to 193. Use of the OPEN-APPLE key to abort &INPUT x$ may be detected by using IF OA GOTO.

SOLID-APPLE (when used to modify another key) aborts &INPUT x$ and sets the Applesoft variable SA to 128 plus the ASCII value for the key that was pressed (i.e., high-ASCII). (If SOLID-APPLE-key is not used to abort &INPUT x$, the value of variable SA will be set to zero.) If both the OPEN-APPLE and the SOLID-APPLE keys are used to modify another key, then both OA and SA will be assigned the high-ASCII value of the key that was pressed.

Another Applesoft variable, FL, may be used to set the maximum field length; that is, the value of FL will determine the maximum length for the edit string. For example, if you are using &INPUT x$ to input a filename under ProDOS, you would want to set FL = 15 because that is the maximum length of a ProDOS filename. If, during editing, you attempt to increase the length of the edit string beyond the value of FL, you will be bleeped. If you execute &INPUT x$ with a default value for x$ that is greater in length than the value of FL, you will generate an Applesoft STRING TOO LONG error. You will get the same error (STRING TOO LONG) if your default string is so long that the top line scrolls off the top of the text screen window as the string is printed. If FL = 0, the maximum field length will be 255 characters.

&GET x$

&GET x$ works as the &INPUT x$ command does, except that the string is limited to exactly one character, no default string is displayed on screen, and ESCape may not be used to abort. The Applesoft variables OA and SA work as with &INPUT x$. Following &GET x$, the high-ASCII value of the key that was pressed may be retrieved from address $3C = 60 using PEEK(60). The new value of x$ will be the single character that was typed at the keyboard.

&GET x$ may be used to get any encoded keypress except CTRL-RESET or OPEN-APPLE-CTRL-RESET. To determine if

ESCape was pressed during &GET x$, use ON (PEEK(60) = 155) GOTO.

As with &INPUT x$, use of the OPEN-APPLE or SOLID-APPLE keys may be detected using IF OA GOTO and/or IF SA GOTO.

While &INPUT x$ and &GET x$ are waiting for keystrokes, they advance a 16-bit unsigned integer in locations $4E,$4F (78, 79) to a new "random" value. (This value may be used to "seed" a pseudorandom number generator.) The "random" value may be obtained using PEEK(78) + 256 * PEEK(79).

&PRINT x$

&PRINT x$ prints the current value of the specified string variable, x$, to the text window with word-wrapping. Lines are broken at spaces, if possible. &PRINT x$ leaves the text screen cursor immediately to the right of the last character that was printed.

I believe this program will be of great interest to readers of The Sourceror's Apprentice, most of whom are intermediate-level Apple II programmers who delight in finding new ways by which the power of Apple II assembly language may be released in their own programs.

```
           *****************  This routine adds three ampersand
           *****************  commands to Applesoft.  The first,
           **             **  &INPUT x$, is a "defaulted input
           **  DEFAULTED  **  almost anything" command that
           **    INPUT    **  inputs up to 255 characters to any
           **             **  string variable x$.  The maximum
           **  WORD-WRAP  **  number of characters in the edit
           **    PRINT    **  string is set by the value of the
           **             **  variable FL.  The current value of
           **  GOOD GET$  **  x$ is the default.  The default
           **             **  string may be edited, then accepted
           *****************  by pressing <RETURN>.  The INPUT may
           *****************  be aborted by pressing <ESC>, which
           *                  will set the value of variable ES to
           *PUBLIC DOMAIN     one.  The &INPUT also may be aborted
           *APPLE // UTILITY  by pressing one of the apple keys in
           *  written for     conjunction with another key, in
           * "Reboot" and     which case variable OA or SA will be
           * The Sourceror's  assigned the value of the key that
           *  Apprentice      was pressed.  The second command,
           *      by          &GET x$, inputs a single keystroke.
           *Robert C. Moore   Control codes may be entered using
           *1204 Marton St.   &GET x$, and OA and SA work as
           *Laurel, MD 20707  with &INPUT x$.  The third command,
           *                  &PRINT x$, prints x$ with word-wrap.
           *Most recent code  Both 40- and 80-column text screens
           *update was done:  are supported, and the boundaries
           *March 29, 1989    of the text window are observed.
                                        ;
           *Assembled using 6502 opcodes only
                                        ;
           *Compatible with all Apple II computers
                                        ;
           *Compatible with ProDOS 8
                                        ;
           *Compatible with DOS 3.3
                                        ;
           *Zero-page usage
                                        ;
=000D        CHARAC    EQU    $0D       ;String term for STRLT2
=000E        ENDCHR    EQU    $0E       ;String term for STRLT2
=0010        DIMFLG    EQU    $10       ;Dimension flag in PTRGET
=0011        VALTYP    EQU    $11       ;Numeric: 0; String: $FF
```

```
=0012     INTFLG   EQU   $12        ;$80 if integer, else $00
=0020     WNDLFT   EQU   $20        ;Text window left
=0021     WNDWID   EQU   $21        ;Text window width
=0022     WNDTOP   EQU   $22        ;Text window top
=0023     WNDBOT   EQU   $23        ;Text window bottom + 1
=0024     CH       EQU   $24        ;40-col horizontal cursor
=0025     CV       EQU   $25        ;40-col vertical cursor
=0028     TBASE    EQU   $28        ;Text base address
=003C     SOURCE   EQU   $3C        ;Source address for move
                                    ;
=003C     KEYCOD   EQU   $3C        ;OA, SA or GET keycode
=003D     BOTCV    EQU   $3D        ;Bottom display CV
=003E     BOTCH    EQU   $3E        ;Bottom display CH
=003F     OLDCV    EQU   $3F        ;Old vertical cursor
=0040     OLDCH    EQU   $40        ;Old horizontal cursor
=0041     FLDLEN   EQU   $41        ;Maximum field length
=0042     DEST     EQU   $42        ;Dest. address for move
=0042     STRLEN   EQU   $42        ;String length
=0043     TOPCV    EQU   $43        ;V cursor for top
=0043     OAFLAG   EQU   $43        ;Open-apple flag
=0044     TOPCH    EQU   $44        ;H cursor for top
=0044     SAFLAG   EQU   $44        ;Solid-apple flag
=0045     SWITCH   EQU   $45        ;Software switch
=0046     ESCFLG   EQU   $46        ;Escape flag
=0046     TEMPX    EQU   $46        ;X-reg temporary store
=0047     TEMPY    EQU   $47        ;Y-reg temporary store
=004E     RANDOM   EQU   $4E        ;Random number
=006F     FRETOP   EQU   $6F        ;Bottom of string storage
=0073     HIMEM    EQU   $73        ;Top of free memory
=0081     VARNAM   EQU   $81        ;Variable name
=0083     VARPNT   EQU   $83        ;Variable pointer
=0085     FORPNT   EQU   $85        ;Destination string addr
=00AB     STRNG1   EQU   $AB        ;String pointer #1
                                    ;
                                    ;
                                    ; Buffer for edit string
                                    ;
=0200     EDBUF    EQU   $200       ;Buffer for edit string
                                    ;
         *Notice that because this program uses the input
         *buffer as a workspace in which to form the edit
         *string, calls to this program from immediate mode
         *will almost always end in a ?SYNTAX ERROR.  This
         *program was designed for use in deferred mode only.
                                    ;
                                    ;
                                    ; Ampersand hook
                                    ;
=03F5     AMPERH   EQU   $3F5       ;Ampersand hook
                                    ;
                                    ;
                                    ; Screen hole usage
                                    ;
=057B     CH80     EQU   $57B       ;80-col horizontal cursor
                                    ;
                                    ;
                                    ; BASIC.SYSTEM entry points
                                    ;
=BE09     ERROUT   EQU   $BE09      ;BASIC error handler
=BEF5     GETBUFR  EQU   $BEF5      ;Get buffer space
                                    ;
                                    ;
                                    ; ProDOS entry point
                                    ;
=BF00     PROMLI   EQU   $BF00      ;ProDOS M.L. Interface
                                    ;
                                    ;
```

```
                                        ; Hardware page usage
                                        ;
=C000     KEYBD     EQU   $C000         ;Keyboard data & strobe
=C001     STORE80   EQU   $C001         ;PAGE2 switches 1 and 1X
=C010     STROBE    EQU   $C010         ;Clear keyboard strobe
=C01F     RD80COL   EQU   $C01F         ;Read 80-col switch
=C054     PAGE1     EQU   $C054         ;Select page 1
=C055     PAGE2     EQU   $C055         ;Select page 2 (or 1X)
=C061     READOA    EQU   $C061         ;Read open-apple key
=C062     READSA    EQU   $C062         ;Read solid-apple key
                                        ;
                                        ;
                                        ; Applesoft entry points
                                        ;
=00B1     CHRGET    EQU   $00B1         ;Get next character
=00B7     CHRGOT    EQU   $00B7         ;Get current character
=D412     ERROR     EQU   $D412         ;Process error code in X
=D539     GDBUFS    EQU   $D539         ;Form string in EDBUF
=EB27     STORE     EQU   $EB27         ;(FAC) to real variable
                                        ;at address FORPNT
=DA7B     PERMST    EQU   $DA7B         ;Make temp str permanent
=DD6C     CHKSTR    EQU   $DD6C         ;Check for string var
=DEC9     SYNERR    EQU   $DEC9         ;Report syntax error
=DFE3     PTRGET    EQU   $DFE3         ;Get pointer to variable
=E04F     VARLOC    EQU   $E04F         ;Locate real variable
=E301     SNGFLT    EQU   $E301         ;Float unsigned int (Y)
=E3ED     STRLT2    EQU   $E3ED         ;Build string descriptor
=E6FB     CONINT    EQU   $E6FB         ;Convert (FAC) to byte
=EAF9     MOVFM     EQU   $EAF9         ;Move (Y,A) into FAC
                                        ;
                                        ;
                                        ; Applesoft keyword tokens
                                        ;
=0084     INPTKN    EQU   $84           ;Token for "INPUT"
=00BA     PRNTKN    EQU   $BA           ;Token for "PRINT" or "?"
=00BE     GETTKN    EQU   $BE           ;Token for "GET"
                                        ;
                                        ;
                                        ; Monitor entry points
                                        ;
=FBDD     BEEP      EQU   $FBDD         ;Beep speaker
=FC22     VTAB      EQU   $FC22         ;Vertical tab
=FDED     COUT      EQU   $FDED         ;Output a character
                                        ;
                                        ;
                                        ; Initial load address for main program
                                        ;
=2100     INITAD    EQU   $2100         ;Initial load address
                                        ;for main program must
                                        ;be on a page boundary
                                        ;(i.e., $xx00).
                                        ;
                                        ;
                                        ;Length of installation code
                                        ;
=006A     INSTAL    EQU   $6A           ;Installer length
                                        ;
                                        ;
                    ORG   INITAD-INSTAL ;Initial load address
                                        ; for object code
                                        ;
                                        ;
          *During installation the installation code and the
          *main program are BLOADed into INITAD-INSTAL.  The
          *memory from that location through INITAD+$3FF is
          *used temporarily.  The value of INITAD should be
          *chosen so that the installation process doesn't
          *clobber anything important.  As an example, if
```

```
                      *INITAD=$2100, memory from $2096 through $24FF will
                      *be used as a temporary buffer during installation.
                                                 ;
                                                 ;
                                                 ;          INSTALLATION CODE
                                                 ;
                     *The installer lowers HIMEM by $400 (DOS3.3) or re-
                     *quests a 4-page buffer (ProDOS BASIC.SYSTEM).  The
                     *main program then is relocated above HIMEM, and the
                     *Applesoft ampersand hook is vectored to it.  (The
                     *&-hook is chained to whatever ampersand routines
                     *were installed previously.)  The main program re-
                     *duces the amount of free memory by 1024 bytes.
                     *Under ProDOS, a call to FREEBUFR ($BEF8) will re-
                     *move this program from memory without resetting
                     *the ampersand hook at $3F5; so if you "disinstall"
                     *by calling FREEBUFR (CALL 48888), be very careful
                     *to reset the ampersand hook!  No peace-loving
                     *human being ever calls FREEBUFR, unless it is to
                     *disinstall a block of code he himself recently
                     *installed.  A word to the wise is sufficient.
                                                 ;
                     *To install the program, simply execute the following
                     *(this assumes the object file is INPUT.PRINT.GET):
                     *DOS3.3 command: PRINT CHR$(4);"BRUN INPUT.PRINT.GET"
                     *or with ProDOS: PRINT CHR$(4);"-INPUT.PRINT.GET".
                                                 ;
                     *Notice that, under DOS3.3, the pointer to the bottom
                     *(of string storage (FRETOP) will be set equal to the
                     *pointer to the top of string storage (HIMEM).  This
                     *assumes that no strings have been created at the
                     *time the installation code is executed.  Make sure
                     *that the BRUN INPUT.PRINT.GET command is executed
                     *before any strings have been created.
                                                 ;
002096: AD 00 BF              LDA   PROMLI       ;Are we under ProDOS?
002099: C9 4C                 CMP   #$4C         ;JMP op-code if ProDOS
00209B: F0 11 =20AE           BEQ   PRODOS
                                                 ;
00209D: 38                    SEC                ;It's DOS3.3, so
00209E: A5 74                 LDA   HIMEM+1      ;lower HIMEM by $400.
0020A0: E9 04                 SBC   #4
0020A2: 85 74                 STA   HIMEM+1
0020A4: 85 70                 STA   FRETOP+1     ;FRETOP too!
                                                 ;
                     *(Assumes no string assignments have been made.)
                     *Accumulator now holds high byte of buffer addr.
                                                 ;
0020A6: A0 00                 LDY   #0           ;Force low byte to zero
0020A8: 84 73                 STY   HIMEM        ;to simplify relocation.
0020AA: 84 6F                 STY   FRETOP
0020AC: F0 0A =20B8           BEQ   L0           ;Always taken
                                                 ;
0020AE: A9 04        PRODOS   LDA   #4           ;Request 4 256-byte pages
0020B0: 20 F5 BE              JSR   GETBUFR      ;using GETBUFR.
                                                 ;
                     *Accumulator now holds high byte of buffer addr.
                                                 ;
0020B3: 90 03 =20B8           BCC   L0           ;Continue if no error,
0020B5: 4C 09 BE              JMP   ERROUT       ;else exit thru ERROUT.
                                                 ;
0020B8: AC F5 03     L0       LDY   AMPERH       ;Chain into the
0020BB: 8C 14 21              STY   OLDHOOK      ;ampersand hook.
0020BE: AC F6 03              LDY   AMPERH+1
0020C1: 8C 15 21              STY   OLDHOOK+1
0020C4: AC F7 03              LDY   AMPERH+2
0020C7: 8C 16 21              STY   OLDHOOK+2
0020CA: 8D C1 23              STA   JMP1+2       ;Fix JMP instructions
```

```
0020CD: 8D C6 23             STA   JMP2+2
                                              ;
                 * JMP1 and JMP2 are the only
                 * instructions in the main program
                 *that reference addresses in the
                 *first page of the main program.
                 *The high-order bytes of these
                 *addresses need to be adjusted.
                                              ;
0020D0: 8D F7 03             STA   AMPERH+2   ;Fix ampersand hook.
0020D3: 18                   CLC
0020D4: 69 01                ADC   #1         ;Step to 2nd page of main
0020D6: 8D A2 23             STA   JMP3+2     ;Fix JMP instructions
                                              ;
                 *JMP3 is the only instruction in
                 *the main program that references
                 *an address in the second page of
                 *the main program.  The high-order
                 *byte of this address needs to be
                 *adjusted.
                                              ;
0020D9: 69 02                ADC   #2         ;Step to 3rd page of main
0020DB: 85 43                STA   DEST+1     ;DEST=BUFFER+$300
0020DD: A9 24                LDA   #>INITAD+$300 ;SOURCE=INITAD+$300
0020DF: 85 3D                STA   SOURCE+1
0020E1: A0 00                LDY   #0         ;Initialize index
0020E3: 84 42                STY   DEST
0020E5: 84 3C                STY   SOURCE
0020E7: 8C F6 03             STY   AMPERH+1
0020EA: A9 4C                LDA   #$4C       ;JMP op-code
0020EC: 8D F5 03             STA   AMPERH     ;(Just to be certain!)
0020EF: A2 04                LDX   #4         ;Move 4 256-byte pages
0020F1: B1 3C         MOVE   LDA   (SOURCE),Y ;Get a byte
0020F3: 91 42                STA   (DEST),Y   ;Relocate it
0020F5: C8                   INY
0020F6: D0 F9 =20F1          BNE   MOVE       ;Back until page is done
0020F8: C6 3D                DEC   SOURCE+1   ;Step to next page
0020FA: C6 43                DEC   DEST+1
0020FC: CA                   DEX              ;Decrement page counter
0020FD: D0 F2 =20F1          BNE   MOVE       ;Back if not done
0020FF: 60                   RTS              ;Installation complete!
                                              ;
                                              ;
                                              ;         MAIN PROGRAM
                                              ;
                 *The main program parses the text that follows the
                 *ampersand and responds accordingly.  If an INPUT,
                 *GET, or a PRINT token is not found, control is
                 *passed to any previously installed ampersand routine.
                 *With &GET x$, the current value of x$ is not printed.
                 *With &INPUT x$ and &PRINT x$, the current value of
                 *x$ is printed to the text screen window beginning
                 *at the current cursor location, with (&PRINT) or
                 *without (&INPUT) word-wrapping.  With &INPUT, this
                 *default string then may be edited using the blinking
                 *underscore "insert" cursor and the following keys:
                                              ;
                 *ARROW keys move the blinking underscore cursor.
                 *If the string occupies more than one line,
                 *the up- and down-arrow keys will work.
                 *DELETE deletes character to left of cursor.
                 *CTRL-D deletes character under the cursor.
                 *CTRL-X erases the edit string.
                 *CTRL-Y clears from cursor to end of edit string.
                 *CTRL-B moves cursor to beginning of edit string.
                 *CTRL-N moves cursor to end of edit string.
                 *CTRL-C toggles the case of the character under
```

```
                        *the cursor; if it is a letter:
                        *upper case letters are converted to lower;
                        *lower case letters are converted to upper.
                        *RETURN accepts the current edit string and
                        *assigns it to the variable, x$.
                        *ESC aborts with a variable (ES) set to "1".
                        *OPEN-APPLE (in conjunction with another key)
                        *aborts with a variable (OA) set to
                        *the code for the key that was pressed.
                        *SOLID-APPLE (in conjunction with another key)
                        *aborts with a variable (SA) set to
                        *the code for the key that was pressed.
                                                 ;
                        *If the specified string, x$, has a length that ex-
                        *ceeds the specified maximum field_length, FL, then a
                        *STRING TOO LONG error will be generated.  The same
                        *error will be generated if the edit string ever
                        *grows so long that its top line scrolls out of the
                        *text window.  The text window must be at least two
                        *characters wide.  If &INPUT is aborted by pressing
                        *<ESC>, this may be detected using an ON ESCAPE GOTO
                        *statement.  An apple-key combination may be detected
                        *using an IF OA GOTO or IF SA GOTO statement.  When
                        *nonzero, the value of OA or SA is the hi-ASCII
                        *keycode. All three ampersand routines leave the
                        *text screen cursor just beyond the end of the
                        *printed string.  A blinking underscore cursor is
                        *used during &GET and &INPUT editing.  When control
                        *returns to Applesoft, the text cursor always will
                        *be restored to whatever cursor was in use at the
                        *time the ampersand routine was invoked.  If no
                        *FL variable was defined prior to &INPUT, or if
                        *the value of FL had been set equal to zero, the
                        *field length defaults to 255 characters.  Zero-page
                        *locations $3C through $47 are used temporarily by
                        *this program; their original contents are destroyed.
                        *If the variables FL, ES, OA, and SA do not exist
                        *prior to invoking &INPUT, &GET, or &PRINT, they
                        *will be created for you, and FL will default to
                        *zero (which indicates field_length = 255).  When
                        *terminated by <RETURN>, &INPUT strips trailing
                        *spaces from the edit string before making a new x$.
                        *Following &GET, PEEK(60) will yield the Hi-ASCII
                        *code for the character in x$; PEEK(60)-128 will
                        *give the Lo-ASCII code.  &GET always clears the
                        *variable ES to zero, even if the key that was
                        *"gotten" was <ESC>.  OA and SA behave exactly the
                        *same with &GET as they do with &INPUT, except that
                        *the GET is not aborted.  If you &GET an apple key
                        *combination, x$ receives the character, location
                        *60 receives the Hi-ASCII code, and OA and/or SA
                        *also receive(s) the Hi-ASCII code.  To determine
                        *if <ESC> was pressed during an &GET, use
                        *ON (PEEK(60)=155) GOTO instead of ON ESCAPE GOTO.
                        *&INPUT will respond to all ASCII codes except
                        *control codes and DELETE.
                                                 ;
002100: 20 B7 00                JSR    CHRGOT    ;Get character after "&"
002103: A2 FF                   LDX    #$FF      ;Flag value for &INPUT
002105: C9 84                   CMP    #INPTKN   ;Compare to INPUT token
002107: F0 0E =2117             BEQ    L1        ;If &INPUT, SWITCH=$FF
002109: E8                      INX              ;Flag for &PRINT ($00)
00210A: C9 BA                   CMP    #PRNTKN   ;Compare to PRINT token
00210C: F0 09 =2117             BEQ    L1        ;If &PRINT, SWITCH=$00
00210E: A2 FE                   LDX    #$FE      ;Flag for &GET ($FE)
```

```
002110: C9 BE                   CMP   #GETTKN    ;Compare to GET token
002112: F0 03 =2117             BEQ   L1         ;If &GET, SWITCH=$FE
002114: 4C C9 DE      OLDHOOK   JMP   SYNERR     ;Old &-hook stored here
                                                 ;
002117: 86 45         L1        STX   SWITCH     ;Set switch
                                                 ;
                      *SWITCH = $00 for &PRINT
                      *SWITCH = $FE for &GET
                      *SWITCH = $FF for &INPUT
                                                 ;
002119: A0 00                   LDY   #0         ;Default STRLEN to 0
00211B: 84 42                   STY   STRLEN
                                                 ;
00211D: 84 10                   STY   DIMFLG     ;Initialize flags
00211F: 84 11                   STY   VALTYP     ;Numeric, not string
002121: 84 12                   STY   INTFLG     ;Real, not integer
                                                 ;
                      ***Find value of variable FL***
002123: A9 46                   LDA   #$46       ;Lo-ASCII 'F'
002125: 85 81                   STA   VARNAM
002127: A9 4C                   LDA   #$4C       ;Lo-ASCII 'L'
002129: 85 82                   STA   VARNAM+1
00212B: 20 4F E0                JSR   VARLOC     ;Locate the variable FL
00212E: 20 F9 EA                JSR   MOVFM      ;Move (Y,A) to FAC
002131: 20 FB E6                JSR   CONINT     ;Integer in X reg
002134: 8A                      TXA              ;Examine value of FL
002135: D0 01 =2138             BNE   STORFL
002137: CA                      DEX              ;Default to 255
002138: 86 41         STORFL    STX   FLDLEN     ;Store field_length
                                                 ;
                      ***Locate string variable***
00213A: 20 B1 00                JSR   CHRGET     ;Advance TXTPTR
00213D: 20 E3 DF                JSR   PTRGET     ;Get ptr to str descript
002140: 85 85                   STA   FORPNT     ;Save pointer in FORPNT
002142: 84 86                   STY   FORPNT+1   ;for later use by PERMST.
002144: 20 6C DD                JSR   CHKSTR     ;Check for string var
002147: 20 B7 00                JSR   CHRGOT     ;Examine next character
00214A: F0 03 =214F             BEQ   SYNTOK     ;Branch if : or EOL
00214C: 4C C9 DE                JMP   SYNERR     ;Error if not : or EOL
                                                 ;
00214F: A5 24         SYNTOK    LDA   CH         ;Update text cursor
002151: F0 03 =2156             BEQ   L2         ;Update CH80 only if
002153: 8D 7B 05                STA   CH80       ;CH > 0.
002156: A4 25         L2        LDY   CV
002158: 2C 1F C0                BIT   RD80COL    ;80-column display?
00215B: 10 03 =2160             BPL   SAVCUR
00215D: AD 7B 05                LDA   CH80
002160: 84 43         SAVCUR    STY   TOPCV      ;Cursor values for
002162: 85 44                   STA   TOPCH      ;start of string
                                                 ;
002164: A0 00         REDO      LDY   #0         ;Initialize index
002166: 84 47                   STY   TEMPY
002168: A6 45                   LDX   SWITCH
00216A: E0 FE                   CPX   #$FE       ;Is this &GET?
00216C: F0 04 =2172             BEQ   GETPNT     ;If so, use default.
00216E: B1 83                   LDA   (VARPNT),Y ;Get length of string
002170: 85 42                   STA   STRLEN     ;Store in string_length
002172: C8            GETPNT    INY              ;Step to next character
002173: B1 83                   LDA   (VARPNT),Y ;LOB of pointer
002175: 85 AB                   STA   STRNG1     ;STRNG1 points to x$
002177: C8                      INY              ;Step to next character
002178: B1 83                   LDA   (VARPNT),Y ;HOB of pointer
00217A: 85 AC                   STA   STRNG1+1
```

```
00217C: A0 00                  LDY   #0          ;Reset index
00217E: A5 45                  LDA   SWITCH      ;Test switch
002180: D0 4C =21CE            BNE   PRNWRD      ;INPUT or GET: go print
002182: 84 47          L3      STY   TEMPY       ;Else word-wrap
002184: A2 01                  LDX   #1          ;Initialize char count
002186: C0 00                  CPY   #0          ;No leading space at
002188: D0 01 =218B            BNE   L4          ;start of string.
00218A: CA                     DEX
00218B: C4 42          L4      CPY   STRLEN      ;Reached end of string?
00218D: B0 09 =2198            BCS   CHKWRD      ;Branch if yes
00218F: C8                     INY
002190: E8                     INX               ;Increment word length
002191: B1 AB                  LDA   (STRNG1),Y
002193: C9 20                  CMP   #$20        ;Lo-ASCII space?
002195: D0 F4 =218B            BNE   L4          ;No.  Keep going.
002197: 38                     SEC               ;Yes, prepare to SBC.
002198: A5 21          CHKWRD  LDA   WNDWID      ;Get window width
00219A: 2C 1F C0               BIT   RD80COL     ;80-column display?
00219D: 10 04 =21A3            BPL   L5
00219F: ED 7B 05               SBC   CH80        ;Compute dist to R edge
0021A2: 2C                     DFB   $2C         ;Skip next instruction
0021A3: E5 24          L5      SBC   CH
0021A5: 86 46                  STX   TEMPX
0021A7: AA                     TAX               ;Save distance to go
0021A8: C5 46                  CMP   TEMPX       ;Will it fit?
0021AA: B0 08 =21B4            BCS   L7          ;Yes.  Go print it.
                                                 ;
0021AC: A9 A0          L6      LDA   #$A0        ;(Hi-ASCII space)
0021AE: 20 ED FD               JSR   COUT        ;Pad with spaces
0021B1: CA                     DEX               ;to end of line.
0021B2: D0 F8 =21AC            BNE   L6
                                                 ;
0021B4: A4 47          L7      LDY   TEMPY       ;Restore index
0021B6: F0 16 =21CE            BEQ   PRNWRD      ;No space if 1st char
0021B8: A5 24                  LDA   CH
0021BA: 2C 1F C0               BIT   RD80COL     ;80-column display?
0021BD: 10 03 =21C2            BPL   L8
0021BF: AD 7B 05               LDA   CH80        ;Get horiz cursor
0021C2: C9 00          L8      CMP   #0          ;At L edge?
0021C4: F0 08 =21CE            BEQ   PRNWRD      ;Yes.  Don't print space.
0021C6: A9 A0                  LDA   #$A0        ;Hi-ASCII space
0021C8: 99 FF 01               STA   EDBUF-1,Y   ;Put copy in EDBUF
0021CB: 20 ED FD               JSR   COUT        ;Print a space
                                                 ;
0021CE: C4 42          PRNWRD  CPY   STRLEN      ;At end of string?
0021D0: B0 3D =220F            BCS   ENDPRT      ;Yes.  Finished.
0021D2: B1 AB                  LDA   (STRNG1),Y  ;Get next character
0021D4: 09 80                  ORA   #$80        ;Convert to hi-ASCII
0021D6: C8                     INY               ;Increment string index
0021D7: A6 45                  LDX   SWITCH      ;Test switch
0021D9: D0 04 =21DF            BNE   L9          ;No wrap if &INPUT
0021DB: C9 A0                  CMP   #$A0        ;Hi-ASCII space
0021DD: F0 A3 =2182            BEQ   L3          ;Go see if next word fits
0021DF: 99 FF 01       L9      STA   EDBUF-1,Y   ;Put copy in EDBUF
0021E2: A6 25                  LDX   CV
0021E4: 20 ED FD               JSR   COUT        ;Print the character
0021E7: E8                     INX
0021E8: E4 23                  CPX   WNDBOT      ;Were we on bottom line?
0021EA: D0 E2 =21CE            BNE   PRNWRD      ;No, no scroll was done.
                                                 ;
0021EC: A6 24                  LDX   CH          ;Get horizontal cursor
0021EE: 2C 1F C0               BIT   RD80COL     ;80-column display?
0021F1: 10 03 =21F6            BPL   L10
```

```
0021F3: AE 7B 05              LDX   CH80       ;Use CH80 instead
0021F6: E0 00          L10    CPX   #0         ;Are we at L edge?
0021F8: D0 D4 =21CE           BNE   PRNWRD     ;No, no scroll was done.
0021FA: A6 45                 LDX   SWITCH     ;If &PRINT,
0021FC: F0 D0 =21CE           BEQ   PRNWRD     ;keep printing.
                                               ;
                       * We don't care if top line scrolls out of
                       * the text window during &PRINT, but we
                       * must flag it as an error during &INPUT.
                                               ;
0021FE: C6 3F                 DEC   OLDCV      ;If scroll due to INSERT
002200: C6 43                 DEC   TOPCV      ;Modify start cursor
002202: 30 06 =220A           BMI   L11        ;Error if negative
002204: A6 43                 LDX   TOPCV
002206: E4 22                 CPX   WNDTOP     ;Did top scroll off?
002208: B0 C4 =21CE           BCS   PRNWRD     ;No.  Keep going.
                                               ;
00220A: A2 B0          L11    LDX   #$B0       ;Code for STRING TOO LONG
00220C: 4C 12 D4              JMP   ERROR      ;Exit thru error process
                                               ;
00220F: A5 24          ENDPRT LDA   CH         ;Store CH in BOTCH
002211: 2C 1F C0              BIT   RD80COL    ;80-column display?
002214: 10 03 =2219           BPL   STRCH
002216: AD 7B 05              LDA   CH80
002219: 85 3E          STRCH  STA   BOTCH
00221B: A5 25                 LDA   CV         ;Store CV in BOTCV
00221D: 85 3D                 STA   BOTCV
                                               ;
00221F: 24 45          TESTSW BIT   SWITCH     ;Test switch
                                               ;
                       *SWITCH = $00 indicates &PRINT.
                       *SWITCH = $FF indicates initial &INPUT entry.
                       *SWITCH = $FE indicates &GET.
                       *SWITCH = $40 indicates return from <CTRL-R>.
                       *SWITCH = $80 indicates return from <CTRL-X>,
                       *<CTRL-Y>, <CTRL-D>, or INSERT.
                                               ;
002221: 30 03 =2226           BMI   L12
002223: 70 09 =222E           BVS   CTRLR      ;Return from <CTRL-R>
002225: 60                    RTS              ;&PRINT is done!
                                               ;
002226: 50 14 =223C    L12    BVC   RESTORE    ;Return from <CTRL-Y>,
                                               ;<CTRL-D>, or INSERT.
                                               ;
002228: A5 41                 LDA   FLDLEN     ;FLDLEN < STRLEN?
00222A: C5 42                 CMP   STRLEN
00222C: 90 DC =220A           BCC   L11        ;Yes.  STRING TOO LONG.
                                               ;
00222E: A5 44          CTRLR  LDA   TOPCH      ;Cursor to top
002230: 85 40                 STA   OLDCH
002232: A5 43                 LDA   TOPCV
002234: 85 3F                 STA   OLDCV
002236: A0 00                 LDY   #0         ;Index to beginning
002238: 84 47                 STY   TEMPY
00223A: 84 46                 STY   ESCFLG     ;Clear escape flag to "0"
                                               ;
00223C: A0 00          RESTORE LDY  #<EDBUF    ;Aim STRNG1 at EDBUF
00223E: 84 AB                 STY   STRNG1
002240: A9 02                 LDA   #>EDBUF
002242: 85 AC                 STA   STRNG1+1
                                               ;
002244: A5 3F                 LDA   OLDCV      ;Get previous CV
002246: 85 25                 STA   CV         ;Store in current CV
```

```
002248: A5 40                      LDA   OLDCH       ;Get previous CH
00224A: 85 24                      STA   CH          ;Store in current CH
00224C: 8D 7B 05                   STA   CH80
                                                     ;
00224F: 20 22 FC          GETCHR   JSR   VTAB        ;Update TBASE
002252: A4 24                      LDY   CH          ;Get CH
002254: 8C 7B 05                   STY   CH80        ;Update CH80
002257: 2C 1F C0                   BIT   RD80COL     ;80-column display?
00225A: 10 10 =226C                BPL   GETCH2
00225C: 8D 01 C0                   STA   STORE80     ;PAGE2 switches 1 and 1X
00225F: 98                         TYA
002260: 45 20                      EOR   WNDLFT      ;LSB=1 if char in main
002262: 4A                         LSR               ;Carry clear if aux
002263: B0 04 =2269                BCS   GETCH1
002265: 8D 55 C0                   STA   PAGE2       ;Select AUX memory
002268: C8                         INY               ;If WNDLFT odd
002269: 98                GETCH1   TYA
00226A: 4A                         LSR               ;Compute index
00226B: A8                         TAY
00226C: B1 28             GETCH2   LDA   (TBASE),Y   ;Get the character
00226E: 48                         PHA               ;Save original character
00226F: 49 DF                      EOR   #$DF        ;(Hi-ASCII underscore)
002271: D0 02 =2275                BNE   NOZMSK      ;If screen char is "_",
002273: A9 7F                      LDA   #$7F        ;treat as if space.
002275: 48                NOZMSK   PHA               ;Mask onto stack
002276: 68                GETCH3   PLA               ;Retrieve mask
002277: 48                         PHA               ;Toggle between
002278: 51 28                      EOR   (TBASE),Y   ;original character
00227A: 91 28                      STA   (TBASE),Y   ;and underscore.
00227C: 2C 00 C0          GETCH4   BIT   KEYBD       ;See if key pressed
00227F: 30 12 =2293                BMI   GOTKEY
002281: E6 4E                      INC   RANDOM      ;Use random # as a
002283: D0 F7 =227C                BNE   GETCH4      ;flashing cursor timer.
002285: A5 4F                      LDA   RANDOM+1
002287: E6 4F                      INC   RANDOM+1
002289: 45 4F                      EOR   RANDOM+1    ;Leaves 1 if bit changed
00228B: 29 40                      AND   #%01000000  ;Did bit six change?
00228D: F0 ED =227C                BEQ   GETCH4
00228F: D0 E5 =2276                BNE   GETCH3      ;Always taken
                                                     ;
002291: F0 9B =222E       CTRLB0   BEQ   CTRLR       ;Bounce-back point
                                                     ;
002293: 68                GOTKEY   PLA               ;Remove mask from stack
002294: 68                         PLA               ;Retrieve original char
002295: 91 28                      STA   (TBASE),Y   ;Put it back
002297: AD 00 C0                   LDA   KEYBD       ;Get key code
00229A: 85 3C                      STA   KEYCOD      ;Save it for later
                                                     ;
00229C: A2 FF                      LDX   #$FF
00229E: 2C 61 C0                   BIT   READOA      ;Check open-apple key
0022A1: 10 04 =22A7                BPL   CHKSA
0022A3: 86 43                      STX   OAFLAG      ;Set open-apple flag
0022A5: A9 8D                      LDA   #$8D        ;Fake a <RETURN>
                                                     ;
0022A7: 2C 62 C0          CHKSA    BIT   READSA      ;Check solid-apple key
0022AA: 10 04 =22B0                BPL   CHKGET
0022AC: 86 44                      STX   SAFLAG      ;Set solid-apple flag
0022AE: A9 8D                      LDA   #$8D        ;Fake a <RETURN>
                                                     ;
0022B0: A6 45             CHKGET   LDX   SWITCH
0022B2: E0 FE                      CPX   #$FE        ;Is this an &GET?
0022B4: D0 0D =22C3                BNE   CHKCAS
0022B6: A6 3C                      LDX   KEYCOD      ;Get keycode
0022B8: 8E 00 02                   STX   EDBUF       ;Put it in buffer
```

```
0022BB: E6 42                  INC   STRLEN     ;Set string_length = 1
0022BD: A9 8D                  LDA   #$8D       ;Fake a <RETURN>
0022BF: D0 29 =22EA            BNE   CONTN1     ;Always taken
                                                ;
0022C1: 90 8C =224F    GETO    BCC   GETCHR     ;Bounce-back point
                                                ;
0022C3: C9 83          CHKCAS  CMP   #$83       ;Check for CTRL-C
0022C5: D0 1F =22E6            BNE   CONTIN
0022C7: A6 47                  LDX   TEMPY      ;Process CTRL-C
0022C9: E4 42                  CPX   STRLEN     ;Must be char in string
0022CB: B0 17 =22E4            BCS   NOALPH     ;Else skip it
0022CD: B1 28                  LDA   (TBASE),Y  ;Get the character
0022CF: 09 20                  ORA   #$20       ;Force lower case
0022D1: C9 FB                  CMP   #$FB       ;Hi-ASCII "("
0022D3: B0 0F =22E4            BCS   NOALPH     ;Not an alpha
0022D5: C9 E1                  CMP   #$E1       ;Hi-ASCII "a"
0022D7: 90 0B =22E4            BCC   NOALPH     ;Not an alpha
0022D9: B1 28                  LDA   (TBASE),Y  ;Retrieve character
0022DB: 49 20                  EOR   #$20       ;Toggle its case
0022DD: 91 28                  STA   (TBASE),Y  ;Put it back
0022DF: A4 47                  LDY   TEMPY
0022E1: 99 00 02               STA   EDBUF,Y    ;Make change in string
0022E4: A9 95          NOALPH  LDA   #$95       ;Hi-ASCII R-ARROW
                                                ;
0022E6: A2 80          CONTIN  LDX   #$80
0022E8: 86 45                  STX   SWITCH     ;Default SWITCH to $80
0022EA: 8D 54 C0       CONTN1  STA   PAGE1      ;Back to pg1 if needed
                                                ;
                       *We default to text page 1 because it is
                       *assumed that text page 2 was not in use
                       *at the time this program was called.  If
                       *you wish to work with text page 2 you
                       *will have to modify the program.
                                                ;
0022ED: 8D 10 C0               STA   STROBE     ;Clear keyboard strobe
                                                ;
0022F0: C9 82                  CMP   #$82       ;Check for CTRL-B
0022F2: D0 02 =22F6            BNE   CHKDEL
0022F4: F0 9B =2291            BEQ   CTRLB0     ;Always taken
                                                ;
0022F6: C9 FF          CHKDEL  CMP   #$FF       ;Check for <DELETE>
0022F8: D0 1C =2316            BNE   L13
                                                ;
                      ***PROCESS <DELETE>***
0022FA: AA                     TAX              ;Leave signature ($FF)
                                                ;
                      ***MOVE CURSOR LEFT***
                      *A "signature" in the X register indicates which
                      *key processor transferred to CURLFT:
                      *X=$88 indicates <L-ARROW>
                      *X=$FF indicates <DELETE>
0022FB: A4 47          CURLFT  LDY   TEMPY      ;Get string index
0022FD: F0 2B =232A            BEQ   REJECT     ;If at L end, no go.
0022FF: C6 47                  DEC   TEMPY      ;Decrement string index
002301: A4 24                  LDY   CH         ;Get CH
002303: D0 06 =230B            BNE   CURL
002305: C6 25                  DEC   CV         ;Step up one line
002307: C6 3F                  DEC   OLDCV      ;Update old CV
002309: A4 21                  LDY   WNDWID     ;Step to R edge
00230B: 88             CURL    DEY              ;Move left 1 character
00230C: 84 24                  STY   CH         ;Update CH
00230E: 84 40                  STY   OLDCH      ;Update old CH
002310: 8A                     TXA              ;Check signature
```

```
002311: 4A                         LSR                ;Examine LSB
002312: B0 58 =236C                BCS   CTRLD        ;Process ctrl-D
002314: 90 AB =22C1      GET1      BCC   GET0         ;Always taken
                                                      ;
002316: C9 88            L13       CMP   #$88         ;Check <L-ARROW>
002318: D0 03 =231D                BNE   CHAR
                                                      ;
                         ***PROCESS <L-ARROW>***
00231A: AA                         TAX                ;Leave signature ($88)
00231B: D0 DE =22FB                BNE   CURLFT       ;Always taken
                                                      ;
00231D: C9 A0            CHAR      CMP   #$A0         ;Character to insert?
00231F: 90 40 =2361                BCC   CONTRL       ;Control character
                                                      ;
                         ***PROCESS INSERT***
002321: A4 42                      LDY   STRLEN
002323: F0 16 =233B                BEQ   L16
002325: AA                         TAX                ;Save char for later
002326: C4 41                      CPY   FLDLEN       ;STRLEN < FLDLEN?
002328: 90 0C =2336                BCC   L15          ;Yes.  Continue.
                                                      ;
00232A: 20 DD FB         REJECT    JSR   BEEP         ;Beep speaker
00232D: F0 2F =235E                BEQ   GETCLC       ;Always taken
                                                      ;
00232F: 88               L14       DEY
002330: B9 00 02                   LDA   EDBUF,Y      ;Open up a hole for
002333: 99 01 02                   STA   EDBUF+1,Y    ;the insertion.
002336: C4 47            L15       CPY   TEMPY
002338: D0 F5 =232F                BNE   L14
00233A: 8A                         TXA                ;Retrieve the char
00233B: 99 00 02         L16       STA   EDBUF,Y      ;Insert it
00233E: E6 42                      INC   STRLEN
002340: A2 00                      LDX   #0           ;Leave signature ($00)
                                                      ;
                         ***MOVE CURSOR RIGHT***
                         *A "signature" in the X register indicates which
                         *key processor transferred to CURRT:
                         *X=$00 indicates INSERT
                         *X=$95 indicates <R-ARROW>
002342: A4 47            CURRT     LDY   TEMPY        ;Get string index
002344: C4 42                      CPY   STRLEN       ;Is TEMPY < STRLEN?
002346: B0 E2 =232A                BCS   REJECT       ;No.  Bad news.
002348: E6 47                      INC   TEMPY
00234A: A4 24                      LDY   CH           ;Get CH
00234C: C8                         INY
00234D: C4 21                      CPY   WNDWID       ;CH < WIDTH?
00234F: 90 06 =2357                BCC   L17          ;Yes.  Go store it.
002351: A0 00                      LDY   #0           ;No.  Move to next line.
002353: E6 25                      INC   CV
002355: E6 3F                      INC   OLDCV
002357: 84 24            L17       STY   CH           ;Replace CH
002359: 84 40                      STY   OLDCH        ;Update old CH
00235B: 8A                         TXA                ;Retrieve signature
00235C: F0 4F =23AD                BEQ   REPRNT       ;If called by INSERT
00235E: 18               GETCLC    CLC                ;Force branch
00235F: 90 B3 =2314      GET2      BCC   GET1         ;Always taken
                                                      ;
002361: C9 95            CONTRL    CMP   #$95         ;Check R-arrow
002363: D0 03 =2368                BNE   L18
                                                      ;
                         ***PROCESS <R-ARROW>***
002365: AA                         TAX                ;Leave signature ($95)
002366: D0 DA =2342                BNE   CURRT        ;Always taken
```

```
                                                        ;
002368: C9 84           L18       CMP    #$84         ;Check <CTRL-D>
00236A: D0 5B =23C7               BNE    L19
                                                        ;
                        ***PROCESS <CTRL-D>***
00236C: A4 47           CTRLD     LDY    TEMPY        ;Get string index
00236E: C4 42                     CPY    STRLEN       ;Is TEMPY < STRLEN?
002370: B0 B8 =232A     REJ1      BCS    REJECT       ;No.  Bad news.
002372: B9 01 02        CTRLD1    LDA    EDBUF+1,Y    ;Get character to right
002375: 99 00 02                  STA    EDBUF,Y      ;Store it here
002378: C8                        INY                 ;Step to next character
002379: C4 42                     CPY    STRLEN       ;Reached end of string?
00237B: D0 F5 =2372               BNE    CTRLD1
00237D: A2 00                     LDX    #0           ;Leave signature ($00)
                                                        ;
                       ***ERASE STRING***
                       *A "signature" in the X register indicates which
                       *key processor transferred to ERASE:
                       *X=$00 indicates <DELETE> or <CRTL-D>
                       *X=$40 indicates <CTRL-R>
                       *X=$99 indicates <CTRL-Y> or <CTRL-X>
00237F: A5 44           ERASE     LDA    TOPCH        ;Get top CH
002381: 85 24                     STA    CH           ;Put in CH
002383: 8D 7B 05                  STA    CH80
002386: A5 43                     LDA    TOPCV        ;Get top CV
002388: 85 25                     STA    CV           ;Put in CV
00238A: 20 22 FC                  JSR    VTAB         ;Update TBASE
00238D: A4 42                     LDY    STRLEN
00238F: F0 08 =2399               BEQ    CHKSIG       ;Nothing to erase!
002391: A9 A0           ERASE1    LDA    #$A0         ;Hi-ASCII space
002393: 20 ED FD                  JSR    COUT         ;Print it
002396: 88                        DEY
002397: D0 F8 =2391               BNE    ERASE1
002399: 8A              CHKSIG    TXA                 ;Check signature
00239A: D0 07 =23A3               BNE    ERASE2
00239C: C6 42                     DEC    STRLEN       ;<DELETE> or <CTRL-D>
00239E: D0 0D =23AD               BNE    REPRNT
0023A0: 4C 1F 22        JMP3      JMP    TESTSW
                                                        ;
0023A3: 30 04 =23A9     ERASE2    BMI    ERASE3
0023A5: 85 45                     STA    SWITCH       ;<CTRL-R>: $40 to SWITCH
0023A7: D0 04 =23AD               BNE    REPRNT       ;Always taken
                                                        ;
0023A9: A4 47           ERASE3    LDY    TEMPY        ;<CTRL-Y>
0023AB: 84 42                     STY    STRLEN       ;Chop from cursor to end
                                                        ;
                        ***REPRINT STRING***
                        *A "signature" in the X register indicates which
                        *key processor transferred to REPRNT (via ERASE):
                        *X=$00 indicates INSERT (via CURRT),
                        *<CTRL-D>, or <DELETE>
                        *X=$40 indicates <CTRL-R>
                        *X=$99 indicates <CTRL-Y> or <CTRL-X>
0023AD: A5 44           REPRNT    LDA    TOPCH        ;Get top CH
0023AF: 85 24                     STA    CH           ;Put in CH
0023B1: 8D 7B 05                  STA    CH80
0023B4: A5 43                     LDA    TOPCV        ;Get top CV
0023B6: 85 25                     STA    CV           ;Put in CV
0023B8: 20 22 FC                  JSR    VTAB
0023BB: E0 40                     CPX    #$40         ;Check signature
0023BD: D0 03 =23C2               BNE    REPRN1
0023BF: 4C 64 21        JMP1      JMP    REDO         ;<CTRL-R>
0023C2: A0 00           REPRN1    LDY    #0
```

```
0023C4: 4C CE 21      JMP2     JMP   PRNWRD     ;INSERT, <CTRL-D>,
                                                ;<DELETE>, or <CTRL-Y>
                                                ;
0023C7: C9 99         L19      CMP   #$99       ;Check <CTRL-Y>
0023C9: D0 03 =23CE            BNE   L20
                                                ;
                      ***PROCESS <CTRL-Y>***
0023CB: AA                     TAX              ;Leave signature ($99)
0023CC: D0 B1 =237F   CTRLY1   BNE   ERASE      ;Always taken
                                                ;
0023CE: C9 92         L20      CMP   #$92       ;Check <CTRL-R>
0023D0: D0 06 =23D8            BNE   L21
                                                ;
                      ***PROCESS <CTRL-R>***
0023D2: A2 40                  LDX   #$40       ;Leave signature ($40)
0023D4: D0 A9 =237F            BNE   ERASE      ;Always taken
                                                ;
0023D6: 90 87 =235F   GET3     BCC   GET2       ;Bounce-back point
                                                ;
0023D8: C9 8A         L21      CMP   #$8A       ;Check <D-ARROW>
0023DA: D0 2A =2406            BNE   CHKUPA
                                                ;
                      ***PROCESS <D-ARROW>***
0023DC: A5 25                  LDA   CV         ;CV < BOTCV?
0023DE: C5 3D                  CMP   BOTCV
0023E0: 90 02 =23E4            BCC   DOWN1
0023E2: B0 8C =2370   REJ2     BCS   REJ1       ;No.  Bad news.
0023E4: E6 25         DOWN1    INC   CV         ;Step down 1 line
0023E6: 18                     CLC              ;Prepare to add
0023E7: A5 47                  LDA   TEMPY      ;WNDWID to
0023E9: 65 21                  ADC   WNDWID     ;TEMPY
0023EB: 85 47                  STA   TEMPY
0023ED: A5 25                  LDA   CV         ;CV < BOTCV?
0023EF: 85 3F                  STA   OLDCV
0023F1: C5 3D                  CMP   BOTCV
0023F3: 90 0F =2404            BCC   DOWN3
0023F5: A5 3E                  LDA   BOTCH      ;No.  Beyond end?
0023F7: C5 24                  CMP   CH
0023F9: B0 08 =2403            BCS   DOWN2
0023FB: 85 24                  STA   CH         ;Yes.  Go back
0023FD: 85 40                  STA   OLDCH      ;to bottom.
0023FF: A5 42                  LDA   STRLEN
002401: 85 47                  STA   TEMPY
002403: 18            DOWN2    CLC
002404: 90 D0 =23D6   DOWN3    BCC   GET3       ;Always taken
                                                ;
002406: C9 8B         CHKUPA   CMP   #$8B       ;Check <U-ARROW>
002408: D0 2D =2437            BNE   CHKCTX
                                                ;
                      ***PROCESS <U-ARROW>***
00240A: A5 43                  LDA   TOPCV      ;TOPCV < CV?
00240C: C5 25                  CMP   CV
00240E: 90 02 =2412            BCC   UPARR1
002410: B0 D0 =23E2   REJ3     BCS   REJ2       ;No.  Bad news.
002412: C6 25         UPARR1   DEC   CV         ;Step up 1 line
002414: A5 25                  LDA   CV
002416: 85 3F                  STA   OLDCV
002418: 38                     SEC
002419: A5 47                  LDA   TEMPY
00241B: E5 21                  SBC   WNDWID
00241D: 85 47                  STA   TEMPY
00241F: A5 43                  LDA   TOPCV
002421: C5 25                  CMP   CV         ;TOPCV < CV?
```

```
002423: 90 DF =2404  884            BCC   DOWN3
002425: A5 24        885            LDA   CH          ;No.  Left of top?
002427: C5 44        886            CMP   TOPCH
002429: B0 D8 =2403  887            BCS   DOWN2
00242B: A5 44        888            LDA   TOPCH       ;Yes.  Go to top.
00242D: 85 24        889            STA   CH
00242F: 85 40        890            STA   OLDCH
002431: A9 00        891            LDA   #0
002433: 85 47        892            STA   TEMPY
002435: F0 CC =2403  893            BEQ   DOWN2       ;Always taken
                     894                              ;
002437: C9 98        895  CHKCTX    CMP   #$98        ;Check <CTRL-X>
002439: D0 10 =244B  896            BNE   CHKESC
                     897                              ;
                     898 ***PROCESS <CTRL-X>***
00243B: AA           899            TAX
00243C: A9 00        900            LDA   #0          ;Go to the top
00243E: 85 47        901            STA   TEMPY
002440: A5 44        902            LDA   TOPCH
002442: 85 40        903            STA   OLDCH
002444: A5 43        904            LDA   TOPCV
002446: 85 3F        905            STA   OLDCV
002448: E8           906            INX               ;$99 to X register
002449: D0 81 =23CC  907            BNE   CTRLY1      ;Always taken
                     908                              ;
00244B: C9 9B        909  CHKESC    CMP   #$9B        ;Check <ESC>
00244D: D0 04 =2453  910            BNE   CHKCTN
                     911                              ;
                     912  ***PROCESS <ESC>***
00244F: E6 46        913            INC   ESCFLG      ;Escape flag to "1"
002451: D0 48 =249B  914            BNE   ESCENT      ;Always taken
                     915                              ;
002453: C9 8E        916  CHKCTN    CMP   #$8E        ;Check <CTRL-N>
002455: D0 13 =246A  917            BNE   CHKRTN
                     918                              ;
                     919 ***PROCESS <CTRL-N>***
002457: A4 42        920            LDY   STRLEN      ;Go to bottom
002459: 84 47        921            STY   TEMPY       ;of string.
00245B: A5 3D        922            LDA   BOTCV
00245D: 85 25        923            STA   CV
00245F: 85 3F        924            STA   OLDCV
002461: A5 3E        925            LDA   BOTCH
002463: 85 24        926            STA   CH
002465: 85 40        927            STA   OLDCH
002467: 18           928            CLC
002468: 90 9A =2404  929            BCC   DOWN3       ;Always taken
                     930                              ;
00246A: C9 8D        931  CHKRTN    CMP   #$8D        ;Check <RETURN>
00246C: F0 03 =2471  932            BEQ   RETURN
00246E: 38           933            SEC
00246F: B0 9F =2410  934            BCS   REJ3        ;Always taken
                     935                              ;
                     936  ***PROCESS <RETURN>***
002471: A5 45        937  RETURN    LDA   SWITCH      ;Check &GET
002473: C9 FE        938            CMP   #$FE
002475: F0 08 =247F  939            BEQ   FORMST      ;If &GET, form string
002477: 24 43        940            BIT   OAFLAG      ;Else check for
002479: 30 20 =249B  941            BMI   ESCENT      ;open-apple or
00247B: 24 44        942            BIT   SAFLAG      ;solid-apple
00247D: 30 1C =249B  943            BMI   ESCENT      ;abort of &INPUT.
                     944                              ;
00247F: A6 42        945  FORMST    LDX   STRLEN      ;Get string_length
002481: F0 0A =248D  946            BEQ   RTN2        ;Length = zero?
```

```
002483: A9 A0                  LDA    #$A0        ;Delete trailing spaces
002485: DD FF 01        RTN1   CMP    EDBUF-1,X   ;Is character a space?
002488: D0 03 =248D            BNE    RTN2        ;No.  Go create string.
00248A: CA                     DEX                ;Yes.  Strip it off.
00248B: D0 F8 =2485            BNE    RTN1        ;Go back if more chars

00248D: 20 39 D5        RTN2   JSR    GDBUFS      ;Form string in EDBUF
                                                  ;
                        *GDBUFS puts a null ($00) at the
                        *end of the string in EDBUF and
                        *masks off the MSB of all bytes.
                        *GDBUFS expects string_length in X
                        *GDBUFS returns with (A)=0, (Y)=1
                                                  ;
002490: C8                     INY                ;(Y,A) set to $200
002491: 85 0D                  STA    CHARAC      ;No other terminator
002493: 85 0E                  STA    ENDCHR      ;except a null byte.
002495: 20 ED E3               JSR    STRLT2      ;Form temporary string
                                                  ;
                       *STRLT2 expects Y,A to point to
                       *a literal low-ASCII string.  A
                       *temporary string is formed in
                       *memory space that is requested
                       *below FRETOP.  In addition to
                       *the null ($00) terminator, the
                       *values in CHARAC and ENDCHR
                       *are used as string terminators.
                                                  ;
002498: 20 7B DA               JSR    PERMST      ;Make it permanent
                                                  ;
00249B: A9 00           ESCENT LDA    #0          ;<ESC> enters here
00249D: 85 10                  STA    DIMFLG      ;Initialize flags
00249F: 85 11                  STA    VALTYP
0024A1: 85 12                  STA    INTFLG
0024A3: A9 4F                  LDA    #$4F        ;Lo-ASCII 'O'
0024A5: 85 81                  STA    VARNAM
0024A7: A9 41                  LDA    #$41        ;Lo-ASCII 'A'
0024A9: 85 82                  STA    VARNAM+1
0024AB: 20 4F E0               JSR    VARLOC      ;Locate the variable OA
0024AE: 85 85                  STA    FORPNT      ;Aim FORPNT at the
0024B0: 84 86                  STY    FORPNT+1    ;variable value.
0024B2: A0 00                  LDY    #0          ;Default OA to zero
0024B4: 24 43                  BIT    OAFLAG
0024B6: 10 02 =24BA            BPL    FLTOA
0024B8: A4 3C                  LDY    KEYCOD      ;If flag, use KEYCODe
0024BA: 20 01 E3        FLTOA  JSR    SNGFLT      ;Float new OA value
0024BD: 20 27 EB               JSR    STORE       ;Store it in OA
                                                  ;
0024C0: A9 53                  LDA    #$53        ;Lo-ASCII 'S'
0024C2: 85 81                  STA    VARNAM
                                                  ;
                                                  ;(VARNAM+1 still holds 'A')
                                                  ;
0024C4: 20 4F E0               JSR    VARLOC      ;Locate the variable SA
0024C7: 85 85                  STA    FORPNT      ;Aim FORPNT at the
0024C9: 84 86                  STY    FORPNT+1    ;variable value.
0024CB: A0 00                  LDY    #0          ;Default SA to zero
0024CD: 24 44                  BIT    SAFLAG
0024CF: 10 02 =24D3            BPL    FLTSA
0024D1: A4 3C                  LDY    KEYCOD      ;If flag, use KEYCODe
0024D3: 20 01 E3        FLTSA  JSR    SNGFLT      ;Float new SA value
0024D6: 20 27 EB               JSR    STORE       ;Store it in SA
                                                  ;
```

```
0024D9: A9 45              LDA   #$45       ;Lo-ASCII 'E'
0024DB: 85 81              STA   VARNAM
0024DD: A9 53              LDA   #$53       ;Lo-ASCII 'S'
0024DF: 85 82              STA   VARNAM+1
0024E1: 20 4F E0           JSR   VARLOC     ;Locate the variable ES
0024E4: 85 85              STA   FORPNT     ;Aim FORPNT at the
0024E6: 84 86              STY   FORPNT+1   ;variable value.
0024E8: A4 46              LDY   ESCFLG     ;"1" if <ESC>, else "0"
0024EA: 20 01 E3           JSR   SNGFLT     ;Float new ES value
0024ED: 20 27 EB           JSR   STORE      ;Store it in ES
                                            ;
0024F0: A5 3D              LDA   BOTCV      ;Move cursor to
0024F2: 85 25              STA   CV         ;bottom of display
0024F4: 20 22 FC           JSR   VTAB       ;(one character position
0024F7: A5 3E              LDA   BOTCH      ;beyond last char in
0024F9: 85 24              STA   CH         ;string, including
0024FB: 8D 7B 05           STA   CH80       ;trailing spaces),
0024FE: 60                 RTS              ;and exit.
```

The Gentleman's GS: *A Polite Introduction to the 16-bit II*

Part II

by Ross W. Lambert

Last month we eased into a few definitions and a cursory examination of the tool startup order. I finished by suggesting that we'll "revisit" that demonic (for me) piece of code I called Generic Start.

Let me preface that visitation by saying that the GS can be a time bomb. It really pays to learn how to do things right the first time because erroneous code might not produce problems right away (believe me, I know from *experience*, positively embarrassing experience at that, as y'all know). Your program might actually crash in a section of code far removed from the point of the error. Some programs might not crash at all - right away. They save their explosions for an opportune time (opportune being defined as that moment in which a crash will cause the most distressing mischief).

This has always been the case with assembly code (aw heck, it's true in *any* programming environment), but it is particularly pervasive in *my* assembly language GS programs. The reason? I mentioned it briefly last month: the method Apple chose for passing parameters to and from the toolbox is to place them on top of the stack. This is not a bad thing, really, but if you don't watch your pushes and pulls (PHAs and PLAs or `PushWords` and `PullWords`, etc.), you can get them out of balance. Since many of the tool calls require multiple parameters of various sizes, it is easier to screw them up than you might think. If you return from a subroutine with an extraneous parameter squatting astride the stack, for example, your program will try to return to the wrong address. It is more than likely that you will be teleported into oblivion.

That said, we can attack the startup procedure again. Let's take it one step at a time.

A quick stroll down memory lane

First, a fact: the GS memory is organized into 64K banks. Like the main mem and aux mem switching from days of old, you can have a program running in one bank that reads and writes data in another. For the purposes of startup, however, your program will usually want to read data from and write data to the same bank in which it lives.

Unlike the good ol' 8-bit days (?) when you read a softswitch or two, the 65816 CPU has a few new appendages which determine where the processor looks for instructions and data. These new limbs are called the program bank register and the data bank register.

Getting the program bank and the data bank to be one and the same can be accomplished by grabbing the value of the program bank register and pushing it onto the stack. Then, in a not so subtle manipulation, yank the bugger back off the stack and stuff it into the data bank register.

This effectively makes the data bank equal to the program bank. It is a maneuver you'll see often in GS code, and looks like this:

```
Start   phk     ;push program bank register.
        plb     ;pull back into data bank register.
```

You might be wondering why you cannot set the data bank directly, akin to switching between main and auxiliary memory on a IIc or 128K IIe. The reason is that GS programs don't really need to know where they live, at least not very often. The Memory Manager takes care of that. Programs are therefore relocatable and have to set things like data banks indirectly (like the method used above).

An aside - before I started working with the GS (last fall - yes, I *am* new at this, but I think I'm living, breathing proof that a rank beginner can really have good time with the machine), I thought that writing relocatable code for the GS meant jumping through all of the same hoops that it did for the 8 bit Apples. I thought I could never reference labels within my own program, for example. But lo and behold, Apple created a beast called the OMF (Object Module Format). This object code format includes a relocating dictionary which helps the GS (the system loader, actually) relocate your code on its own! Instead of writing your own relocator module or forcing your code to be absolutely and purely relocatable ala' the 8-bit world, the system worries about it for you.

You can write fixed position code for the GS if you really want to since the design team built in all kinds of flexibility into the memory manager. But since relocation worries are pretty much behind us, it is almost pointless.

Notice I said "almost". There are times and instances, I can imagine, wherein carefully crafted, fixed position code could blow the socks off standard OMF performance. But the instances are few and the disadvantages outweigh the advantages for all of the applications I'm inclined to write. (Incidentally and FYI - although I don't reccommend the idea, Micol Systems of Canada has created their own proprietary "fastload" object code format which greatly speeds up the rate at which a program is plopped into memory. There is, as they say, more than one way to skin a cat.)

Back to our subject. The next step in the startup process is to start the Tool Locator. This is *always* the first tool started because it is the bus that all the others ride. We're dead in the water without it, if you'll excuse mixed metaphors.

The code looks like this:

```
_TLStartUp          ;start tool locator
```

Roger is different...

If you own Roger Wagner's *Apple IIGS Assembly Language Programming for Beginners*, you'll notice that the Tool Locator startup looks like this instead:

```
LDX #0201           ;Tool Locator StartUp call number
JSL $E10000         ;tool call entry point
```

This example is taken from p. 321, if you care to look it up. The reason for the apparent discrepancy is that my `_TLStartup` is a macro name. The macro creates Roger's expanded code immediately above this paragraph. Roger discussed creating your own tool macros in the book, the reason being that the text must've been written before the Merlin disk included all of the Tool.Macros macro libraries. I'm certainly glad they are there now!

Needless to say, it is much easier to work with the macro names than to do tool calls "by hand". Remembering the tool call numbers is next to impossible. But now you know that the macros at least include code to load the X register with the tool number and do a long jump (i.e. between 64K banks) to the subroutine that handles toolbox calls.

A tilde for Hilda...

There's yet another class of macros on the recent Merlin disks, these by Dave Klimas (for you APW folks, there is a set of identical macros available from PunkWare, P.O. Box 874043, Wasilla, AK 99687-4073. Send $15 and ask for "PW Macros"). Called tilde macros because they're prescripted with the tilde character (~), they combine all of the "pushes" for parameter passing into one step. We'll look at these in more detail later in this series. Some programmers swear by them, but I think beginners like me need to grow into them. I find myself forgetting whether I'm working with single bytes, words (two bytes), or long words (four bytes). The tilde macros can make debugging a little more complicated for me because I cannot readily see the size of the parameter I pushed on the stack. Once you've got a given tool call down pat, though, you may grow weary of typing all of the `PHA`s, `PushWords` or `PushLongs`. That being the case, you're ready for Dave's macros.

The Tool Locator toolset is a permanent resident of your GS - it's in ROM. In this respect it is different than most of the other toolsets. But we'll get to that next month.

Until then, then.

The Sourceror's Apprentice

Copyright (C) 1989 by Ross W. Lambert and Ariel Publishing

All programs in **The Apprentice** are in the public domain and may be freely copied and distributed, but NOT sold. Apple User Groups and other important folks may reprint articles upon request. Just gimme a call at 509/923-2025 or drop me a line at P.O. Box 398, Pateros,WA 98846.

American prices in US Dollars... 1 year $28, 2 years $52

Canadian and Mexican subscribers add $5 per year, all other non-North American subscribers add $15 per year for first class postage.

Editor and Publisher....... Ross W. Lambert
Technical Editors (and moral support)........ Eric Mueller, Jay Jennings, Robert Moore
Subscription Services........Tamara Lambert, Cindy Eckels
Stamp Licking.......... Rebecca Lambert

Ariel Publishing
Box 398
Pateros, WA 98846

[redacted]