AMPERFORMATTER

APPLE UTILITIES

With AmperFormatter, text formatting on the Apple // becomes nearly automatic!

Like most programmers, I've struggled with the task of formatting text on the Apple screen. Whether you use text in games, application programs or utilities, formatting can make the difference between a well-designed, user friendly program and a poorly designed one.

For instance, words shouldn't break randomly when they reach the edge of the screen. You don't want information to scroll off the screen. If you own an Apple II Plus, your formatting routine should prevent garbage on the screen when lower-case characters are entered.

AmperFormatter does all this and more. This utility is designed for use within Applesoft programs running under DOS 3.3 or ProDOS. It automatically performs word wrap on both 40- and 80-column screens, page breaks and lower-case to upper-case conversion. Plus modifying already-formatted text is simple. AmperFormatter is easy to use and is written in machine language for speedy execution.

USING AMPERFORMATTER

To use AmperFormatter in your own program, you must first insert the following line at the beginning:

```
10 HIMEM:38400:PRINT CHR$(4);
   "BRUN AMPER.FORMAT":HIMEM:
   38144
```

It doesn't have to be line 10, of course, but it should be near the beginning of your program, before you do any printing to the screen. This line sets up the ampersand vector to point to the AmperFormatter routine, and sets HIMEM to prevent AmperFormatter from being overwritten by BASIC. Note that any other programs that occupy high memory will be destroyed by this process.

Before each display of text and after clearing the screen, the line count must be initialized so that AmperFormatter will know how many lines of text to print before doing a page break. This is done by POKEing a zero into location 38394 before calling the AmperFormatter routine.

Now, whenever you want to print something on the screen, instead of using the PRINT command, use an ampersand (&) instead. AmperFormatter takes care of all the formatting problems for you. It automatically detects whether you are using 40 or 80 columns, and takes care of word wrap and page breaks. To print a blank line, print one blank space using:

```
& " "
```

Do not use the ampersand without a string of some kind to print. AmperFormatter expects something after the ampersand, and will generate a SYNTAX ERROR if there is nothing there.

Listing 1 is the AmperFormatter program, and **Listing 2** is a demonstration program that prints a description of AmperFormatter and shows you how to use it.

ENTERING THE PROGRAMS

To enter the programs, first enter the Monitor by typing CALL −151. At the asterisk prompt, type in the hexadecimal code in **Listing 1**, and save it with the command:

BSAVE AMPER.FORMAT,A$94F5, L$106

Or, if you prefer, you may type the listing into your assembler and assemble it. Now type in **Listing 2** and save it with the command:

SAVE AMPER.DEMO

If your Apple does not support lower-case, just use all capitals. Note that the Key Perfect table will not match.

For help with entering *Nibble* programs, see "A Welcome to New *Nibble* Readers" at the beginning of this issue.

HOW THE PROGRAMS WORK

AMPER.DEMO (**Listing 2**) prints text using AmperFormatter and the & symbol.

Some mention might be made of the two PEEKs used: the value PEEKed **in lines 390-400** (PEEK (49183)) is greater than 128 if the 80-column card is turned on. PEEK (−1101) returns a value of 6 when the program is run on an Apple //e or //c (**lines 470-480**).

In the actual AmperFormatter program (**Listing 1**), **lines 37-41** set up the ampersand vector and exit. Once that is done, this part of the program is no longer needed, and can be overwritten by Applesoft. **Lines 47-48** put the address of the string following the ampersand in the X and Y Registers, and put the length in the Accumulator. The beginning and end pointers for the string are saved in PTR and PTREND, respectively, in **lines 49-59**.

AmperFormatter is easy to use and is written in machine language for speedy execution.

Lines 63-66 change the last character of the string, which is normally in positive ASCII, to negative. This way, AmperFormatter will know when it has reached the last character of the string.

Lines 70-74 set a flag indicating whether or not an Apple //e or //c is being used. A $00 signifies that a //e is not being used. This information determines whether or not to print lower-case text.

PRINTMSG (**lines 80-83**) sets the word length (COUNT) to zero and determines the present location in the string. **Lines 85-94** count the number of letters in the word. The end of the word is signaled by a space ($20) or the end of the string (**line 87**). The length of the word is stored in COUNT.

Lines 96-105 start a new line if the length of the word plus the current cursor position

LISTING 1: AMPER.FORMAT

```
                     1
                     2    *******************************
                     3    *                             *
                     4    *         AMPER.FORMAT        *
                     5    *       by Howard Huang       *
                     6    *      Copyright (c) 1986     *
                     7    *      By MicroSPARC, Inc     *
                     8    *      Concord, MA  01742     *
                     9    *                             *
                    10    *      Assembler:  Merlin     *
                    11    *                             *
                    12    *******************************
                    13
                    14              ORG   $94F5
                    15
                    16    COUNT     =     $4            ;Letters in word
                    17    MACHID    =     $5            ;Machine ID: //e or not?
                    18    PTR       =     $6            ;Pointer to string
                    19    PTREND    =     $8            ;End of string
                    20    WNDWDTH   =     $21           ;Width of text window
                    21    CH        =     $24           ;Cursor horizontal
                    22    CHRGET    =     $B1           ;Get next character
                    23    CHRGOT    =     $B7           ;Get last character
                    24    AMPERV    =     $3F5          ;Ampersand vector
                    25    OURCH     =     $57B          ;Cursor on 80-cols
                    26    RD80VID   =     $C01F         ;80-cols on or not
                    27    FRMEVL    =     $DD7B         ;Evaluate a formula
                    28    FREFAC    =     $E600         ;Get address of string
                    29    PRBL2     =     $F94A         ;Print spaces
                    30    VERSION   =     $FBB3         ;F8VERSION
                    31    RDKEY     =     $FD0C         ;Get a keypress
                    32    CROUT     =     $FD8E         ;Print carriage return
                    33    COUT      =     $FDED         ;Print a character
                    34    RESTORE   =     $FF3F         ;Recover registers
                    35    SAVE      =     $FF4A         ;Save registers
                    36
94F5: A9 95         37    SETAMPV   LDA   #>START
94F7: 8D F7 03      38              STA   AMPERV+2
94FA: A9 00         39              LDA   #START
94FC: 8D F6 03      40              STA   AMPERV+1      ;Set ampersand vector
94FF: 60            41              RTS                 ;Ready for use!
                    42
                    43    *======================
                    44    * PRINTMSG Main Routine
                    45    *======================
                    46
9500: 20 7B DD      47    START     JSR   FRMEVL        ;Evaluate formula
9503: 20 00 E6      48              JSR   FREFAC        ;Find string in memory
9506: 84 07         49              STY   PTR+1
9508: 86 06         50              STX   PTR           ;Address of string
950A: AA            51              TAX
950B: CA            52              DEX
950C: 8A            53              TXA                 ;Decrease ACC
950D: 18            54              CLC
950E: 65 06         55              ADC   PTR           ;Add length of string
9510: 85 08         56              STA   PTREND        ;to get location of end
9512: 98            57              TYA
9513: 69 00         58              ADC   #0            ;Get high byte + carry
9515: 85 09         59              STA   PTREND+1
                    60
                    61    * Negate last byte of string
                    62
9517: A0 00         63              LDY   #0
9519: B1 08         64              LDA   (PTREND),Y    ;Get last byte
951B: 09 80         65              ORA   #$80          ;Change to negative
951D: 91 08         66              STA   (PTREND),Y
                    67
                    68    * Check for //e
                    69
951F: AD B3 FB      70              LDA   VERSION
9522: C9 06         71              CMP   #6            ;//e signature byte?
9524: D0 01         72              BNE   NOTIIE
9526: 88            73              DEY                 ;Minus = //e
9527: 84 05         74    NOTIIE    STY   MACHID
                    75
                    76    *
```

exceeds the screen width. Note that since the width of the screen is used instead of 40 or 80 for this calculation, you can set the window width to anything you want, and the text will still print properly.

Lines 109-111 place the pointer on the correct page of memory. **Lines 113-134** print the text. Any carriage returns are handled by CARRIAGE instead of the normal CROUT. **Lines 122-126** convert lower-case to upper-case, if necessary.

When a space is found (**lines 129-130**) or the end of the string is reached (**line 115**), execution goes to DIDWORD (**lines 138-142**), which increments the pointer and goes back for another word.

AmperFormatter takes care of all the formatting problems for you.

When the end of the string is found, a check is made for a semicolon. If there is no semicolon, a carriage return is printed; otherwise, flow passes to RESET (**lines 153-159**), which sets the last character of the string back to positive ASCII.

The CARRIAGE subroutine (**lines 165-195**) prints a carriage return via CROUT at $FD8E and then checks the line counter to see if the bottom of the screen was reached. If it was not, flow passes to EXITCR and returns to the main program. Otherwise:

1. The counter is reset to zero (**lines 171-172**).
2. The message (CONTINUED) is printed (**lines 174-179**).
3. A keypress is fetched (**line 181**).
4. The message is erased (**lines 182-189**).
5. Flow returns to the main program (**lines 191-192**).

CUSTOMIZATION

Without the disposable initialization section, AmperFormatter occupies just under a page of memory, so making any change will probably require that you change the program origin (**line 14**). This shouldn't change the BASIC interface, except for the HIMEM command.

If you print only one- or two-line strings, you won't need the line counting routine used for the page-break feature. In addition, you could remove **lines 117-120 and 165-195**, and change all of the JSR CARRIAGE commands to JSR CROUT, to save some memory.

LISTING 1: AMPER.FORMAT *(continued)*

```
                  77   * Print the string
                  78   *
                  79
9529: AC F9 95    80   PRINTMSG LDY   INDEX       ;Position in string
952C: A9 00       81            LDA   #0
952E: AA          82            TAX
952F: 85 04       83            STA   COUNT       ;Letters in word
                  84
9531: E6 04       85   WRAP     INC   COUNT
9533: B1 06       86            LDA   (PTR),Y     ;Get byte
9535: 30 0C       87            BMI   ADD         ;End of string
9537: C9 20       88            CMP   #$20
9539: F0 08       89            BEQ   ADD         ;End of word
953B: C8          90   NEXTLET  INY
953C: D0 F3       91            BNE   WRAP
953E: E6 07       92            INC   PTR+1       ;Next page
9540: E8          93            INX               ;Flag next page
9541: 10 EE       94            BPL   WRAP
                  95
9543: 18          96   ADD      CLC
9544: A5 04       97            LDA   COUNT       ;Letters in word
9546: 2C 1F C0    98            BIT   RD80VID     ;80-columns?
9549: 10 04       99            BPL   FORTYCOL
954B: 6D 7B 05    100           ADC   OURCH       ;Add letters to cursor position
954E: 2C          101           HEX   2C
954F: 65 24       102  FORTYCOL ADC   CH          ;Add length to current position
9551: C5 21       103           CMP   WNDWDTH     ;Is it over screen width?
9553: 90 03       104           BLT   PRINTWD     ;Doesn't go past screen
9555: 20 B1 95    105           JSR   CARRIAGE    ;Print <return>
                  106
                  107  * Print the word
                  108
9558: CA          109  PRINTWD  DEX
9559: 30 02       110           BMI   PRINT       ;Still on same page
955B: C6 07       111           DEC   PTR+1       ;Restore correct page
                  112
955D: AC F9 95    113  PRINT    LDY   INDEX
9560: B1 06       114  PRTWORD  LDA   (PTR),Y
9562: 30 2F       115           BMI   END         ;Last char in string
9564: 09 80       116           ORA   #$80
9566: C9 8D       117           CMP   #$8D
9568: D0 06       118           BNE   LOWERC      ;Not a return
956A: 20 B1 95    119           JSR   CARRIAGE    ;Substitute for CROUT
956D: 4C 81 95    120           JMP   NEXTCHAR
                  121
9570: 24 05       122  LOWERC   BIT   MACHID
9572: 30 06       123           BMI   PRTCHAR     ;//e -- print lowercase
9574: C9 E0       124           CMP   #$E0
9576: 90 02       125           BLT   PRTCHAR     ;Not lowercase
9578: 29 DF       126           AND   #$DF        ;Convert
                  127
957A: 20 ED FD    128  PRTCHAR  JSR   COUT
957D: C9 A0       129           CMP   #$A0
957F: F0 07       130           BEQ   DIDWORD     ;Done with a word
9581: C8          131  NEXTCHAR INY
9582: D0 DC       132           BNE   PRTWORD
9584: E6 07       133           INC   PTR+1
9586: D0 D8       134           BNE   PRTWORD     ;Finish the word
                  135
                  136  * Prepare for another word
                  137
9588: C8          138  DIDWORD  INY
9589: D0 02       139           BNE   DONE
958B: E6 07       140           INC   PTR+1       ;Point to next word
958D: 8C F9 95    141  DONE     STY   INDEX       ;Save position
9590: 4C 29 95    142           JMP   PRINTMSG
                  143
                  144  * End of the message
                  145
9593: 20 ED FD    146  END      JSR   COUT
9596: 20 B7 00    147           JSR   CHRGOT      ;Get this character
9599: C9 3B       148           CMP   #$3B
959B: F0 05       149           BEQ   RESET       ;Semicolon
959D: 20 B1 95    150           JSR   CARRIAGE    ;Print <return> if not ";"
95A0: D0 03       151           BNE   NEGATIVE
                  152
95A2: 20 B1 00    153  RESET    JSR   CHRGET      ;Point to next char
95A5: A0 00       154  NEGATIVE LDY   #0
95A7: B1 08       155           LDA   (PTREND),Y
95A9: 29 7F       156           AND   #$7F        ;Reset last byte of string
95AB: 91 08       157           STA   (PTREND),Y
95AD: 8C F9 95    158           STY   INDEX       ;Reset position
95B0: 60          159           RTS
                  160
                  161  *
                  162  * CARRIAGE: Print carriage return
                  163  *
                  164
95B1: 20 4A FF    165  CARRIAGE JSR   SAVE        ;Save registers
95B4: 20 8E FD    166           JSR   CROUT
95B7: EE FA 95    167  CHECK    INC   BOTTOM      ;count the line
95BA: AD FA 95    168           LDA   BOTTOM
95BD: C9 17       169           CMP   #$17        ;At bottom of screen?
95BF: 90 28       170           BLT   EXITCR
```

```
95C1: A9 00             LDA  #0
95C3: 8D FA 95          STA  BOTTOM      ;Reset counter

95C6: A0 00             LDY  #0
95C8: B9 ED 95  CONT    LDA  CONTINUE,Y
95CB: F0 06             BEQ  WAIT
95CD: 20 ED FD          JSR  COUT        ;Print "(CONTINUED)"
95D0: C8                INY
95D1: D0 F5             BNE  CONT

95D3: 20 0C FD  WAIT    JSR  RDKEY       ;Get a keypress
95D6: A9 00             LDA  #0
95D8: 85 24             STA  CH
95DA: 8D 7B 05          STA  OURCH       ;Put cursor on column 0
95DD: A2 0B             LDX  #11
95DF: 20 4A F9          JSR  PRBL2       ;Erase "(CONTINUED)"
95E2: A9 00             LDA  #0
95E4: 85 24             STA  CH
95E6: 8D 7B 05          STA  OURCH       ;Set column again

95E9: 20 3F FF  EXITCR  JSR  RESTORE     ;Restore registers
95EC: 60                RTS

95ED: A8 C3 CF  CONTINUE ASC "(CONTINUED)"
95F0: CE D4 C9 CE D5 C5 C4 A9
95F8: 00                HEX  00
95F9: 00        INDEX   HEX  00
95FA: 00        BOTTOM  HEX  00

--End assembly, 262 bytes, Errors: 0
```

END OF LISTING 1

```
              KEY PERFECT 5.0
                   RUN ON
                AMPER.FORMAT
 ==========================================
   CODE-5.0    ADDR# - ADDR#     CODE-4.0
   --------    -------------     --------
   D74B875E    94F5 - 9544         2669
   CAC3D707    9545 - 9594         28AC
   7EA16EDF    9595 - 95E4         2A1B
   BD28DF82    95E5 - 95FA         096C
   A5327904 = PROGRAM TOTAL =      0106
```

LISTING 2: AMPER.DEMO

```
10  REM ********************
20  REM * AMPER.DEMO       *
30  REM * BY: HOWARD HUANG *
40  REM * COPYRIGHT (C) 1986 *
50  REM * BY MICROSPARC, INC *
60  REM * CONCORD, MA  01742 *
70  REM ********************
80  HIMEM: 38400: PRINT : PRINT  CHR$ (4)"BRU
     NAMPER.FORMAT"
90  HIMEM: 38144        (PRODOS use HIMEM:37120)
100 RET$ = " ":COUNT = 38394
110  REM
120  REM === TITLE PAGE ===
130  REM
140 WIDE =  PEEK (33) / 2 + 1
150  TEXT : HOME : NORMAL : VTAB 10: HTAB WID
     E - 9: & "Amper-Format Demo": PRINT : HTAB
     WIDE - 8: & "By Howard Huang": PRINT : HTAB
     WIDE - 20: & "* Copyright 1986 by MicroS
     PARC, Inc. *"
160  GOSUB 820
170  REM
180  REM === SELF-PRAISE ===
190  REM
200  HOME : & "Welcome to Amper Formatter!"
210  & RET$: & "   Amper Formatter is a machi
     ne language utility designed to format t
     ext strings for screen display."
220  & "   The features include:": & RET$: &
```

LISTING 2: AMPER.DEMO *(continued)*

```
     "1) Automatic word wrapping": & "2) Dete
     ction of 40 or 80 column screen"
230  & "3) Conversion of lower to upper case"
     : & "4) Page breaks on the screen"
240  & "5) DOS 3.3 - ProDOS compatibility": &
     RET$: & "   This is a short demonstratio
     n that highlights these features and sho
     ws you how to use Amper Formatter from w
     ithin your own programs."
250  GOSUB 820
260  REM
270  REM === WORD WRAP ===
280  REM
290  & "   First, let's look at word wrapping
     . Normally, when the Apple reaches the e
     dge of the text screen, it automatically
      goes down to the next line. This someti
     mes causes";
300  & " words to be split in the middle, res
     ulting in unreadability and an unprofess
     ional appearance.": & RET$
310  & "   With Amper Formatter, you may use
     messages of any length in your programs
     without worrying about word breaks; Ampe
     r Formatter will break them for you!": &
     RET$
320  GOSUB 820
330  & "   If you list this program, you'll f
     ind that all the text is unformatted, an
     d that the lines are being properly form
     atted and printed by Amper Formatter, wi
     th carriage returns in the proper places
     ."
340  & RET$: & "    Just type your messages o
     nce and they are automatically formatted
      for the size of the screen.": GOSUB 820
350  REM
360  REM === 40/80 COLUMNS ===
370  REM
380  & "   Another nice feature is the abilit
     y to detect whether or not the 80-column
      card is active. If the card is active,
     Amper Formatter will take advantage of i
     t and break the words accordingly.": & R
     ET$
390  & "   You might want to try running this
      program in ";: IF  PEEK (49183) > 128 THEN
      & "40";
400  IF  PEEK (49183) < 128 THEN  & "80";
410  & " column mode to see the difference.":
     GOSUB 820
420  REM
430  REM === LOWERCASE ===
440  REM
450  & "   If you have an Apple II or II+, an
     y lowercase text displayed by Amper Form
     atter will be converted automatically to
      uppercase. This way, Apple //e and //c
     owners can write programs that are compa
     tible with all Apple II's while ";
460  & "still taking advantage of the //e and
      //c's new features.": & RET$
470  & "   To see the difference, try using t
     his program on an Apple ";: IF  PEEK ( -
     1101) = 6 THEN  & "II or II+";
480  IF  PEEK ( - 1101) <  > 6 THEN  & "//e o
     r //c";
490  & ".": GOSUB 820
500  REM
510  REM === PAGE BREAKS ===
520  REM
530  & "   Amper Formatter will also handle t
     ext displays longer than one screen long
     .  Amper Formatter will count the number
      of lines printed, and when 23 lines hav
     e been printed, ";
540  & "the word (CONTINUED) is printed at th
     e bottom of the screen, and the program
     waits until the user presses a key.": &
     RET$: & "Here's an example:": GOSUB 820
550  FOR I = 1 TO 25:A$ =  STR$ (I) + " Nibbl
     e Magazine": & A$: NEXT I
560  & RET$: & "   When you are finished prin
     ting text, you should set the line count
     er back to 0 with POKE 38394,0.": & RET$
     : GOSUB 820
570  REM
580  REM === DOS - ProDOS ===
590  REM
600  & "   Amper Formatter is compatible with
      both DOS 3.3 and ProDOS.": & RET$
610  & "   All of these features, plus machin
     e language speed, make Amper Formatter f
     ast and easy to use.": & RET$
620  & "   Now, let's briefly look at how to
     use Amper Formatter.": GOSUB 820
630  REM
640  REM === USING ===
650  REM
660  & "   It's easy to set up Amper Formatte
     r. BRUN the program and set HIMEM: 38144
     .": & RET$
670  & "   To actually use the formatter is j
     ust as simple. Wherever you want to prin
     t something, just use an ampersand [&] i
     nstead of PRINT. Here are some examples:
     ": & RET$
680 A$ = "& " +  CHR$ (34) + "Hello, how are
     you today?" +  CHR$ (34): & A$
690  & "& A$": & "& CHR$(160)": & "& C$(I)": &
     "& A$:": & RET$
700  GOSUB 820
710  & "   The only restriction is with print
     ing blank lines. In BASIC, this is done
     with a simple PRINT statement, with no s
     tring specified. With Amper Formatter, y
     ou must supply a string of some kind, ev
     en if it's just one space. "
720  GOSUB 820
730  REM
740  REM === WRAP IT UP ===
750  REM
760  & "   Amper Formatter will be a super ad
     dition to your utility library. Use it w
     ell!": & RET$
770  HTAB WIDE - 2: & "End.": POKE COUNT,0
780  END
790  REM
800  REM === KEYPRESS ===
810  REM
820  VTAB 23: HTAB WIDE - 17: & "PRESS <RETUR
     N> TO CONTINUE"
830  WAIT  - 16384,128: POKE  - 16368,0: POKE
     COUNT,0
840  HOME : RETURN
```

END OF LISTING 2

```
             KEY PERFECT 5.0
                 RUN ON
               AMPER.DEMO
==========================================
   CODE-5.0     LINE# - LINE#    CODE-4.0
   --------     -------------    --------
   48C82D77       10 -   100       6679
   A52146F1      110 -   200       6B0B
   C1AA3157      210 -   300     014D3B
   BA90501E      310 -   400     015001
   AE9B4554      410 -   500       D054
   17EBF828      510 -   600     010BFA
   D05BF56F      610 -   700       F231
   E49BFD29      710 -   800       AF69
   1BAD2402      810 -   840       2388
   9112A29C = PROGRAM TOTAL =      0EDB
```