



# THINK IT THROUGH

Define your task, break it down into a logical series of steps—and you're halfway there. If you know how to use structured BASIC to implement a program, you can achieve professional results quickly.

By JOE ABERNATHY

THE KEY TO PROGRAMMING, LIKE everything else, is to begin with a good idea. But how do you create that spark? And how do you turn it into software?

This month I'll illustrate the complete program-design process and offer a library of source-code procedures with which you can manage file printing and display tasks. You'll turn these procedures into a classic desk accessory (CDA) and learn an easy method of writing programs that you can use with any IIGS language.

## DEFINE THE PROBLEM

Think of your favorite utility and ask yourself how it originated. Chances are you're convinced you could write a more efficient program for your IIGS. *That's* where the best software always originates.

The first step in writing a program is to define a problem or a task. How do you work with your Apple, and what do you find annoying about the process? Is there something your Apple could do better? Therein lies your program's definition.

And that's how I got the idea to create ShowCDA (**Listing 1**). While using my GS, I booted Cat Doctor (a catalog utility that comes with ProSel drive-management software) to find and read a text file I knew was somewhere in my library. I grew annoyed at the endless

steps involved: exit current application; launch utility; find and read file; exit; relaunch.

What I needed was something that wouldn't force me to exit my current application to consult my notes. A CDA was the perfect solution. Its capabilities would have to include display to screen, print, catalog, and set prefix. A "print with header" option would be useful for printing archival source code.

Choosing the compiler was simple—Micol Advanced BASIC, the only one that generates CDA files. And since CDAs don't use QuickDraw Toolbox screen output, the user interface must be the 80-column text screen. I set out to design a custom interface in the fashion of a CDA.

With this much decided, I had yet to write a line of code. Because I had clearly defined the problem, though, the program was already half finished. I let structured programming style write the application for me, breaking down the big picture into pieces small enough that they fell into order.

## SOLVE THE PROBLEM

Your outline should translate directly into source code. Each of the program's features, including the CDA user interface, should end up as a portable procedure or a group of related procedures in the final listing. Look ahead to the source code and you'll see this is so: The procedures FileType and DoTypeFile handle output to screen; FilePrint and Do-

PrintFile, printing; and HeaderPrint and DoHPrintFile, print with headers. SetPrefix and ShowCatalog speak for themselves. Note that the procedures that do the work remain separate from their program-specific calling routines. Because of this design, we can reuse the procedures in the desktop environment. (See **Listings 2** and **3**.)

Something has to bring these procedures alive. In our CDA, RouteMain, DoCDAScreen, and MainScreen give us access to the CDA's utilities. The short REPEAT/UNTIL structure at the end of the program ties them together.

You can quickly modify this group of procedures to create other CDAs. To do so, update the program-specific screen-menu strings, and the *INDEX menucommands\$* filter string. Note that you also must take along the global variables specified at the top of the program.

The last thing you must do with ShowCDA is create text files for the on-line help function (**Listings 4-9**). Create a directory called \*/SYSTEM/DISCGOV.DATA/ in which to place these files.

That's it—a fast, direct path to completing a program. Compile ShowCDA as listed to create a link file, then use Micol's EXE.TO.CDA utility to convert the link file to a classic desk accessory. Place the file into your System Disk's SYSTEM/DESK.ACCS folder, and you're done. (You could just as easily compile this as a launchable .SYS16 file, rather than a CDA, but that would defeat the purpose of this particular program.)

## FINISH IT RIGHT

Simply finish a program—any program, at any level of quality—and you've beaten perhaps 80 percent of your competition. But if you finish your program right, you have something worth money.

After I'm satisfied that a program is complete, I ask a friend (preferably one who dislikes computers) to give it a try. This resulted, ►

## Listing 1. ShowCDA.

```

PROGRAM ShowCDA

{ ----- }
{ ShowCDA                                Version }
{ By Joe Abernathy                        1.0 }
{ (C)1989, inCider.                      ===== }
{ All Rights Reserved. }
{ Compiler: Micol Advanced BASIC }
{ ----- }
{ This application is designed to be }
{ compiled as a Classic Desk Accessory. }
{ Its purpose is to provide a way of }
{ finding, reading and printing text }
{ files. The work procedures are suited }
{ to inclusion in your source code }
{ library. }
{ ----- }
{ Set up: }

FileNum% = 1 { file index }
Done! = FALSE { quit flag }
Terror! = FALSE { task error }
Herror! = FALSE { help error }
InputErr! = FALSE { shared flag }
title$ = " ShowFile CDA "
author$ = " By Joe Abernathy "
company$ = " (C)1989, inCider "

{ ----- }
{ PROC DoPrintFile }
PROC DoPrintFile [WhichF$]
IF FILE (WhichF$) THEN BEGIN
  PRON { turn on printer }
  ROPE (8) WhichF$ { buffered read }
  Counter% = 1
  REPEAT
    INPUT (8) a$
    PRINT a$
    UNTIL EOF(8)
    Counter% = Counter% + 1
    IF Counter% = 64 THEN Counter% = 1
  CLOSE (8)
  IF Counter% < 64 THEN BEGIN
    x% = 64 - Counter%
    FOR i% = 1 TO x%
      PRINT
    NEXT i%
  ENDIF
  SCRNON { printer off }
ENDIF
ENDPROC { DoPrintFile }

{ ----- }
{ PROC DoHPrintFile }
{ .. print with header }

PROC DoHPrintFile [WhichF$]
IF FILE (WhichF$) THEN BEGIN
  PRON { turn on printer }
  ROPE (8) WhichF$
  counter% = 1
  tmp$ = "File: " + WhichF$
  tmp% = LEN(tmp$)
  tmp1% = LEN("Page Number: XX")
  tmp% = (80 - (tmp% + tmp1%))
  REPEAT
    PRINT "File: "; WhichF$;
    PRINT SPC(tmp%);
    x$ = "Page Number: " + (STR$(counter%))
    PRINT x$
    PRINT
    FOR i% = 1 TO 61 UNTIL EOF(8)
      INPUT (8) a$
      PRINT a$
      NEXT i%
    counter% = counter% + 1
    IF i% < 61 THEN BEGIN
      t% = 61 - i%
      FOR x% = 1 TO t%
        PRINT { compensate for ... }
        NEXT x% { ..file's last page }
      ENDIF
    ENDIF
  ENDPROC { DoHPrintFile }

```

```

ENDIF
PRINT SPC(24);
PRINT "Date: "; DATE$; " "; "Time: "; TIME$
PRINT
UNTIL EOF(8)
CLOSE (8)
SCRNON { printer off }
ENDIF
ENDPROC { DoHPrintFile }

{ ----- }
{ PROC DoTypeFile }
PROC DoTypeFile [Which$]
IF FILE (Which$) THEN BEGIN
  ROPE (8) Which$
  counter% = 1
  REPEAT
    HOME
    VTAB (24)
    x$ = "< Press RETURN to continue >"
    x% = (80 - LEN(x$)) / 2
    HTAB (x%)
    PRINT x$
    HTAB (1)
    VTAB (1)
    PRINT "Type File: "; Which$;
    x$ = "Page Number: " + (STR$(counter%))
    x% = (80 - LEN(x$))
    HTAB (x%)
    PRINT x$
    FOR i% = 1 TO 80
      PRINT " ";
      NEXT i%
    PRINT
    VTAB (4)
    HTAB (1)
    FOR i% = 1 TO 18 UNTIL EOF(8)
      INPUT (8) a$
      PRINT a$
      NEXT i%
    counter% = counter% + 1
    GET a$
    HOME
    UNTIL EOF(8)
  CLOSE (8)
ENDIF
ENDPROC { DoTypeFile }

{ ----- }
{ Do Help -- Show help files }
PROC DoHelp[hfile$]
a$ = hfile$
{ list of allowed keystrokes: }
menucmnds$ = "HQSCWPT"
occurrence% = INDEX (a$, menucmnds$)
IF occurrence% = 0 THEN Herror! = TRUE
a$ = "*/SYSTEM/DISGOV.DATA/" + a$
IF FILE (a$) THEN Herror! = FALSE
IF NOT Herror! THEN BEGIN
  GOSUB DoCDAScreen[title$, author$, company$]
  ROPE (8) a$
  VTAB (6)
  HTAB (1)
  WHILE NOT EOF(8)
    INPUT (8) a$
    PRINT a$
    WEND
  CLOSE (8)
  ENDIF
  HTAB (1)
  VTAB (1)
  GET a$
  Herror! = FALSE
ENDPROC { DoHelp }

{ ----- }
{ Set Prefix }

PROC SetPrefix
HTAB (1)
VTAB (6)
PRINT " _ Set GS/OS Prefix _ "

```

Continued

Continued

```

VTAB (8)
INPUT "Enter Pathname -> ";x$
IF x$ = "" THEN Terror! = TRUE
GOSUB CheckInput[x$]
IF InputErr! THEN Terror! = TRUE
IF NOT Terror! THEN BEGIN
    PREFIX x$
    ENDIF
    InputErr! = FALSE
ENDPROC { SetPrefix }
{ ----- }
{ Show Catalog of current volume }
PROC ShowCatalog
    HOME
    PRINT "Catalog: CTRL-S = Stop/Start"
    CATALOG ""
    PRINT "Press a key to continue: ";
    GET a$
ENDPROC { ShowCatalog }
{ ----- }
{ Print with header }
PROC HeaderPrint
    HTAB (1)
    VTAB (6)
    PRINT "    Print With Info Header    "
    VTAB (8)
    INPUT "Enter Pathname -> ";x$
    IF x$ = "" THEN Terror! = TRUE
    GOSUB CheckInput[x$]
    IF InputErr! THEN Terror! = TRUE
    IF NOT Terror! THEN BEGIN
        IF FILE (x$) THEN GOSUB DoHPrintFile [x$]
        HOME
        GOSUB MainScreen
        ENDIF
        InputErr! = FALSE
    ENDPROC { HeaderPrint }
{ ----- }
{ Print file }
PROC FilePrint
    HTAB (1)
    VTAB (6)
    PRINT "    Print File    "
    VTAB (8)
    INPUT "Enter Pathname -> ";x$
    IF x$ = "" THEN Terror! = TRUE
    GOSUB CheckInput[x$]
    IF InputErr! THEN Terror! = TRUE
    IF NOT Terror! THEN BEGIN
        IF FILE (x$) THEN GOSUB DoPrintFile [x$]
        HOME
        GOSUB DoCDAScreen[title$,author$,company$]
        ENDIF
        InputErr! = FALSE
    ENDPROC { FilePrint }
{ ----- }
{ File Typer }
PROC FileType
    HTAB (1)
    VTAB (6)
    PRINT "    Type a File    "
    VTAB (8)
    INPUT "Enter Pathname -> ";x$
    IF x$ = "" THEN Terror! = TRUE
    GOSUB CheckInput[x$]
    IF InputErr! THEN Terror! = TRUE
    IF NOT Terror! THEN BEGIN
        IF FILE (x$) THEN BEGIN
            HOME
            GOSUB DoTypeFile [x$]
            ENDIF
            InputErr! = FALSE
        ENDPROC { FileType }
{ ----- }
{ Online Help Function }
PROC BringHelp
    GOSUB MainScreen

```

```

x$ = "< Choose Help Key, or Press Q to quit >"
x% = (80-LEN(x$))/2
VTAB (24)
HTAB (x%)
PRINT x$;
HTAB (1)
VTAB (1)
GET a$
IF a$ = "" THEN Terror! = TRUE
IF a$ = " " THEN Terror! = TRUE
a$ = UPPER$ (a$)
IF a$ = "Q" THEN Terror! = TRUE
IF NOT Terror! THEN BEGIN
    GOSUB DoHelp[a$]
    ENDIF
ENDPROC { BringHelp }
{ ----- }
{ Work routine for Main Menu }
PROC RouteMain[actn$]
    a$ = actn$
    menucmnds$ = "HQSCWPT"
    occurrence% = INDEX (a$,menucmnds$)
    IF occurrence% <> 0 THEN BEGIN
        HOME
        IF a$ = "Q" THEN BEGIN { Quit }
            Done! = TRUE
        ELSE IF a$ = "H" THEN BEGIN { Help }
            GOSUB DoHelp["H"]
            REPEAT
                GOSUB BringHelp
            UNTIL Terror!
            Terror! = FALSE
        ELSE IF a$ = "S" THEN BEGIN { Set Prefix }
            GOSUB DoCDAScreen[title$,author$,company$]
            GOSUB SetPrefix
            Terror! = FALSE
        ELSE IF a$ = "C" THEN BEGIN { Show Catalog }
            GOSUB ShowCatalog
        ELSE IF a$ = "W" THEN BEGIN { Print w/header }
            REPEAT
                GOSUB DoCDAScreen[title$,author$,company$]
                GOSUB HeaderPrint
            UNTIL Terror!
            Terror! = FALSE
        ELSE IF a$ = "P" THEN BEGIN { Print file }
            REPEAT
                GOSUB DoCDAScreen[title$,author$,company$]
                GOSUB FilePrint
            UNTIL Terror!
            Terror! = FALSE
        ELSE IF a$ = "T" THEN BEGIN { Type File }
            REPEAT
                GOSUB DoCDAScreen[title$,author$,company$]
                GOSUB FileType
            UNTIL Terror!
            Terror! = FALSE
        ENDIF { keypress }
    ENDIF
ENDPROC { RouteMain }
{ ----- }
{ Draw shared background for a CDA }
PROC DoCDAScreen[titl$,authr$,publshr$]
    HOME
    x% = (80-LEN(titl$))/2
    VTAB (1)
    FOR i% = 1 TO 80
        PRINT "-";
    NEXT i%
    HTAB (x%)
    VTAB (1)
    PRINT titl$
    HTAB (1)
    VTAB (3)
    FOR i% = 1 TO 80
        PRINT "-";
    NEXT i%
    HTAB (1)
    VTAB (3)

```

Continued

Continued

```

PRINT authr$
x% = 80-LEN(publshr$)
HTAB (x%)
VTAB (3)
PRINT publshr$
HTAB (1)
VTAB (22)
FOR i% = 1 TO 80
  PRINT "-";
NEXT i%
VTAB (24)
x$ = "< Press <RETURN> for Main Menu >"
x% = (80-LEN(x$))/2
HTAB (x%)
PRINT x$;
ENDPROC { DoCDAScreen }
{ ----- }
{ Build Menu Screen }
PROC MainScreen
  GOSUB DoCDAScreen[title$,author$,company$]
  x$ = "
  x% = (80-LEN(x$))/2
  VTAB (24)
  HTAB (x%)
  PRINT x$;
  VTAB (8)
  HTAB (10)
  PRINT "T -> Type a File";
  HTAB (45)
  PRINT "P -> Print a File"
  VTAB (9)
  HTAB (10)
  PRINT "W -> Print File With Header";
  HTAB (45)
  PRINT "C -> Catalog Current Volume"
  VTAB (10)
  HTAB (10)
  PRINT "S -> Set GS/OS Prefix";
  HTAB (45)
  PRINT "H -> Help"
  VTAB (11)
  HTAB (45)
  PRINT "Q -> Quit"
ENDPROC { MainScreen }
{ ----- }
{ CheckInput }
{ check for errors in input string }
PROC CheckInput[chkstr$]
  a$ = chkstr$
  a$ = UPPER$(a$)
  i% = LEN(a$)
  FOR x% = 1 TO i%
    y$ = MID$(a$,x%,1)
    legal$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ./"
    myindex% = INDEX(y$,legal$)
    IF myindex% = 0 THEN InputErr! = TRUE
  NEXT x%
  InputErr! = FALSE
ENDPROC { CheckInput }
{ ----- }
{ BEGIN ShowFile CDA }
{ This is where main in-line execution }
{ occurs: }
REPEAT
  GOSUB MainScreen
  HTAB (1)
  VTAB (1)
  GET a$
  a$ = UPPER$(a$)
  GOSUB RouteMain [a$]
UNTIL Done! { quit flag }
END { ShowFile CDA }
{ ----- }

```

## Listing 2. inCider.Shell v. 1.1.

```

File: inCider.Shell Version 1.1 2/26/89
By Joe Abernathy
(C)1989, inCider. All Rights Reserved.
Compiler: AC/BASIC for the Apple IIgs.
Compile this shell with the "no standard menus"
option selected. V1.1 inCider.Tools required.

Portions of this program include material copyrighted
(C) by Absoft Corp. 1988. Used with permission. All
other copyrights acknowledged.

This listing contains only the version 1.1 additions
to the inCider Shell. You can obtain the full shell
by sending SASE, or download from inCider BBS.

menuproc v1.1 2-23-89
Note that labels 50, 60, and 70 refer different
procedures in version 1.0. Just relabel them and
add the new procedure 50.

menuproc:
  menunum = MENU(0)      ' Interpret menu events
  itemnum = MENU(1)      ' Read which menu
  IF menunum = 1 THEN    ' Read which item
    IF itemnum = 1 THEN  ' .. FILE menu
      GOSUB 10           ' New
    ELSEIF itemnum = 2 THEN ' Edit
      GOSUB 20
    ELSEIF itemnum = 3 THEN ' Delete
      GOSUB 30
    ELSEIF itemnum = 4 THEN ' Print
      GOSUB 40
    ELSEIF itemnum = 5 THEN ' Print file w/header
      GOSUB 50
    ELSEIF itemnum = 6 THEN ' Type File
      GOSUB 60
    ELSEIF itemnum = 7 THEN ' Quit
      GOSUB 70
    END IF
  END IF
  RETURN

' Print With Header added in v1.1 2-23-89.

50:
  f$ = "null"           ' PRINT w/header
  WHILE f$ <> ""         ' Force first While
    f$ = FILES$(1,"TXTSRC") ' file dialog box
    IF f$ <> "" THEN      ' If not Cancel ..
      HPrintFile(f$)      ' from inCider.Tools
    END IF
  WEND
  MENU                  ' Unhighlight menu bar
  RETURN

' DoMenu v1.1 2/23/89
' Eliminates flicker when menu is being built.

SUB DoMenu
  FOR p = 1 TO 6         ' Create menu bar
    FOR e = 0 TO 12 STEP 4 ' End screen flicker
      PALETTE p,e+0,1,1,1 ' by whitening-out the
      NEXT e              ' menu bar.
    NEXT e                ' Thanks to Lee Rimar
    MENU 1,0,1,"File"     ' of Absoft for code.
    MENU 1,1,1,"New"      ' Build FILE menu
    MENU 1,2,1,"Edit"     ' and its entries ...
    MENU 1,3,1,"Delete"
    MENU 1,4,1,"Print"
    MENU 1,5,1,"Type"
    MENU 1,6,1,"Quit"
  FOR p = 1 TO 6         ' Restore palette ..
    FOR e = 0 TO 12 STEP 4
      PALETTE p,e+0,0,0,0
    NEXT e
  NEXT e
END SUB

' The End. (inCider.Shell v1.1)

```



## Listing 3. inCider.Tools v. 1.1.

```

'-----
' File : inCider.Tools   Version 1.1  2/13/89
' (C)1989, Joe Abernathy. All Rights Reserved.
' Compiler : AC/BASIC for the Apple IIgs.
'-----
' This library requires global variables declared in
' inCider.Shell.
' Add the procedure HPrintFile below to your Version
' 1.0 inCider.Tools file.
'-----
' Procedure HPrintFile -- Print a file (added 2/13/89)
' You must pass name of file to print in thfile$. If
' the headers do not fall precisely at the top and
' bottom of pages on your printer, adjust the "Loop%"
' counter.
SUB HPrintFile(ThFile$)
  SHARED FileNum
  Counter% = 1
  Loop% = 1
  FileNum = FileNum + 1
  Tmp$ = "File: " + ThFile$
  Tmp% = LEN(tmp$)
  Tmpl% = LEN("Page Number: XX")
  Tmp% = (80-(tmp% + tmpl%))
  OPEN ThFile$ FOR INPUT AS FileNum
  OPEN "LPT1:PROMPT" FOR OUTPUT AS FileNum + 1
  i$ = "File: " + ThFile$ + "    "...
  ... + "Page Number: " + (STR$(Counter%))
  WHILE NOT EOF(FileNum)
    IF Loop% = 1 THEN
      PRINT #FileNum + 1, i$
      PRINT #FileNum + 1, ""
      Loop% = 3
    END IF
    LINE INPUT #FileNum, a$
    PRINT #FileNum + 1, a$
    Counter% = Counter% + 1      ' the page number
    Loop% = Loop% + 1          ' internal counter
    IF Loop% = 33 THEN
      PRINT #FileNum + 1, ""
      PRINT #FileNum + 1, SPC(24);
      PRINT #FileNum + 1, "Date: ";DATES;...
      ... " "; "Time: ";TIMES
      PRINT #FileNum + 1, ""
      Loop% = 1
    END IF
  WEND
  CLOSE #FileNum
  CLOSE #FileNum + 1
  FileNum = FileNum - 1
END SUB
'-----
' The end. (v1.1 inCider.Tools additions)

```

## Listing 4. Set GS/OS prefix.

### Set GS/OS Prefix

This command will prompt you for a volume name to which the operating system should default for disk operations.

You can press RETURN by itself to abort out of Set Prefix. The command line prompt will accept all standard wildcards and extensions as a part of the file name argument.

Use the "C"atalog command to view the files in the new default directory.

Example: Enter new prefix: \*/SRC/BASIC/CDA/

for example, in my writing the on-line help function included in **Listing 1** (procedures BringHelp and DoHelp). Occasionally, it's caused major changes in a program's design.

To "finish it right," you need to weigh the expectations of what you want the program to do against what it actually does and what someone else might try to make it do. These variables change with every program, but you can observe three general rules:

**Document it.** Explain in common language what every feature is, how to invoke the program's features and make the most of them, and how to abort an operation. You might choose to write your documentation in the form of a manual, put it on line, or both.

**Make it bulletproof.** If something you write crashes someone's computer, that isn't going to be pleasant for either of you. In ShowCDA, I found that Micol Advanced BASIC's FILE function contained a bug that could make the program crash. The CheckInput procedure, that ShowCDA calls after every filename prompt is a filter that prevents a crash.

## GS BASICS Q&A

### FINDER PROBLEMS

I'm using TML BASIC for the IIgs and am trying to run the Finder directly from a program. Whenever I enter RUN FINDER, the Finder loads up, but I get a FATAL SYSTEM ERROR. I've tried renaming the START program in the \*/SYSTEM folder to MYSTAR and then RUNning MYSTAR. The same result occurs. Any suggestions?

Christopher Smeds  
New York

*I know what you're asking, but I'm not sure why. To invoke the Finder directly you need to RUN \*/PRODOS.*

*Generally, the Finder will have launched a program. If so, the fastest way back to the Finder is the END statement, which is actually a ProDOS QUIT that returns to the most recent*

*program executed before the one calling END. Because you'll have used the Finder to launch most programs, it's usually the most recent program.*

*Using the RUN statement to launch the Finder directly defeats the purpose of the GS' powerful memory-management scheme. It will also add quite a bit to the turnaround time required to use the program you're writing.*

### TOOLS FOR LEARNING

I've programmed in Applesoft and Microsoft QuickBASIC; now I want to learn more about programming on my IIgs, particularly the Toolbox, in which I'm a complete novice. What's the best way to start learning about the Toolbox and how to use it with TML BASIC?

Also, can you recommend any references for using TML BASIC?

Carl Beyer  
Midland, MI

*You've already found the best source of information for Toolbox programming under BASIC—inCider's Apple IIgs BASICS. Unfortunately, no one's written a book on this subject yet.*

*Two books you should look at are The Apple IIgs Toolbox References, Volumes 1 and 2, available through Addison-Wesley (6 Jacob Way, Reading, MA 01867, 617-944-3700), B. Dalton bookstores, and the Apple Programmers and Developers Association (APDA, Apple Computer, 20525 Mariani Avenue, Cupertino, CA 95014, 408-996-1010).*

*These volumes have no BASIC-spe-*

*cific information, and in many instances presume a knowledge of C or assembly language on your part. Still, they're the only source of the information necessary to issue tool calls.*

*For source-code examples, order one of the suite of on-disk source samplers available from APDA. There won't be any BASIC, but there will be TML Pascal, which is the dialect most readily portable to TML BASIC.*

*A starter TML BASIC program shell is available free through inCider—send a stamped, self-addressed envelope to receive a copy. This shell will handle all the background work necessary to let you write desktop-style programs.*

*Ninety-five percent of the BASIC source code I write here is quickly portable to TML. Some will be specific to TML, such as the program shell. □*

*Bulletproof* takes on additional meaning when you begin composing programs that write data to disk. You need to ensure that no action, no matter how arcane, can ruin the data on disk. **Sand the edges.** After you've written the program, give yourself two more days just to play with it. Look over the source code, and remember what parts you rushed through trying to finish.

Now that you've met the big challenge, encourage yourself to start tackling other programming issues. Challenge yourself to improve the quality of your programming with every procedure you write. With an endless supply of ideas, you'll have plenty of opportunity to refine your GS programming skills. □

CONTRIBUTING EDITOR JOE ABERNATHY IS A JOURNALIST WITH *THE HOUSTON CHRONICLE*. HE'S A CERTIFIED APPLE DEVELOPER AND THE AUTHOR OR COAUTHOR OF EIGHT APPLE II PROGRAMS. WRITE TO HIM AT P.O. BOX 66046, HOUSTON, TX 77266-6046. ENCLOSE A STAMPED, SELF-ADDRESSED ENVELOPE IF YOU'D LIKE A PERSONAL REPLY.