

## Launching into BASIC

Joe Abernathy

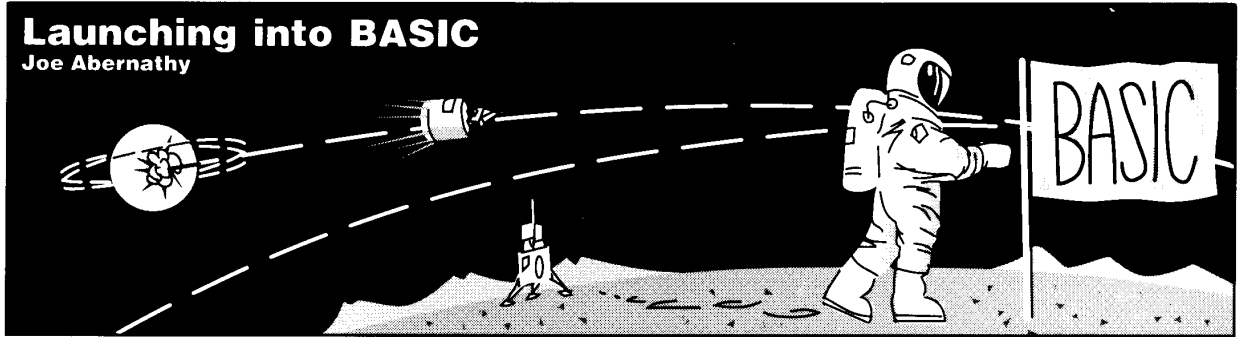


Illustration by Robert Williamson

# G

ood software, in large part, is the product of good software tools. The theory, and the fact, is that you should only have to write the code to handle any particular task once, eventually building a library of tools to meet most needs.

One high profile example of such a library is the Apple IIGS Toolbox, whose purpose is to make the familiar desktop metaphor readily available. Apple Computer didn't invent the idea of software tools, however, it does the best job so far and now is riding the wave.

As nice as the Toolbox is, it leaves room for growth. To fill the gap, we can create personal tool libraries to manage every aspect of applications programming, including more convenient Toolbox access and business quality data handling.

Launching Into BASIC this month presents the first tools in the *Call -A.P.P.L.E.* libraries. Included are low-level text, window, and dialog handlers; a startup screen generator to copyright your programs; and a goodbye usage screen generator. The high level tools are written for AC/BASIC. Some of the low level tools, and all of the theory, also serve TML BASIC. (See the Roar of the Crowd heading if you need the main source code shell for either language.)

These libraries represent a relatively new sophistication in Apple II BASIC, that of structured programming. C and Pascal previously had the field alone until both the TML and AC/BASIC IIGS compilers provided support for structured programming.

This may not be the style of programming you are accustomed to, however, adopting it will provide significant benefits; benefits which run deeper than just making source code reusable. Why this is so will become apparent as we discuss how to design and write a good software tool.

### It's in the Wrist

Procedural programming, at first blush, would appear to be something practiced only in clean rooms. The reality is that the best structured programming takes place on a level that makes it ideal as a learning vehicle. With few exceptions, the best tools are simple procedures growing more sophisticated in the context of their supporting fabric.

For an example, look ahead to Listing 3, at the subroutine MarkLeader. By itself, MarkLeader prints a string, followed by a dividing leader of periods, then right justifies a second string. The end effect is similar to that of the index of your local newspaper.

MarkLeader does not know what strings it is to print until you pass them nor does it know the width the total construction should occupy. It does not even know if it should print to the screen, a disk file, or a printer. These apparent failings are

MarkLeader's strengths: Control structures can use it and so can subroutines to format an entire file, print it two ways, and save it. All of this without using redundant code.

The idea, then, is not to create sophisticated procedures, but rather to do something sophisticated in small enough steps that may be of general utility. Working together, just a few good functions can accomplish a lot, e.g., Hello and GoodBye in Listing 5. This pair of procedures demonstrate how text handlers can be used to create a high level function which itself can become part of a greater whole.

One cost associated with all of this is a forced parting with some of the comfort of earlier development platforms. In order to write programs in terms of tools, GOTO is suspect and planning is necessary. You don't have to relearn flowcharting, just spend the time to envision your program as an ordered collection of capabilities.

To write a software tool, identify a recurring function in your programming, isolate the specific steps it requires, and group them as a procedure in a fashion offering the most general appeal. If you see a need for several related procedures, make a separate library addressing the topical area to which they apply.

### Pass the Parameter

Structured BASIC is a product of C and Pascal. In C, the building block is the function while in Pascal procedures are also available in addition to functions. In Pascal, functions return a value whereas procedures perform a set of tasks, optionally using passed parameters. The C function can perform a set of tasks and indirectly modify values outside the function.

Our procedures resemble C in that they can accept arguments passed by value or by reference. If we pass an argument to a procedure by value, no changes are made to contents of variables in the main program. If we pass the argument by reference, the main program variable used as the argument may be changed along with the corresponding variable in the procedure.

Pascal functions only return a value. Its procedures perform a set of like tasks, allowing arguments to be passed by reference. Our procedures are like those of Pascal in form and in that, you are shielded from the data manipulation that is actually taking place.

When you pass an argument by reference, you are passing to the procedure a pointer to the memory address of the argument. In C and assembly you work with the pointer directly. However, BASIC and Pascal, both teaching languages, hide this address manipulation.

Pointers are powerful tools as evidenced by their heavy use by the Apple IIGS Toolbox. Even though you don't have to know what lies behind procedures in order to write and use them, you may want to open your eyes to the theory of pointers in order to

have a kinder, gentler experience with the Toolbox later. One exception is if you plan to stick with AC/BASIC, but even here, advanced Toolbox access involves the study of data structures.

### What's In It

When you save time, it often means you ultimately have made money. Considering this as a worthy benchmark, time savings would have to be the best aspect of structured programming. Sure, it initially takes longer to plan and write procedures than to string code together, but there is a payoff in the long run.

Suppose you develop a set of low level tools to handle customer, vendor, employee records, and perhaps some business math. What you have is the supporting fabric for a suite of database and planning applications. If you happen to have written these tools in AC/BASIC, you have almost immediate portability to the Macintosh where productivity software is the leading growth market.

Another case study might be if you are a teacher or own your own business. If you develop a set of core software capabilities specific to your industry, you likely have something that can be turned into extra cash or a second stream of revenue.

There are payoffs for the casual programmer as well. For instance, we can create a dialog generator through which entire data management screens can be implemented. Using TML BASIC, the speed and power are available to write games as good as any available on the market today.

In discussing structured BASIC, there are compiler differences that have some meaning. TML supports a style of procedural programming close to that of Pascal, with clearly structured procedures and functions. AC/BASIC has a less clear library interface requiring some extra planning on your part in order to keep things orderly.

### The Source Code

The source code listings implement a startup screen that will automatically copyright your programs, a goodbye screen whose main purpose is window dressing, and a set of general routines to support the high level procedures. Two general purpose dialog generators are included.

(Note that some argument names do not directly match those used in the source listings. The variables in the listings are correct as shown; we just expanded the names for the discussion in order to make the procedures easier to understand.)

Listing 1 shows the changes and additions to the AC/BASIC source code shell needed to implement the new subroutine libraries. This shell has been presented in the previous two installments of this series (January 1989 and February 1989 *Call-A.P.P.L.E.*).

The label INIT should be added right above SETUP in your shell and the redundant values deleted from SETUP. Add the values under the label SETUP in Listing 1 to your existing SETUP. The rest of the code just shows how and where in the program the new procedures can be called.

INIT is kept separate from SETUP for a reason. Eventually, SETUP will consume more time reading data files and the like. The Hello screen will be displayed during this process, hiding the time lag from the user. In order to achieve this effect, you must start the program in the same fashion demonstrated in Listing 1.

For AC/BASIC, the \$Include lines used to attach the new library files in your main program must be placed at the bottom of the main program along with any SUB declarations.

For TML BASIC, add a LIBRARY statement to the shell for each new library. Libraries must be called in each segment which uses their routines.

In Listing 2, the library SHELL.TOOLS includes several low level system utilities of general interest.

**NewWindow (stylenum%):** Prepares and opens a new full screen window that is centered a few pixels away from the screen boundary on all four sides. STYLENUM% is any legal window style. NewWindow increments the global variable WINDOWNUM, the window counter.

**NewDWindow1:** Opens a window of a size and location suitable for a single message dialog. The window holds a message up to 65 characters in length and one or two buttons. It also increments the window counter.

**NewDWindow2:** Opens a window of a size suited to a dialog with a two line message. The window holds two lines up to 65 characters in length and one or two buttons. It also increments the window counter.

**ShutWindow:** Closes the most recently opened window and decrements the window counter.

**Pause (howlong%):** Performs a standard wait loop of the form FOR I = 1 to X : NEXT I, where X is the value passed in HOWLONG%.

**GetDate:** Reads the date maintained by the internal clock and stores it in the global variable THEDATES. Other values returned are MONTH%, 1-12 (Jan.-Dec.); DAY%, 1-31; and YEAR%, 19xx.

**GetTime:** Reads the internal clock and stores the time in the global variable THETIMES. Other values returned are HOUR%, 1-24; MINUTE%, 1-60; and SECOND%, 1-60. The label SETUP calls GetTime then sets the string STARTTIMES for later use by the GoodBye procedure.

If you are using TML BASIC, GetTime and GetDate correspond with the DoTime and DoDate procedures given in the last installment of this series of articles. You can add Pause to your TML library by replacing the SUB and END SUB declarations in this listing with PROC and END PROC.

In Listing 3, the library TEXT.TOOLS contains low level string manipulation procedures. Each routine performs justification of one or more strings within a given width, and can output to the screen, a printer, or a disk file. Listing 6 shows the Pascal version of this library.

Looking at the listings for Hello and GoodBye, you can see that the screen width value passed does not actually match the number of characters you would expect. This is because Geneva, the IIGS system font, employs proportional spacing, throwing things out of kilter for routines such as these. If you have the TechAlliance IIGS fonts disks, you have a monospaced font that can circumvent this problem. Otherwise, figure that Geneva 8-point generally requires a width parameter about twice what you would expect for the output width. We avoided the Print Using statement because it has bugs in TML's implementation.

**QuadCenter (string\$,width%,filename%):** Center justifies a string in a given width. If a file number of 0 is passed, QuadCenter prints to the screen. If any other value is passed, printing will be routed to the output device currently opened with that file number.

**QuadRight (string\$,width%,filename%):** Right justifies a string. The qualifiers operate the same way as with QuadCenter.

**MarkLeader (string1\$, string2\$, width%, filename%):** Left justifies STRING1\$, right-justifies STRING2\$, and fills out WIDTH% columns with a leader (...) between the two. FILENUMBER% operates the same way as with QuadCenter.

**MarkWhite (string1\$, string2\$, width%, filename%):** Left justifies STRING1\$, right justifies STRING2\$, and fills out WIDTH% columns with white spaces. FILENUMBER% operates the same way as with QuadCenter.

*Exercise:* Write a procedure that justifies three or more fields in a given width. This procedure is designed to handle tabular lists of information.

In Listing 4, the library DIALOG.TOOLS has two dialog generators, one to do a single message dialog and another to do a two message dialog. Both will generate one or two buttons, optionally setting a proceed/cancel flag. DIALOG.TOOLS requires SHELL.TOOLS.

**DoDialog1 (text\$,numbuttons%):** Generate a single message dialog box. The text string can be up to 65 characters long. The dialog will automatically have an "OK" button. If you pass the value of 2 for NUMBUTTONS%, a "Cancel" button will also be generated.

If the user selects Cancel, the value ABORT will be set to a non-zero value. In your program, you can use a test such as IF NOT ABORT THEN PROCEED. (Use Abort% with TML BASIC.) Be sure not to use ABORT as a global value elsewhere in your program.

**DoDialog2 (text1\$,text2\$,nummsg%, numbuttons%):** Generate a two message dialog box. Everything works just like DoDialog1.

*Exercise:* These two procedures are widely useful, but basic. Write a dialog generator that will produce buttons with custom text and set a branching flag. Write one that will generate radio buttons.

In Listing 5, the BYLINE.TOOLS library demonstrates most of the low level routines. It will use values declared in INIT to automatically generate a title/copyright screen for your programs, as well as an exit screen. BYLINE.TOOLS requires SHELL.TOOLS and TEXT.TOOLS.

In order to use the GoodBye screen, you must have custom FILE and EDIT menus. This approach is used by our AC/ BASIC shell.

**Hello (program\$, owner\$, byline\$, publisher\$, version\$):** Generates the copyright screen. The program name will be centered using larger type with break-out rules. Owner\$ should be the name of the licensee; byline\$ is your name; publisher\$ is the distributor; and version\$ is the version number of this release of the program.

**EndHello:** Closes the Hello screen after main SETUP is complete. EndHello calls the Pause loop so that the window will be displayed long enough to read. Once your SETUP grows more time consuming, this Pause call can be adjusted or removed.

**GoodBye (program\$, owner\$, byline\$, publisher\$, version\$, mytime\$, mydate\$):** The variables mytime\$ and mydate\$ must be obtained by calling GetTime and GetDate prior to calling GoodBye. Everything else works the same as with Hello.

*Exercise:* These procedures work only in 640 pixel graphics mode. Rewrite them to optionally use 320 pixel graphics mode. (Under SETUP, the value Res is set to the screen width. Rewrite GoodBye to figure total time online.)

## Summary

You don't have to stop here. Use these libraries, but keep in mind that they are incomplete. You can create new libraries of your own or enhance these. TEXT.TOOLS in particular is ripe for further development.

Once you have begun, you'll find one benefit that can't be quantified — structured programming is fun. It is rewarding to identify the solution to a problem, write the procedure, fine-tune it, and use it. ■

## The Roar of the Crowd

Cecil Fretwell, Technical Editor — *The TML shell hangs if the \*/SYSTEM/FONTS/ directory has been deleted from the GS/ OS boot disk in order to save room.*

Joe Abernathy — *There is no Font Manager error code that can be checked to circumvent this situation. Therefore, the directory always should be there even if you don't plan to use it. Even though it wastes a block, it is a part of the 16 bit system software.*

*A workaround is to write a procedure to test for the existence of the file name "FONTS" of type DIR in the \*/SYSTEM/ directory. If it is found, allow the program to construct the Font menu. Otherwise, branch out of Font setup.*

If you have a question or contribution, you can reach us at: AppleLink Personal Edition address JOEA17; developers' AppleLink D1370; or write to Launching Into BASIC in care of Call -A.P.P.L.E.

Also, The Cortland Project, Europa, 1200/2400 bps, (713) 526-0714. Launching Into BASIC source code listings available for download. Electronic posting of questions for Call -A.P.P.L.E. Consultant's Corner.

## Review Board

1. *Your First BASIC Program*, Rodney Zaks, Sybex, 2344 Sixth Street, Berkeley, CA. 94710. This book introduces traditional BASIC from the ground up for those new to computers. Dr. Zaks is one of the finest authors of introductory language reference books.
2. *Celestial BASIC — Astronomy on Your Computer*, Eric Burgess, Sybex. Burgess, a fellow of the Royal Astronomical Society and a widely published journalist, presents a suite of programs for computerized stargazing. Pertinent math and astronomical calculations are covered.
3. *BASIC Programs for Scientists and Engineers*, Alan R. Miller, Sybex. Dr. Miller's book reflects his background as a Ph.D. engineering graduate of Berkeley and long time teacher of methods for engineering programming. He presents over 60 scientific algorithms, including curve fitting, vector and matrix math, numerical integration, and statistics.

## Listing 1

```

\
\ The following code demonstrates how to use
\ the subroutine libraries in your own pro-
\ grams, using the AC/BASIC shell.
\
\ In all listings, a line ending in ellipses
\ .. followed by a line starting in ellipses
\ indicates that a line was broken for
\ typographical purposes.
\
\ At the start of the program, call Hello ..
gosub Init          \ Setup HELLO
Hello title$,owner$,author$,company$,release$
gosub Setup         \ Main set up
EndHello           \ Erase Hello ..

Main:
\ Demo the dialog generators:
DoDialog1 "One text string, ..
          ..one-button dialog.",1
DoDialog1 "One text string, ..
          ..two-button dialog.",2
DoDialog2 "Two text strings","",..
          ..in a one-button dialog.",2,1
DoDialog2 "Two text strings","",..
          ..in a two-button dialog.",2,2
\ When QUIT is selected, call GoodBye,
\ then exit:
GoodBye title$,owner$,author$,company$,..
          ..release$,starttime$,thetime$
end          \ End of main program.

Init:
owner$ = "Call -A.P.P.L.E."      \ Set up HELLO
author$ = "Joe Abernathy"        \ Licensee
title$ = "My Program"            \ Author
company$ = "First Word"          \ Title
release$ = "1.0"                 \ Publisher
res=640                          \ Release #
lwin = 25                        \ Screen res
twin = 23                        \ Window size
rwin = res - 25
bwin = 200 - 10
year$ = right$(date$,4)
return

\
\ Setup:          \ Main setup
\ .. Read any necessary data files, etc.,
\ while the Hello screen is displayed.
GetDate
GetTime
starttime$ = thetime$
return

\
\ At end of program, attach library files.
\
\ Use the name of the volume holding
\ your libraries:
$Include "/CMS/ACBASIC/INTF/SHELL.TOOLS"
$Include "/CMS/ACBASIC/INTF/TEXT.TOOLS"
$Include "/CMS/ACBASIC/INTF/DIALOG.TOOLS"
$Include "/CMS/ACBASIC/INTF/BYLINE.TOOLS"

```

## Listing 2

```

\
\ File: SHELL.TOOLS (C)TechAlliance
\ By Joe Abernathy All Rights Reserved.
\ Compiler: AC/BASIC and compatibles.
\
\ Contains: NewWindow, NewDWindow1, GetDate,
\ NewDWindow2, ShutWindow, Pause, GetTime
\
\ Subroutine NewWindow:
\
sub NewWindow(style%) shared
windownum = windownum + 1
window windownum,"", (lwin,twin)-..
          ..(rwin,bwin),style%
end sub

\
\ Subroutine NewD(Dialog)Window1:
\
sub NewDWindow1 shared
windownum = windownum + 1
window windownum,"", (lwin-5,twin+10)..
          ..-(rwin+5,bwin-100),2
end sub

\
\ Subroutine NewD(Dialog)Window2:
\
sub NewDWindow2 shared
windownum = windownum + 1
window windownum,"", (lwin-5,twin+10)..
          ..-(rwin+5,bwin-85),2
end sub

\
\ Subroutine ShutWindow:
\
sub ShutWindow shared
window close windownum
windownum = windownum - 1
end sub

\
\ Subroutine Pause:
\
sub Pause(HowLong%)
for i = 1 to HowLong%
next i
end sub

\
\ Subroutine GetDate:
\
sub GetDate shared
thetime$ = date$
month% = val(left$(startdate$,2))
day% = val(mid$(startdate$,4,2))
year% = val(right$(startdate$,4))
end sub

\
\ Subroutine GetTime:
\

```

```
sub GetTime shared
  thetime$ = time$
  hour$ = val(left$(thetime$,2))
  minute$ = val(mid$(thetime$,4,2))
  second$ = val(right$(thetime$,2))
end sub
```

### Listing 3

```

\ -----
\ File: TEXT.TOOLS (C)TechAlliance
\ By Joe Abernathy All Rights Reserved.
\ Compiler: AC/BASIC and compatibles.
\ -----
\ Contains: QuadCenter, QuadRight,
\ MarkLeader, MarkWhite
\ -----
\ Subroutine QuadCenter:
\ -----
\           Centered Output
\ -----
sub QuadCenter(strng$,wdth%,filnum%)
  z$ = chr$(32) ` SPACE
  x% = (wdth% - len(strng$))\2
  z% = int(x%)
  for i = 1 to z%
    if filnum% then
      print #filnum%, z$;
    else
      print z$;
    end if
  next i
  if filnum% then
    print #filnum%, strng$
  else
    print strng$           ` Centered
  end if
end sub

\ -----
\ Subroutine QuadRight:
\ -----
\           Flush-right Output
\ -----
sub QuadRight(string$,width%,filnum%)
  z$ = chr$(32) ` SPACE
  x% = (width% - len(string$))
  for i = 1 to x%
    if filnum% then
      print #filnum%, z$;
    else
      print z$
    end if
  next i
  if filnum% then
    print #filnum%, string$
  else
    print string$
  end if
end sub

\ -----
\ Subroutine MarkLeader:
\ -----
\ Classified .....10B
\ Sports .....1C
\ -----
sub MarkLeader(str1$,str2$,width%,filnum%)
  x$ = str1$ + " "
```

```

  y$ = " " + str2$
  z$ = " "
  x% = (width% - (len(x$) + len(y$)))
  if filnum% then
    print #filnum%, x$; ` Flush left
  else
    print x$;
  end if
  for i = 1 to x%
    if filnum% then
      print #filnum%, z$;
    else
      print z$;           ` Pad line ..
    end if
  next i
  if filnum% then
    print #filnum%, y$
  else
    print y$           ` Flush right
  end if
end sub

\ -----
\ Subroutine MarkWhite:
\ -----
\ Classified .....10B
\ Sports .....1C
\ -----
sub MarkWhite(strg1$,strg2$,width%,filnum%)
  z$ = chr$(32) ` SPACE
  x% = (width% - (len(strg1$) + len(strg2$)))
  if filnum% then
    print #filnum%, strg1$; ` Flush left
  else
    print strg1$;
  end if
  for i = 1 to x%
    if filnum% then
      print #filnum%, z$;
    else
      print z$;           ` Pad line ..
    end if
  next i
  if filnum% then
    print #filnum%, strg2$
  else
    print strg2$           ` Flush right
  end if
end sub
```

### Listing 4

```

\ -----
\ File: DIALOG.TOOLS (C)TechAlliance
\ By Joe Abernathy All Rights Reserved.
\ Compiler: AC/BASIC and compatibles.
\ Requires: SHELL.TOOLS
\ Contains: DoDialog1, DoDialog2
\ -----
\ Subroutine DoDialog1:
\ -----
sub DoDialog1 (text$,numbttns%) shared
  abort = 0
  NewDWindow1
  locate 2,2
  print text$
  button 100,1,"OK", (lwin+20,bwin-160)-..
  .. (lwin+120,bwin-145),1
  if numbttns% = 2 then
```

```

        button 101,1,"Cancel", (lwin+135,bwin-..
            ..160)-(lwin+235,bwin-145),1
    end if
    while dialog(0)=0
        wend
    while dialog(0)<>1
        wend
    x = dialog(1)
    if x = 101 then abort = 1
    ShutWindow
end sub

\ -----
\ Subroutine DoDialog2:
\ -----

sub DoDialog2 (text1$,text2$,nummsg$,..
    ..numbuttons$) shared
    abort = 0
    NewDWindow2
    locate 2,2
    print text1$
    if nummsg$ = 2 then
        locate 4,2
        print text2$
    end if
    button 100,1,"OK", (lwin+20,bwin-145)..
        ..(lwin+120,bwin-130),1
    if numbuttons$ = 2 then
        button 101,1,"Cancel", (lwin+135,bwin-..
            ..145)-(lwin+235,bwin-130),1
    end if
    while dialog(0)=0
        wend
    while dialog(0)<>1
        wend
    x = dialog(1)
    if x = 101 then abort = 1
    ShutWindow
end sub

```

### Listing 5

```

\ -----
\ File: BYLINE.TOOLS (C)TechAlliance
\ By Joe Abernathy All Rights Reserved.
\ Compiler: AC/BASIC and compatibles.
\ Requires: SHELL.TOOLS, TEXT.TOOLS
\ Contains: Hello, EndHello, GoodBye
\ -----

\ Subroutine Hello:
\ -----

sub hello(prog$,owned$,byline$,pblshr$..
    ..,version$) shared
    NewWindow 2
    'Draw a window
    Light Gray Background frame:
    line (9,4)-(rwin-32,bwin-27),204,204,b
    Red background frame:
    line (35,120)-(rwin-58,bwin-34),17,17,b
    textsize 14
    line (125,25)-(475,25),204,204,b
    locate 3,2
    QuadCenter prog$,68,0 ' Prog name
    textsize 8
    line (125,80)-(475,80),204,204,b
    locate 8,3
    version$ = "Version " + version$
    QuadCenter version$,125,0
    locate 12,5

```

```

    x$ = "By " + byline$
    y$ = "(C)" + year$ + ", " + pblshr$
    MarkWhite x$,y$,100,0
    locate 16,7
    x$ = "Licensed to: " + owned$
    QuadCenter x$,100,0
end sub

\ -----
\ Subroutine EndHello:
\ -----

sub EndHello shared
    Pause 2000 ' Delay loop
    ShutWindow
end sub

\ -----
\ Subroutine GoodBye:
\ -----

sub GoodBye(prog$,owned$,byline$,pblshr$..
    ..,version$,mytime$,mydate$) shared
    NewWindow 2 ' Draw a window
    Light Gray Background frame:
    line (9,4)-(rwin-32,bwin-27),204,204,b
    Red background frame:
    line (35,50)-(rwin-58,bwin-34),17,17,b
    Setoff lines:
    line (50,57)-(rwin-71,57),204,204,b
    line (50,150)-(rwin-71,150),204,204,b
    locate 3,5
    MarkWhite prog$,version$,113,0
    x$ = "(C)" + year$ + ", " + pblshr$
    y$ = "By " + byline$
    locate 4,5
    MarkWhite x$,y$,102,0
    locate 9,7
    print "Bye for now, ";owned$;":"
    locate 11,11
    MarkLeader "Date",mydate$,79,0
    locate 12,11
    MarkLeader "Started",mytime$,80,0
    GetTime
    locate 13,11
    MarkLeader "Ended",thetime$,82,0
    Red divider line:
    line (80,125)-(rwin-155,125),17,17,b
    Pause 2000
end sub

```

### Listing 6

```

\ -----
\ File: TEXT.TOOLS (C)TechAlliance
\ By Joe Abernathy All Rights Reserved.
\ Compiler: TML BASIC.
\ -----

\ Contains: QuadCenter, QuadRight,
\ MarkLeader, MarkWhite

DEF LIBRARY TextTools
$DEBUG OFF
$KEYBOARDBREAK OFF

\ -----
\ PROC QuadCenter:
\ -----

        Centered Output

```

```

DEF PROC QuadCenter(string$,width%,filnum%)
  z$ = chr$(32) \ SPACE
  x% = (width% - len(string$))\2
  z% = int(x%)
  FOR i = 1 TO z%
    IF filnum% THEN
      PRINT #filnum%, z$;
    ELSE
      PRINT z$;
    END IF
  NEXT i
  IF filnum% THEN
    PRINT #filnum%, string$
  ELSE
    PRINT string$ \ Centered
  END IF
END PROC QuadCenter

\ -----
\ PROC QuadRight:
\ -----
\ Flush-right Output

DEF PROC QuadRight(string$,width%,filnum%)
  $ = chr$(32) \ SPACE
  x% = (width% - len(string$))
  FOR i = 1 TO x%
    IF filnum% THEN
      PRINT #filnum%, z$;
    ELSE
      PRINT z$
    END IF
  NEXT i
  IF filnum% THEN
    PRINT #filnum%, string$
  ELSE
    PRINT string$
  END IF
END PROC QuadRight

\ -----
\ PROC MarkLeader:
\ -----
\ Classified .....10B
\ Sports .....1C

DEF PROC MarkLeader(str1$,str2$,width%..
  ..,flnum%)
  x$ = str1$ + " "
  y$ = " " + str2$
  z$ = "."
  x% = (width% - (len(x$) + len(y$)))
  IF flnum% THEN
    PRINT #flnum%, x$; \ Flush left
  ELSE
    PRINT x$;
  END IF
  FOR i = 1 TO x%
    IF flnum% THEN
      PRINT #flnum%, z$;
    ELSE
      PRINT z$; \ Pad line ..
    END IF
  NEXT i
  IF flnum% THEN
    PRINT #flnum%, y$
  ELSE
    PRINT y$ \ Flush right
  END IF
END PROC MarkLeader

```

```

\ -----
\ PROC MarkWhite:
\ -----
\ Classified .....10B
\ Sports .....1C

DEF PROC MarkWhite(strg1$,strg2$,width%..
  ..,filnum%)
  z$ = chr$(32) \ SPACE
  x% = (width% - (len(strg1$) + len(strg2$)))
  IF filnum% THEN
    PRINT #filnum%, strg1$; \ Flush left
  ELSE
    PRINT strg1$;
  END IF
  FOR i = 1 TO x%
    IF filnum% THEN
      PRINT #filnum%, z$;
    ELSE
      PRINT z$; \ Pad line ..
    END IF
  NEXT i
  IF filnum% THEN
    PRINT #filnum%, strg2$
  ELSE
    PRINT strg2$ \ Flush right
  END IF
END PROC MarkWhite

\ -----
END LIBRARY

```

### Racer One-Liner

The following one line program was submitted by David Wicker. It is a penny arcade type of game whereby you steer a car down a road using the CAPITAL J and CAPITAL K keys. The J key moves the car to the left and the K key moves the car to the right on the screen. After you hit one of the keys, the motion continues in the desired direction until you either hit the other key to move the other direction or hit the SPACE bar to move straight down the road. If you hit the edge of the road or reach the end of the road, the game ends. See if you can change the PRINT "!" statement to cause the bell to ring. (Use the SPEED command to make the game go slower or faster. SPEED=255 is the fastest. SPEED=200 is a reasonably slow speed for beginners.)

IGS owners, your Control Panel Options must be set to NO for Keyboard Buffering, otherwise, once you start the car moving right or left, motion can not be stopped.

Spaces are included for readability in each line of the program. When you key in the program, do not enter spaces and use the question mark (?) instead of PRINT. Note the back slash character (\) used in the first line. When you enter the program, substitute a space for each back slash character.

```

10 TEXT : HOME :A$ = "##\\##": FOR A = 0 TO 23:
  PRINT TAB(17)A$: NEXT :A = 17:B = 20: FOR C =
  1 TO 23 STEP 1 / 8: VTAB 24: PRINT TAB( A)A$:
  HTAB B : VTAB C:D = 99 * ( SCRN( B - 1, INT
  (C) * 2)): PRINT "V": VTAB 1: PRINT INT (C):A
  = A + INT (3 * RND (1) - 1): A = A + (A < 2) -
  (A > 32):F = PEEK (49152) - 200:B = B -
  (F = 2) + (F = 3):C = C + D: NEXT : PRINT "!"

```