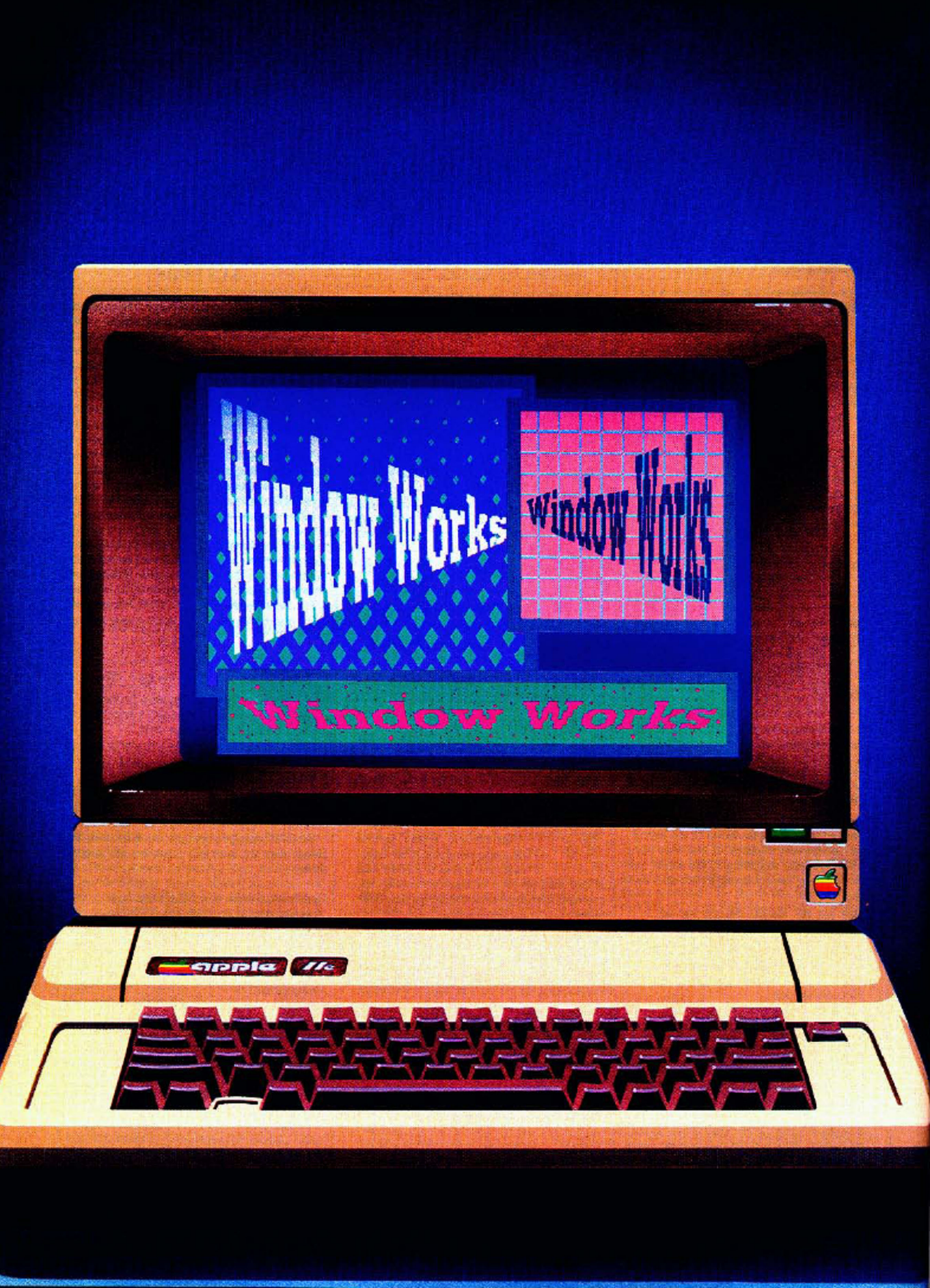
Window Works
Window Works
Window Works
apple //c

WINDOWORKS

COVER FEATURE

Add animated *windows to your Hi-Res screens. Then incorporate complete animation modules in your own programs.*

With WindoWorks, any Apple user — including those with no programming experience whatsoever — can produce up to 50 dazzling, animated windows on-screen, in less than 10 minutes. You can create seven different types of animation within the windows, including scrolling in four directions, plus flashing regions, regions that change colors and flashing window frames. By overlapping windows, you can invent special effects — like text that scrolls to the right halfway across the window, then bumps upward, goes back down and returns to scrolling rightward again. If you can imagine what your application should look like, then WindoWorks can help you achieve the results.

You need only two things before using WindoWorks:

1. A Hi-Res picture drawn with any utility, in any range of colors.
2. A disk with at least 50 sectors of space in which to save your finished masterpiece.

USING THE PROGRAM

When it's first run, WindoWorks asks Apple II Plus users if they can display lower-case. A simple Y or N answer will suffice.

In WindoWorks, you'll make selections from highlighted menus. You move through the menus by pressing the four Arrow keys on the Apple IIe; or on the II Plus, by pressing the Left and Right-Arrow keys, or the A and Z keys. On both machines, pressing the Return key makes the selection.

Options

Let's examine each of the main menu options in detail. Note that some options require that prior actions be performed; for example, the View Hi-Res Screen option will work only if you have loaded a picture. Likewise, you cannot delete or edit a window without having already added one. When you are viewing the Hi-Res screen, you may press the Escape key to return to the main menu.

The lower-right portion of the menu area displays several important pieces of information: the number of windows currently defined in memory (the limit is 50), the status of the Toggle Overlay option (described in detail below), and whether or not a Hi-Res picture has been loaded. If an option is not working, refer to these status messages for an indication of what's wrong.

View Hi-Res Screen — Allows you to view the Hi-Res screen in memory. It displays the location of all active windows as defined by the setting of OVERLAY (described below). To return to the main menu, press Return.

View Animation — If a picture is in memory and at least one window has been defined, the Hi-Res screen is displayed and the windows start moving. To stop the action and return to the main menu, press Return.

Add Window — This option displays the Hi-Res screen with a flashing rectangle in the center. You can use the rectangle to define up to 50 windows. Strategic placement of them — either side by side or overlapping — can produce spectacular effects (see Special Effects, below).

Moving the frame around and changing its size is a simple task. Initially, you'll be in control of the upper-left corner of the frame. Using any of the four Apple IIe Arrow keys (or on the II Plus, the Left- and Right-Arrows and the A and Z keys), move the corner around to see how it affects the rectangle's dimensions. Now press the Space bar, and you're in control of the lower-right corner; the same movement keys control this corner's location. By using these two corner points, any window can be defined.

Pressing Return terminates the window definition, and a menu is displayed with options for the type of animation to place in the window just defined. Indicate your choice with the Arrow keys and press Return.

Next, the speed of the window must be chosen. The speed value indicates the relative speed of the window. For example, a window scrolling at speed 4 will move more quickly than one scrolling at speed 3, and even more quickly than one at speed 2. Use any of the movement keys or press numbers 1-8 and press Return to select a speed. Don't worry about any errors you've made, as the Edit Window option allows you to make corrections.

Delete Window — Displays the Hi-Res screen and highlights the first window defined. Using any of the movement keys, choose the window you want to delete and press Return to delete it. Again, Escape may be used to cancel this function.

Edit Window — This option lets you select a window to edit by using the movement keys and pressing Return. The frame is displayed and you can redimension it, if desired. When the size and location are right, press Return a second time and you'll proceed through the type of animation and speed selection sequences that occur when you add a window.

Delete All Windows — All previously defined windows are erased and only the original Hi-Res picture is displayed. A special yes/no menu prevents accidental deletions.

Load Picture — Must be chosen before any window defining can take place. You are prompted to enter a file name and press Return. You will see the picture loading and hear two short beeps indicating that the picture is fully loaded. When selecting a picture, be sure that it is a binary file (file type B on the left side of the catalog) and that the length is 33 or 34 sectors if you use DOS and 17 blocks if you use ProDOS.

Load Window File — Requires that you enter a file name and press Return to complete the selection (as in Load Picture above). This time, though, you'll need to specify a binary file of three sectors (or two ProDOS blocks). Also, the first two characters of the file name should be 'W.', signifying a WindoWorks window file. The prefix is automatically supplied by the program when it saves a window file.

Save Window File — This option comes in handy if you want to take a break from defining windows to touch up the Hi-Res picture or just grab a cup of coffee. All windows in memory are saved in a special WindoWorks window file. Only the coordinates of the windows, the type of animation in each, and the speed settings are saved to disk — the Hi-Res picture is not part of the file. This information may be used for later changes to your creation, or perhaps to try out this set of windows in another picture.

You need to specify a file name for your WindoWorks window file, which, as noted above, is prefixed with 'W.' to distinguish it from other types of files. You are asked to enter a 15-character file name for this set of windows. The Left-Arrow key deletes one character at a time, Control-X erases the entire line, and pressing Return accepts the file name. The Escape key displays the main menu (be sure that a disk is in the drive and that it is not write-protected, or else an error message will be displayed).

Create Final Module — Creates a standalone module that contains the Hi-Res picture, all window definitions and the program that makes it all happen. When you select this option, you're instructed to enter a 15-character file name including an 'F.' prefix (signifying final module file type), as described in the Save Window File section above. Pressing Return saves the file to disk. For detailed use of this module, see Using the Final Module below.

Modify Data Drive — Tells WindoWorks where the data disk is located. If you use DOS 3.3, this option asks you to enter two values: first, a single digit for the slot (1-7), then the drive number (1-4). Press Return to maintain each displayed value. If you use ProDOS, you must enter the prefix for the data disk. Enter a prefix name with or without a leading slash (WindoWorks will add it if necessary). You should change the data disk prefix whenever you change disks.

Catalog Data Drive — Performs a CATALOG (or "CAT", for ProDOS users) of the selected data disk. If any errors are encountered, check to see that your disk is inserted properly, the drive door is closed, and the DATA DRIVE message at the bottom displays the proper prefix or slot and drive numbers.

Toggle Overlay — During the course of window creation, you may wish to position windows so they adjoin one another or overlap, as described in the Special Effects section below. To do this, you'll need to see the locations of other windows as you define the new one. The Toggle Overlay option turns on and off the display of window on the Hi-Res screen. When the main menu is displayed, a message in the lower-right portion of the screen shows the status of this option. Selecting it again turns it off. This option does not interfere with the operation of the final module or of any of the windows.

With WindoWorks, you can whip up fantastic animation in minutes!

Exit Program — A special yes/no menu double-checks your intention to quit. If you are sure you want to do so, all windows in memory are lost. Use this option when all the editing is completed and you are finished.

A SAMPLE SESSION

Let's get the feel of WindoWorks by trying some of the options. You'll need a standard 33- or 34-sector (17 ProDOS blocks) binary picture file and at least another 50 sectors (or 25 ProDOS blocks) available on the same disk. Make sure your disk contains WINDOWORKS (**Listing 1**) and WINDOWORKS.ML (**Listing 2**). Then get WindoWorks started by typing **RUN WINDOWORKS**, wait for the drive to stop. If you have a II Plus, at the title page, you should answer the question pertaining to lower-case display. Otherwise press Return to display the main menu.

Insert your disk with the picture file and set the data drive (shown at the bottom of the screen) to either the correct slot and drive for DOS 3.3 users, or to the appropriate prefix for ProDOS users. Then select the Load Picture option and enter the file name as prompted. If anything has gone wrong (a misspelled file name, wrong disk, etc.), an error message will appear and you'll need to try again.

Now that you have a picture in memory, select the Add a Window option to define your first window. Position and dimension a window, then press Return. Select the animation type from the menu that appears on the right side of the screen, and the relative

speed setting. The main menu will be automatically displayed when you're done. To see the window performing the selected animation, select View Animation. Repeat the process of adding a window to be sure you have the feel of it — this time creating a different window elsewhere on the screen and with a different animation type.

Next, select the Edit Window option and choose one of your two windows to edit. Having made your choice, you may alter the dimensions and location of the window, and the animation type or speed setting. Try leaving the window the same size, but alter the animation type and speed setting. View the animation to see the change.

Finally, select Create Final Module; enter a file name but retain the 'F.' prefix. A binary file that holds the windows, the picture and the program is saved to the data disk. The main menu is redisplayed and you can choose the Exit Program option to enter BASIC. Catalog your disk and locate the 'F.' file you just created. Now type **BRUN F.*filename*** and press Return. The Hi-Res picture is displayed with animated windows intact. Press Return to stop the action.

SPECIAL EFFECTS

Identical Superimposition

Some more advanced techniques are possible using WindoWorks. One simple technique is the identical superimposition of windows; that is, you create one window that scrolls upward, then define another window that is exactly the same size and place it directly on top of the previous one, but scrolling rightward. The result is a window that scrolls diagonally up and rightward. By selecting different scroll speeds for each, the angle may be altered, as well.

The same basic technique will work for scrolling windows combined with inverse or with color change, but not with the frame option. Frame works well only with non-moving windows, so for dynamic combinations, combine a fast inverse window with a slow frame or vice versa. You can even superimpose three or more windows.

Keep in mind that more animation on-screen results in slower overall speed. Whenever possible, try to keep the scrolling windows small. Horizontal scrolling is a bit slower than vertical, so some speed compensation might be necessary.

Partial Overlap

The partial overlap technique produces some of the most visually appealing displays. This involves placing one window on another (but not windows that are exactly the same size), including windows "within" windows, or windows that "hang over" from the inside to the outside of another window. At first you may get garbage results, but with a little practice, you'll be able to use partial overlapping to good advantage.

WINDOWORKS.DEMO (**Listing 3**) demonstrates each type of animation described in this article, from very simple to extremely complex.

ENTERING THE PROGRAMS

To enter WindoWorks, start by keying in the Applesoft program shown in **Listing 1**. Save it with the command:

SAVE WINDOWORKS

If you have an assembler that can handle multiple ORGs, enter the source code from **Listing 2** and assemble it using WINDOWORKS.ML for the name of the object file. If you are using Key Perfect, be sure that the block of memory coded by **line 560** actually contains zeros in the object file. If it doesn't, then load the object file with the command BLOAD WINDOWORKS.ML, enter the Monitor with CALL −151, perform the operations specified in step 3 below and perform the BSAVE command at the end of step 5.

If you don't have an assembler or if your assembler can't handle multiple ORGs, perform the following steps:

1. Enter the Monitor with CALL −151 and key in the hex code from **Listing 2** through **line 468** only. Save the code you have entered so far with the command:

   BSAVE WINDOWORKS.1,A$6000,L$347

2. Enter 8F60: at the Monitor prompt and continue entering object code starting at **line 481**.

3. To save needless typing, you can enter the block of zeros in **line 560** with the following sequence of Monitor commands:

   ```
   8FE9:0
   8FEA<8FE9.9146M
   ```

4. Continue entering the hex code starting at $9148 in **line 565**. When you are done, save this second portion of the program with the command:

   BSAVE WINDOWORKS.2,A$8F60,L$512

5. **BLOAD WINDOWORKS.1**
   BLOAD WINDOWORKS.2,A$6347
   BSAVE WINDOWORKS.ML,A$6000,L$859

If you need to make changes in the object code, BLOAD the appropriate segment at its original address, make the changes, save the segment and repeat the commands in step 5 to create the final program.

Finally, enter the Applesoft program shown in **Listing 3** and save it with the command:

SAVE WINDOWORKS.DEMO

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

USING THE FINAL MODULE

Now that you've used WindoWorks to design a graphic masterpiece, you're ready to incorporate it into your own programs, whether BASIC or machine language. This is accomplished with some simple programming tricks.

From BASIC

If you are programming in BASIC, you should keep a few details in mind. All WindoWorks animation is carried out on Hi-Res page 2, so your programs should use the HGR2 command in place of HGR. The program variables are stored above Hi-Res page 2. Under DOS 3.3, make the first line of your program:

```
HIMEM: 16384
```

Under ProDOS, use the following statement instead:

```
HIMEM: 15360
```

This will protect Hi-Res page 2 from being overwritten by variables. Thereafter, when you want to recall a module, you just type the following:

PRINT CHR $ (4) ; "BRUN F.*filename*"

where *filename* is the name you specified when saving the module. Repeat this process as necessary for multiple displays in the same program. When activated in this manner, the Hi-Res screen is displayed and the animation begins immediately. It continues until you press any key. Then the BASIC program continues with the next statement.

From Machine Language

If you are programming in machine language, remember that the graphics are displayed on Hi-Res page 2, so program code shouldn't be stored in locations $4000-$5FFF. Final modules are stored starting at $95FF and building down. Locations $8F60-$95FF hold the scrolling routines and the picture unpacker. The actual picture data starts at $8F5F and builds down in memory, depending on the size of the packed picture. If you need to use memory locations $6000-

$8F5F, then, check the storage locations of each module so you'll know how low in memory they go. A simple BRUN is all that is necessary to load and execute the animation package.

PROGRAM STRUCTURE

WindoWorks actually consists of three segments:

1. The BASIC portion (**Listing 1**), which is the user interface for window design, editing, etc.
2. The run-time module, which contains the animation code
3. The edit-time module, which performs the editing process features

Modules 2 and 3 are combined in **Listing 2**. The edit-time module compacts the Hi-Res picture immediately after loading, then moves it next to the run-time module at location $8F60. The edit module also contains the audio routines and the window sizing and positioning routines.

The run-time module contains: seven different animation routines, the picture unpacker, and the table of window data (which includes the locations of windows, the animation type of each and the speed settings). It also holds the control handler, which activates the windows.

I would advise machine language programmers to use my special routines SaveReg and RestReg to maintain the 6502 registers. SaveReg is called every time a subroutine is entered and RestReg is called when it is exited. These short routines ensure that all three registers are not destroyed, the stack is not altered, and that the zero page locations used are preserved. They are handy for maintaining orderliness in a program that does a lot of JSRing around.

Why settle for sluggish BASIC animation or slogging through the long, hard process of machine language programming? With WindoWorks, you can whip up fantastic animation in minutes!

Listing 1 for WindoWorks

WINDOWORKS

```
1  REM **********************
2  REM *      WINDOWORKS      *
3  REM *  by  Bob Thrasher   *
4  REM * Copyright (C) 1987 *
5  REM * by MicroSPARC, Inc *
6  REM * Concord, MA  01742 *
7  REM **********************
10  TEXT : PRINT  CHR$ (12) CHR$ (21): HOME :
     HIMEM: 15360: REM ** HIMEM=$3C00
20  GOTO 1160
30  REM ** DOS or ProDOS?
40 PD = ( PEEK (48896) = 76):Z = 1 - PD: IF P
    D = 0 THEN  VTAB 24: HTAB 1: PRINT "SLOT
     "DS", DRIVE "DD;: RETURN
50  IF DP$ = "" THEN  PRINT D$"PREFIX": INPUT
     DP$
60  VTAB 24: HTAB 1: PRINT "PREFIX: "DP$;: RETURN

70  REM ** Add Prefix/Suffix to H$
80  IF PD = 0 THEN H$ = H$ + ",S" +  STR$ (DS
    ) + ",D" +  STR$ (DD): RETURN
90 H$ = DP$ + H$: RETURN
100  REM ** Various Subroutines
110  VTAB A: HTAB B: PRINT "Press any Arrow";
     : VTAB A + 1: HTAB B: PRINT "or A & Z to
     ";: VTAB A + 2: HTAB B: PRINT "move then
      press";: VTAB A + 3: HTAB B: PRINT "Ret
     urn if OK";: RETURN
120  POKE 34,15: HOME : TEXT : RETURN
130  CALL A(5): POKE D + 2,0: POKE D + 7,0: POKE
     D + 5,0: POKE D,0: RETURN
140  POKE 49168,0: VTAB 1
150  IF  PEEK (49152) < 128 THEN 150
160  GET G$: VTAB 1: HTAB 1: PRINT : GOSUB 29
     0: RETURN
170  POKE 49168,0: VTAB 1
180  IF  PEEK (49152) < 128 THEN 180
190  GET G$: PRINT : RETURN
200  GOSUB 320: VTAB A + 2: HTAB 13: PRINT F$
     ;: GOSUB 140: RETURN
210 A = 16:B = 21:CM = 2: GOSUB 330: RETURN
220 A$ = "Load a Hi-Res picture first":A = 16
     : GOSUB 300: GOSUB 200: GOTO 1280
230 A$ = "No windows in memory":A = 16: GOSUB
     300: GOSUB 200: GOTO 1280
240  VTAB 1: HTAB 1: PRINT : PRINT D$H$: RETURN

250  ONERR  GOTO 270
260  VTAB 1: HTAB 1: PRINT : PRINT D$"BLOADWI
     NDOWORKS.ML": RETURN
270  PRINT "FATAL ERROR -": PRINT : PRINT : PRINT
     "THIS PROGRAM REQUIRES THE BINARY FILE":
      PRINT : INVERSE : PRINT "WINDOWORKS.ML"
     ;: NORMAL
280  PRINT " TO BE ON THE SAME DISK": PRINT "
      AND IT IS MISSING.": PRINT : PRINT "PRO
     GRAM STOPPED.": END
290  POKE A(1),10: POKE A(3),10: CALL A(7): RETURN

300  FOR E = 1 TO 10: POKE A(3),2: POKE A(1),
     20: CALL A(7): POKE A(3),2: POKE A(1),50
     : CALL A(7): NEXT : RETURN
310  FOR E = 100 TO 10 STEP  - 10: POKE A(3),
     3: POKE A(1),E: CALL A(7): NEXT : RETURN

320  VTAB A: HTAB 20 -  LEN (A$) / 2: PRINT A
     $;: RETURN

330  GOSUB 110: IF CM = 1 THEN 380
340  IF CM = 2 THEN 400
350  VTAB 1: HTAB 3: POKE 32,2: PRINT "View H
     i-Res Screen": PRINT "View Animation": PRINT
     "Add Window": PRINT "Delete Window": PRINT
     "Edit Window"
360  PRINT "Delete All Windows": PRINT "Load
     Picture": PRINT "Load Window File": PRINT
     "Save Window File": PRINT "Create Final
     Module"
370  PRINT "Modify Data Drive": PRINT "Catalo
     g Data Drive": PRINT "Toggle Overlay": PRINT
     "Exit Program":V1 = 1:V2 = 14:H1 = 1:H2 =
     23: GOTO 410
380  VTAB 1: HTAB 27: POKE 32,26: PRINT "Scro
     ll Up": PRINT "Scroll Down": PRINT "Scro
     ll Left": PRINT "Scroll Right": PRINT "I
     nverse": PRINT "Color Change": PRINT "Fr
     ame":V1 = 1:V2 = 7:H1 = 25:H2 = 40
390  GOTO 410
400  VTAB 12: HTAB 27: POKE 32,26: PRINT "No"
     : PRINT "Yes":V1 = 12:V2 = 13:H1 = 25:H2
      = 31
410  TEXT :C = 0: IF CM = 0 THEN C = CC
420  VTAB V1 + C: HTAB H1: INVERSE : PRINT "
     ";: NORMAL : PRINT ">";: HTAB H2 - 1: PRINT
     "<";: INVERSE : PRINT " ";: NORMAL :C1 =
     C
430  GOSUB 170: IF G$ = "A" OR G$ = AL$ OR G$
      = AU$ THEN C1 = C - 1
440  IF G$ = "Z" OR G$ = AR$ OR G$ = AD$ THEN
     C1 = C + 1
450  IF C1 < 0 THEN C1 = V2 - V1
460  IF C1 > V2 - V1 THEN C1 = 0
470  IF G$ = CR$ THEN 500
480  VTAB V1 + C: HTAB H1: PRINT "  ";: HTAB
     H2 - 1: PRINT "  ";
490 C = C1: GOTO 420
500  GOSUB 120: GOSUB 290: IF CM = 0 THEN CC =
     C
510  RETURN
520  REM ** Global Error Handler
530  TEXT : HOME :I =  PEEK (222):A$ = "": IF
     I = 4 THEN A$ = "Disk is Write Protected
     "
540  IF I = 8 THEN A$ = "I/O ERROR on this di
     sk"
550  IF I = 9 THEN A$ = "Sorry, DISK FULL err
     or.  Use another."
560  IF I = 13 THEN A$ = "Must be a binary fi
     le"
570  IF I = 6 THEN A$ = "File not on this dis
     k"
580  IF I = 10 THEN A$ = "File exists and is
     locked"
590  ON I = 16 AND PD = 0 GOTO 640: IF I = 16
      AND PD THEN A$ = "Bad pathname"
600  IF I = 17 THEN A$ = "ProDOS directory fu
     ll"
610  IF I = 255 THEN A$ = "Please do not pres
     s Control-C"
620  IF A$ = "" THEN A$ = "Error Encountered.
     ..Please retry"
630 A = 5: GOSUB 200: GOTO 1280
640  VTAB 5: PRINT "An error is present in": PRINT
     : HTAB 10: PRINT "LINE # " PEEK (218) +
      PEEK (219) * 256: PRINT : PRINT "Please
      correct, save the new version,": PRINT
     : PRINT "and rerun WINDOWORKS": END
650  REM ** Speed Setting Routine
660  VTAB 18: HTAB 3: PRINT "Slow --:--:--:--
     :--:--:--:--:-- Fast";: FOR E = 1 TO 8: VTAB
     19: HTAB 7 + E * 3: PRINT E;: NEXT :C =
     B(5) * (M <  > K)
670 A = 23:A$ = "Use Arrows, A & Z, or (1-8)
     to select": GOSUB 320:A = 24:A$ = "Press
      Return when satisfied": GOSUB 320
680  INVERSE : VTAB 18: HTAB 30 - (C * 3): PRINT
     "-:-";: NORMAL : GOSUB 140:P = 0: IF G$ =
     "A" OR G$ = AL$ OR G$ = AU$ THEN P = 1
```

Listing 1 for WindoWorks

WINDOWORKS *(continued)*

```
690  IF G$ = ES$ THEN C = 255: RETURN
700  IF G$ = "Z" OR G$ = AR$ OR G$ = AD$ THEN
     P =  - 1
710  IF G$ = CR$ THEN  RETURN
720  IF G$ >  = "1" AND G$ <  = "8" THEN P =
     (8 -  VAL (G$)) - C
730  VTAB 18: HTAB 30 - (C * 3): PRINT "-:-";
     :C = C + P: IF C > 7 OR C < 0 THEN C = C
      - P
740  GOTO 680
750  REM ** Create A Window
760  GOSUB 130: IF L = 1 THEN  CALL A(6)
770  IF M = K THEN  POKE A(12),65: POKE A(13)
     ,130: POKE A(14),13: POKE A(15),27: GOTO
     790
780  FOR E = 0 TO 3: POKE A(E + 12), PEEK (A(
     17) + M * 7 + E): NEXT : FOR E = 0 TO 5:
     B(E) =  PEEK (A(17) + M * 7 + E): NEXT
790  CALL A(0): TEXT : IF  PEEK (A(12)) = 255
      THEN WF = 255: RETURN
800 A = 11:B = 25:CM = 1:G = B(4) * (M <  > K
     ): GOSUB 330: IF C = 255 THEN WF = C
810 F = A(17) + M * 7: FOR E = 0 TO 3: POKE F
      + E, PEEK (A(12 + E)): NEXT : POKE F +
     4,C: GOSUB 660: POKE F + 5,C: IF C = 255
      THEN WF = C
820  RETURN
830  FOR E = 0 TO 5: POKE F + E,B(E): NEXT : GOTO
     1280
840  GOSUB 130:M = 0
850  FOR E = 0 TO 3: POKE A(E + 12), PEEK (A(
     17) + M * 7 + E): NEXT
860  CALL A(9): FOR E = 1 TO 20: NEXT : CALL
     A(9): IF  PEEK (49152) < 128 THEN 860
870  GET G$: GOSUB 290: IF (G$ = AL$ OR G$ =
     "A" OR G$ = AU$) AND M <  > 0 THEN M = M
      - 1
880  IF (G$ = "Z" OR G$ = AR$ OR G$ = AD$) AND
     (M + 1) <  > K THEN M = M + 1
890  IF G$ = ES$ THEN M = 255: RETURN
900  IF G$ <  > CR$ THEN 850
910  RETURN
920  REM ** Input a String Routine
930 A = 17:A$ = "Use <- for erase, Control-X
     to clear": GOSUB 320
940 A = 19:A$ = "Press Return when finished":
      GOSUB 320: VTAB 22: HTAB 6: FOR E = 1 TO
     15 + Z * 15: PRINT "-";: NEXT : GOSUB 31
     0
950  VTAB 21: HTAB 6: PRINT C$;: FOR E =  LEN
     (C$) TO 15 + Z * 15: PRINT " ";: NEXT : HTAB
     6 +  LEN (C$): INVERSE : PRINT " ";: NORMAL

960  GOSUB 170: ON G$ < " " OR G$ =  CHR$ (12
     7) GOTO 980: IF PD = 1 THEN  ON G$ < "."
      OR G$ > "Z" OR (G$ > "9" AND G$ < "A") OR
      LEN (C$) > E - 1 GOTO 960
970 C$ = C$ + G$: VTAB 21: HTAB 5 +  LEN (C$)
     : PRINT G$;: INVERSE : PRINT " ";: NORMAL
     : GOTO 960
980  IF (G$ = AL$ OR G$ =  CHR$ (127)) AND ( LEN
     (C$) > 1) THEN C$ =  LEFT$ (C$, LEN (C$)
      - 1): GOTO 950
990  IF G$ = ES$ THEN C$ = "": RETURN
1000  IF G$ = AL$ OR G$ =  CHR$ (127) OR G$ =
      CHR$ (24) THEN C$ = "": GOTO 950
1010  IF G$ <  > CR$ THEN 960
1020  POKE A(17) - 1,K: VTAB 21: HTAB 6: PRINT
     C$" ";: RETURN
1030  REM ** Modify Data Disk Loc.
1040  GOSUB 40: IF PD THEN 1090
1050  VTAB 24: HTAB 6: GET C$: ON C$ = CR$ GOTO
     1070:C =  VAL (C$): IF C < 1 OR C > 7 THEN
     1050
1060  PRINT C$;:DS = C
1070  HTAB 15: GET C$: ON C$ = CR$ GOTO 1280:
     C =  VAL (C$): IF C < 1 OR C > 4 THEN 10
     70
1080  PRINT C$;:DD = C: GOTO 1280
1090  VTAB 23: HTAB 1: PRINT "Enter new ProDO
     S prefix (max. 30 chars.)";:C$ = "/":Z =
     1: GOSUB 930:Z = 0: IF  LEN (C$) < 2 THEN
     1280
1100  IF  LEFT$ (C$,1) <  > "/" THEN C$ = "/"
      + C$
1110  IF  RIGHT$ (C$,1) <  > "/" THEN C$ = C$
      + "/"
1120  PRINT D$"PREFIX"C$:DP$ = C$: GOTO 1280
1130  REM ** Catalog Data Drive
1140  HOME :H$ = "CAT": IF PD = 0 THEN H$ = H
     $ + "ALOG,S" +  STR$ (DS) + ",D" +  STR$
     (DD) + ",V" +  STR$ (DV)
1150  GOSUB 240: FOR C = 1 TO 38: PRINT "-";:
      NEXT : GOSUB 140: PRINT : GOTO 1280
1160 DS = 6:DD = 1:D$ =  CHR$ (4):D = 49232: DIM
     A(18):B$(0) = "OFF":B$(1) = "ON":E$(0) =
     "NO":E$(1) = "YES":F$ = "-PRESS RETURN-"
1170 AL$ =  CHR$ (8):AR$ =  CHR$ (21):AU$ =  CHR$
     (11):AD$ =  CHR$ (10):CR$ =  CHR$ (13):E
     S$ =  CHR$ (27)
1180  GOSUB 250: ONERR  GOTO 530
1190  FOR E = 0 TO 18:A(E) = ( PEEK (24576 +
     E * 2) + 256 *  PEEK (24577 + E * 2)): NEXT
     : CALL A(8): CALL A(18): REM ** Relocate
      RUNTIME code
1200  REM ** Title Page
1210 A = 3:A$ = "WINDOWORKS": GOSUB 320:A = 5
     :A$ = "BY BOB THRASHER": GOSUB 320:A = 9
     :A$ = "COPYRIGHT (C) 1987": GOSUB 320:A =
     11:A$ = "BY MICROSPARC, INC.": GOSUB 320
1220  IF  PEEK ( - 1101) = 6 THEN A = 18:A$ =
     " ": GOSUB 200: GOTO 1260
1230 A = 20:A$ = "PLEASE TYPE ONLY IN UPPER-C
     ASE": GOSUB 320: PRINT : PRINT : PRINT "
       CAN YOU DISPLAY LOWER-CASE (Y/N)?";: FLASH
     : PRINT " ";: NORMAL : GOSUB 310
1240  GOSUB 140: IF G$ = "N" THEN  CALL A(4):
      GOTO 1260
1250  IF G$ <  > "Y" THEN 1240
1260 G = 0:J = 0:K = 0:L = 0: POKE A(17),255
1270  REM ** Main Menu
1280  ONERR  GOTO 530
1290  TEXT : HOME :A = 16:B = 5:CM = 0: VTAB
     16: HTAB 25: PRINT "WINDOWS:";K: HTAB 25
     : PRINT "OVERLAY:";B$(L): HTAB 25: PRINT
     "PICTURE:";E$(J): GOSUB 40
1300  GOSUB 330: ON C + 1 GOTO 1330,1370,1400
     ,1450,1490,1540,1580,1640,1710,1750,1040
     ,1140,1800,1820
1310  GOTO 1300
1320  REM ** View Hi-Res Screen
1330  IF J = 0 THEN 220
1340  GOSUB 130: IF L = 1 THEN  CALL A(6)
1350  GOSUB 140: GOTO 1280
1360  REM ** View Animation
1370  IF K = 0 THEN 230
1380  FOR E = 0 TO 49: POKE A(17) + E * 7 + 6
     ,0: NEXT : GOSUB 130: CALL A(10): GOTO 1
     280
1390  REM ** Add Window
1400  IF J = 0 THEN 220
1410  IF K = 50 THEN A$ = "Limit of 50 window
     s":A = 16: GOSUB 300: GOSUB 200: GOTO 12
     80
1420 M = K:WF = 0: GOSUB 760: IF WF = 255 THEN
     830
1430 K = K + 1: POKE F + 7,255: GOTO 1280
1440  REM ** Delete Window
1450  IF K = 0 THEN 230
1460  GOSUB 840: IF M = 255 THEN 1280
1470  FOR E = A(17) + M * 7 TO A(17) + (49 *
     7): POKE E, PEEK (E + 7): NEXT :K = K -
     1: GOTO 1280
1480  REM ** Edit Window
1490  IF K = 0 THEN 230
1500  GOSUB 840: IF M = 255 THEN 1280
1510 WF = 0: GOSUB 760: IF WF = 255 THEN 830
1520  GOTO 1280
1530  REM ** Delete All Windows
1540  IF K = 0 THEN 230
1550  GOSUB 210: IF C = 1 THEN K = 0: POKE A(
     17),255
1560  GOTO 1280
1570  REM ** Load Picture
1580 A = 16:A$ = "Enter filename of picture t
     o load:": GOSUB 320:C$ = "": GOSUB 930: IF
     C$ = "" THEN 1280
1590 H$ = C$: GOSUB 80:H$ = "VERIFY" + H$: GOSUB
     240:FS =  PEEK (38941) +  PEEK (38942) *
     256:FT =  PEEK (38949): IF (FT <  > 4 AND
     FT <  > 132) * Z THEN 1620
```

```
1600  IF (FS < 33 OR FS > 34) * Z THEN 1620
1610  HGR2 :H$ = "BLOAD" +  RIGHT$ (H$, LEN (
      H$) - 6) + ",A$4000": GOSUB 240: GOSUB 2
      90: CALL A(2): GOSUB 290:H = ( PEEK (A(1
      6)) +  PEEK (A(16) + 1) * 256) - 3: POKE
      H,76: POKE H + 1,96: POKE H + 2,143: GOSUB
      140:J = 1: GOTO 1280
1620  POKE 32,24: POKE 33,15: HOME : TEXT : VTAB
      10: HTAB 25: PRINT "Please select a";: VTAB
      11: HTAB 25: PRINT "33 or 34 sector";: VTAB
      12: HTAB 25: PRINT "binary file.";: VTAB
      14: HTAB 25: PRINT F$;: GOSUB 300: GOSUB
      140: GOTO 1280
1630  REM ** Load Window File
1640  IF J = 0 THEN 220
1650 A = 16:A$ = "Enter filename of window da
      ta to load:": GOSUB 320:C$ = "W.": GOSUB
      930: IF  LEN (C$) < 3 THEN 1280
1660 H$ = C$: GOSUB 80:H$ = "VERIFY" + H$: GOSUB
      240:FS =  PEEK (38941) +  PEEK (38942) *
      256:FT =  PEEK (38949): IF (FT <  > 4 AND
      FT <  > 132) * Z THEN 1620
1670  IF (FS <  > 3) * Z THEN 1690
1680 H$ = "BLOAD" +  RIGHT$ (H$, LEN (H$) - 6
      ) + ",A" +  STR$ (A(17) - 1): GOSUB 240:
      K =  PEEK (A(17) - 1): GOTO 1280
1690  POKE 32,24: POKE 33,15: HOME : TEXT : VTAB
      10: HTAB 25: PRINT "Please select a";: VTAB
      11: HTAB 25: PRINT "3 sector binary";: VTAB
      12: HTAB 25: PRINT "window file.";: VTAB
      14: HTAB 25: PRINT F$;: GOSUB 300: GOSUB
      140: GOTO 1280
1700  REM ** Save Window File
1710  IF K = 0 THEN 230
1720 C$ = "W.":A = 16:A$ = "Enter filename fo
      r window data:": GOSUB 320: GOSUB 930: IF
       LEN (C$) < 3 THEN 1280
1730 H$ = C$: GOSUB 80:H$ = "BSAVE" + H$ + ",
      A" +  STR$ (A(17) - 1) + ",L351": GOSUB
      240: GOTO 1280
1740  REM ** Create Final Module
1750  IF J = 0 THEN 220
1760  IF K = 0 THEN 230
1770 C$ = "F.":A = 16:A$ = "Enter filename fo
      r final module:": GOSUB 320: GOSUB 930: IF
       LEN (C$) < 3 THEN 1280
1780 H$ = C$: GOSUB 80:H$ = "BSAVE" + H$ + ",
      A" +  STR$ (H) + ",L" +  STR$ (38400 - H
      ): GOSUB 240: GOTO 1280
1790  REM ** Toggle Overlay
1800 L = 1 - L:A = 16:A$ = "The Window Overla
      y Option is now " + B$(L): GOSUB 200: GOTO
       1280
1810  REM ** Exit Program
1820  GOSUB 210: IF C <  > 1 THEN 1280
1830  HOME : END
```

END OF LISTING 1

```
KEY PERFECT 5.0
RUN ON
WINDOWORKS
=======================================
   CODE-5.0   LINE# - LINE#    CODE-4.0
   --------   -------------    --------
   CA5443F4      1 -    30       6A07
   702F8036     40 -   130       BBE0
   3EBD3475    140 -   230       7EFB
   6AE8DFD8    240 -   330       AFA4
   0C32C4E3    340 -   430      011DCC
   56726B94    440 -   530       6457
   D6195C55    540 -   630       AC31
   BE605BAF    640 -   730       FA4D
   03A79335    740 -   830       A348
   6C99173C    840 -   930       901F
   4259F07A    940 -  1030       D45F
   98CCD2E0   1040 -  1130       998A
   B341E1BC   1140 -  1230      013D05
   817A7D8A   1240 -  1330       9D34
   8D8E8E5F   1340 -  1430       70DF
   327EE195   1440 -  1530       59CB
   5F72C01A   1540 -  1630       FA95
   5B08362C   1640 -  1730      0108A4
   26A7B413   1740 -  1830       99A9
   2C76C489 = PROGRAM TOTAL =    1C34
```

Listing 2 for WindoWorks

WINDOWORKS.ML

```
                        1   ********************
                        2   *  WINDOWORKS.ML   *
                        3   *  by  Bob Thrasher  *
                        4   * Copyright (C) 1987 *
                        5   * by MicroSPARC, Inc *
                        6   * Concord, MA  01742 *
                        7   ********************
                        8
                        9   ********************
                       10   *   Created with    *
                       11   *  128K  Merlin Pro  *
                       12   ********************
                       13
                       14            ORG   $6000      ;Place it up high out of the way
                       15
                       16   ArowLeft EQU   $88
                       17   ArowRigh EQU   $95
                       18   ArowUp   EQU   $8B
                       19   ArowDown EQU   $8A
                       20   Keyboard EQU   $C000
                       21   Strobe   EQU   $C010
                       22   RomDelay EQU   $FCA8
                       23
                       24   * This is the global jump table, which provides a central
                       25   * location to get to any routine in the program.  This is
                       26   * handy for making changes to the routines without needing
                       27   * to recalculate the entry points for everything.
                       28
6000: 29 60            29            DA    Shape
6002: 26 60            30            DA    AudioPit
6004: 32 61            31            DA    Pack
6006: 27 60            32            DA    AudioDur
6008: 20 63            33            DA    Filter
600A: 99 93            34            DA    Unpack
600C: D7 62            35            DA    Overlay
600E: 05 63            36            DA    Audio
6010: AE 62            37            DA    Reloc
6012: 20 93            38            DA    Frame
6014: 60 8F            39            DA    Control
6016: 28 60            40            DA    GetError
6018: 1C 93            41            DA    WTop
601A: 1D 93            42            DA    WBottom
601C: 1E 93            43            DA    WLeft
601E: 1F 93            44            DA    WRight
6020: 97 93            45            DA    LocPack
6022: E9 8F            46            DA    Data
6024: 71 93            47            DA    YGen
                       48
6026: 00               49   AudioPit DFB   0           ;Pitch for Audio routine
6027: 00               50   AudioDur DFB   0           ;Duration for Audio routine
6028: 00               51   GetError DFB   0           ;Which DOS error (if any)
                       52
                       53   * Shape - This routine allows the user to define the size of
                       54   *         the curr. active frame at WTop, WBottom, WLeft,
                       55   *         and WRight.  Use of all arrow keys, A, Z are
                       56   *         interpreted.  Space bar alternates control from
                       57   *         one corner to the other.  Return signals the end.
                       58
6029: 20 15 94         59   Shape    JSR   Savereg     ;Save everything as always
602C: 20 E7 60         60   Shape1   JSR   Shape9     ;This is upper left; Get a key
602F: C9 9B            61            CMP   #$9B       ;ESC?
6031: F0 50            62            BEQ   Shape4
6033: C9 A0            63            CMP   #$A0       ;If Space then go to Shape3
6035: F0 55            64            BEQ   Shape3
6037: C9 8D            65            CMP   #$8D
6039: F0 4D            66            BEQ   Shape2     ;Return is pressed, exit
603B: C9 8B            67            CMP   #ArowUp    ;If not Up Arrow then goto :1
603D: D0 0B            68            BNE   :1
603F: AD 1C 93         69            LDA   WTop       ;Move Top up
6042: F0 E8            70            BEQ   Shape1
6044: CE 1C 93         71            DEC   WTop
6047: 4C 2C 60         72            JMP   Shape1
604A: C9 8A            73   :1       CMP   #ArowDown  ;If not Down Arrow then goto :2
604C: D0 11            74            BNE   :2
604E: AD 1C 93         75            LDA   WTop       ;Move down
6051: 18               76            CLC
6052: 69 01            77            ADC   #$01
6054: CD 1D 93         78            CMP   WBottom
6057: F0 D3            79            BEQ   Shape1
6059: 8D 1C 93         80            STA   WTop
605C: 4C 2C 60         81            JMP   Shape1
605F: C9 88            82   :2       CMP   #ArowLeft  ;If not Left Arrow then goto :3
6061: D0 0B            83            BNE   :3
6063: AD 1E 93         84            LDA   WLeft      ;Move left side left
6066: F0 C4            85            BEQ   Shape1
6068: CE 1E 93         86            DEC   WLeft
606B: 4C 2C 60         87            JMP   Shape1
606E: C9 95            88   :3       CMP   #ArowRigh  ;If not Right Arrow then ignore
6070: D0 BA            89            BNE   Shape1
6072: AD 1E 93         90            LDA   WLeft      ;Move left side right
6075: 18               91            CLC
6076: 69 01            92            ADC   #$01
6078: CD 1F 93         93            CMP   WRight
607B: F0 AF            94            BEQ   Shape1
607D: 8D 1E 93         95            STA   WLeft
6080: 4C 2C 60         96            JMP   Shape1
6083: A9 FF            97   Shape4   LDA   #$FF       ;Escape abort function
6085: 8D 1C 93         98            STA   WTop
6088: 20 42 94         99   Shape2   JSR   Restreg    ;Restore everything and exit
608B: 60              100            RTS
608C: 20 E7 60        101   Shape3   JSR   Shape9     ;lower right; Get a key
608F: C9 9B           102            CMP   #$9B       ;Escape?
6091: F0 F0           103            BEQ   Shape4
6093: C9 A0           104            CMP   #$A0       ;Space then go to Shape1
6095: F0 95           105            BEQ   Shape1
6097: C9 8D           106            CMP   #$8D       ;If Return then exit
6099: F0 ED           107            BEQ   Shape2
609B: C9 8B           108            CMP   #ArowUp
609D: D0 11           109            BNE   :1
609F: AD 1D 93        110            LDA   WBottom    ;Move bottom up
60A2: 38              111            SEC
60A3: E9 01           112            SBC   #$01
60A5: CD 1C 93        113            CMP   WTop
```

Listing 2 for WindoWorks

WINDOWORKS.ML *(continued)*

```
60A8: F0 E2     114            BEQ   Shape3
60AA: 8D 1D 93  115            STA   WBottom
60AD: 4C 8C 60  116            JMP   Shape3
60B0: C9 8A     117   :1       CMP   #ArowDown
60B2: D0 0D     118            BNE   :2
60B4: AD 1D 93  119            LDA   WBottom     ;Move Bottom DOWN
60B7: C9 BF     120            CMP   #$BF
60B9: F0 01     121            BEQ   Shape3
60BB: EE 1D 93  122            INC   WBottom
60BE: 4C 8C 60  123            JMP   Shape3
60C1: C9 88     124   :2       CMP   #ArowLeft
60C3: D0 11     125            BNE   :3
60C5: AD 1F 93  126            LDA   WRight      ;Move Right Left Arrow
60C8: 38        127            SEC
60C9: E9 01     128            SBC   #$01
60CB: CD 1E 93  129            CMP   WLeft
60CE: F0 BC     130            BEQ   Shape3
60D0: 8D 1F 93  131            STA   WRight
60D3: 4C 8C 60  132            JMP   Shape3
60D6: C9 95     133   :3       CMP   #ArowRigh
60D8: D0 B2     134            BNE   Shape3
60DA: AD 1F 93  135            LDA   WRight      ;Move Right RIGHT
60DD: C9 27     136            CMP   #39
60DF: F0 AB     137            BEQ   Shape3
60E1: EE 1F 93  138            INC   WRight
60E4: 4C 8C 60  139            JMP   Shape3
                140   ;-------------------------------------------------
60E7: 20 20 93  141   Shape9   JSR   Frame       ;Display the frame
60EA: A9 01     142            LDA   #$01        ;A short delay
60EC: 20 A8 FC  143            JSR   RomDelay
60EF: 20 20 93  144            JSR   Frame       ;Erase frame
60F2: A9 01     145            LDA   #$01        ;Another delay
60F4: 20 A8 FC  146            JSR   RomDelay
60F7: 20 FD 60  147            JSR   GetDir      ;Get a direction key
60FA: 90 EB     148            BCC   Shape9      ;None pressed
60FC: 60        149            RTS
                150
                151   ; GetDir   Scans the keyboard quickly for either
                152   ;          one of the four arrows or the A & Z keys for
                153   ;          cursor movement, and also notices the Space and
                154   ;          Return key.  The A and Z keys are converted
                155   ;          to ArowUp & ArowDown for processing.  If none of
                156   ;          these was pressed then carry is clear upon exit
                157
60FD: AD 00 C0  158   GetDir   LDA   Keyboard
6100: C9 9B     159            CMP   #$9B
6102: F0 29     160            BEQ   :3
6104: C9 8D     161            CMP   #$8D
6106: F0 25     162            BEQ   :3
6108: C9 A0     163            CMP   #$A0
610A: F0 21     164            BEQ   :3
610C: C9 88     165            CMP   #ArowLeft
610E: F0 1D     166            BEQ   :3
6110: C9 95     167            CMP   #ArowRigh
6112: F0 19     168            BEQ   :3
6114: C9 8B     169            CMP   #ArowUp
6116: F0 15     170            BEQ   :3
6118: C9 8A     171            CMP   #ArowDown
611A: F0 11     172            BEQ   :3
611C: C9 C1     173            CMP   #"A"
611E: D0 05     174            BNE   :1
6120: A9 8B     175            LDA   #ArowUp
6122: 4C 2D 61  176            JMP   :3
6125: C9 DA     177   :1       CMP   #"Z"
6127: F0 02     178            BEQ   :2
6129: 18        179            CLC
612A: 60        180            RTS
612B: A9 8A     181   :2       LDA   #ArowDown
612D: 8D 10 C0  182   :3       STA   Strobe
6130: 38        183            SEC
6131: 60        184            RTS
                185
                186   ; Pack    This routine packs Hi Res page 1 into a data
                187   ;         buffer defined at LocPack.  Accompanying Unpack
                188   ;         routine will reverse the process.
                189   ;         The coded data follows this format:
                190   ;            $00-$7F = This # of repeated bytes followed
                191   ;                      by the actual byte
                192   ;                      Example:  05 AA = AA AA AA AA AA AA
                193   ;            $80-$FF = This # of unique bytes followed by
                194   ;                      the unique data (high bit is to be
                195   ;                      removed first, leaving range $00-$7F)
                196   ;                      Example:  82 AE B0 42 = AE B0 42
                197
                198   PackPtr1 EQU   $00          ;Points to screen mem (not dest.)
                199   PackPtr2 EQU   $02          ;Pts to PackBuffr area (not dest.)
                200   PackPtr3 EQU   $04          ;Pts to curr counter for pass (")
                201   PackBuffr EQU  $6A00        ;Initial storage for packed pic
                202
6132: 20 15 94  203   Pack     JSR   Savereg      ;Save registers first
6135: A9 00     204            LDA   #$00         ;Reset some variables
6137: 8D 69 94  205            STA   Subtemp1     ;Vertical
613A: 8D 69 94  206            STA   Subtemp2     ;Horizontal
613D: A9 00     207            LDA   #<PackBuffr  ;Set PackPtr2 equal to PackBufr
613F: 85 02     208            STA   PackPtr2
6141: A9 6A     209            LDA   #>PackBuffr
6143: 85 03     210            STA   PackPtr2+1
6145: 20 7E 62  211            JSR   Pack7        ;Setup artificial stack
6148: 20 7E 62  212            JSR   Pack7
614B: 20 7E 62  213   Pack0    JSR   Pack7
614E: AD 6A 94  214            LDA   Subtemp3     ;Check if all same, or if unique
6151: CD 6B 94  215            CMP   Subtemp4
6154: D0 05     216            BNE   Pack1
6156: C0 6C 94  217            CMP   Subtemp5
6159: F0 64     218            BEQ   Pack2
615B: A5 02     219   Pack1    LDA   PackPtr2     ;Save PackPtr2 into PackPtr3
615D: 85 04     220            STA   PackPtr3
615F: A5 03     221            LDA   PackPtr2+1
6161: 85 05     222            STA   PackPtr3+1
6163: A9 02     223            LDA   #$02         ;Start counter @ $02
6165: 20 73 62  224            JSR   Pack6
6168: AD 6A 94  225            LDA   Subtemp3
616B: 20 73 62  226            JSR   Pack6        ;Transfer stack into mem
616E: AD 6B 94  227            LDA   Subtemp4
6171: 20 73 62  228            JSR   Pack6
6174: AD 6C 94  229            LDA   Subtemp5
6177: 20 73 62  230            JSR   Pack6
617A: 20 7E 62  231            JSR   Pack7        ;Move stack down by 3
617D: 20 7E 62  232            JSR   Pack7
6180: 20 7E 62  233            JSR   Pack7
6183: 20 91 62  234   :1       JSR   Pack8        ;Finished with screen?
6186: B0 79     235            BCS   Pack3        ;Yes
6188: AD 6A 94  236            LDA   Subtemp3     ;See if stack is unique or sam
618B: CD 6B 94  237            CMP   Subtemp4
618E: D0 05     238            BNE   :2
6190: CD 6C 94  239            CMP   Subtemp5
6193: F0 2A     240            BEQ   Pack2        ;Go to repeating sequence
6195: A0 00     241   :2       LDY   #$00         ;Increment counter @ PackPtr3
6197: B1 04     242            LDA   (PackPtr3),Y
6199: 18        243            CLC
619A: 69 01     244            ADC   #$01
619C: 91 04     245            STA   (PackPtr3),Y
619E: C9 80     246            CMP   #$80         ;Reached limit?
61A0: D0 11     247            BNE   :3           ;Nope
61A2: A9 7F     248            LDA   #$7F         ;Yes, set to limit of $7F
61A4: 91 04     249            STA   (PackPtr3),Y
61A6: A5 02     250            LDA   PackPtr2     ;Reset PackPtr3
61A8: 85 04     251            STA   PackPtr3
61AA: A5 03     252            LDA   PackPtr2+1
61AC: 85 05     253            STA   PackPtr3+1
61AE: A9 00     254            LDA   #$00         ;New counter of $00
61B0: 20 73 62  255            JSR   Pack6
61B3: AD 6A 94  256   :3       LDA   Subtemp3
61B6: 20 73 62  257            JSR   Pack6
61B9: 20 7E 62  258            JSR   Pack7        ;Update stack
61BC: 4C 83 61  259            JMP   :1
61BF: A2 82     260   Pack2    LDX   #$82         ;Start of rep seq, start count
61C1: AD 6A 94  261            LDA   Subtemp3
61C4: 8D 6D 94  262            STA   Subtemp6
61C7: 20 7E 62  263            JSR   Pack7        ;New stack
61CA: 20 7F 62  264            JSR   Pack7
61CD: 20 7E 62  265   :1       JSR   Pack7
61D0: 20 91 62  266            JSR   Pack8
61D3: B0 1A     267            BCS   :2
61D5: AD 6A 94  268            LDA   Subtemp3     ;See if byte 1 in stack is sam
61D8: CD 6D 94  269            CMP   Subtemp6
61DB: D0 12     270            BNE   :2
61DD: E8        271            INX                ;Same, increment counter
61DE: E0 FF     272            CPX   #$FF         ;Limit?
61E0: D0 EB     273            BNE   :1           ;Nope
61E2: 8A        274            TXA
61E3: 20 73 62  275            JSR   Pack6        ;Dump counter to table
61E6: AD 6D 94  276            LDA   Subtemp6
61E9: 20 73 62  277            JSR   Pack6        ;And value for repeat also
61EC: 4C 4B 61  278            JMP   Pack0        ;Start a new run
61EF: 8A        279   :2       TXA
61F0: 20 73 62  280            JSR   Pack6        ;Dump counter to table
61F3: AD 6D 94  281            LDA   Subtemp6     ;And value also
61F6: 20 73 62  282            JSR   Pack6
61F9: 20 91 62  283            JSR   Pack8
61FC: B0 03     284            BCS   Pack3
61FE: 4C 5B 61  285            JMP   Pack1        ;Start of unique sequence
6201: A9 5F     286   Pack3    LDA   #<Control-1
6203: 85 00     287            STA   PackPtr1
6205: A9 BF     288            LDA   #>Control-1
6207: 85 01     289            STA   PackPtr1+1
6209: A0 00     290            LDY   #$00         ;Mv packed pic up against rest
620B: B1 02     291   :1       LDA   (PackPtr2),Y
620D: 91 00     292            STA   (PackPtr1),Y
620F: A5 00     293            LDA   PackPtr1
6211: 38        294            SEC
6212: E9 01     295            SBC   #$01
6214: 85 00     296            STA   PackPtr1
6216: A5 01     297            LDA   PackPtr1+1
6218: E9 00     298            SBC   #$00
621A: 85 01     299            STA   PackPtr1+1
621C: A5 02     300            LDA   PackPtr2
621E: 38        301            SEC
621F: E9 01     302            SBC   #$01
6221: 85 02     303            STA   PackPtr2
6223: A5 03     304            LDA   PackPtr2+1
6225: E9 00     305            SBC   #$00
6227: 85 03     306            STA   PackPtr2+1
6229: C9 6A     307            CMP   #>PackBuffr
622B: D0 DE     308            BNE   :1
622D: A5 02     309            LDA   PackPtr2
622F: C9 00     310            CMP   #<PackBuffr
6231: D0 D8     311            BNE   :1
6233: B1 02     312            LDA   (PackPtr2),Y
6235: 91 00     313            STA   (PackPtr1),Y
6237: A5 00     314            LDA   PackPtr1
6239: 8D 97 93  315            STA   LocPack
623C: A5 01     316            LDA   PackPtr1+1
623E: 8D 98 93  317            STA   LocPack+1
6241: 20 42 94  318            JSR   Restreg
6244: 60        319            RTS
                320   ;-------------------------------------------------
6245: AC 68 94  321   Pack5    LDY   Subtemp1     ;Routine sets PackPtr1 to ver
6248: B9 32 95  322            LDA   YHiresL,Y    ;left of Hi Res line from te
624B: 85 00     323            STA   PackPtr1
624D: B9 72 04  324            LDA   YHiresH,Y
6250: 85 01     325            STA   PackPtr1+1
6252: AC 69 94  326            LDY   Subtemp2     ;Horizontal
6255: B1 00     327            LDA   (PackPtr1),Y ;Get the screen byte
6257: A8        328            TAY
6258: AD 68 94  329            LDA   Subtemp1
625B: 18        330            CLC
625C: 69 02     331            ADC   #$02
625E: C9 C0     332            CMP   #$C0
6260: 90 0C     333            BLT   :1
6262: 38        334            SEC
6263: E9 BF     335            SBC   #$BF
6265: C9 02     336            CMP   #$02
6267: D0 05     337            BNE   :1
6269: EE 69 94  338            INC   Subtemp2
626C: A9 00     339            LDA   #$00
626E: 8D 68 94  340   :1       STA   Subtemp1
6271: 98        341            TYA
6272: 60        342            RTS
```

```
                        343 *-------------------------------------------
8273: A0 00             344 Pack6   LDY   #$00         ;Put data byte into packed table
8275: 91 02             345         STA   (PackPtr2),Y
8277: E6 02             346         INC   PackPtr2
8279: D0 02             347         BNE   :1
827B: E6 03             348         INC   PackPtr2+1
827D: 60                349 :1      RTS
                        350 *-------------------------------------------
827E: AD 6B 94          351 Pack7   LDA   Subtemp4    ;Artificial stack handler
8281: 8D 6A 94          352         STA   Subtemp3
8284: AD 6C 94          353         LDA   Subtemp5
8287: 8D 6B 94          354         STA   Subtemp4
828A: 20 45 62          355         JSR   Pack5
828D: 8D 6C 94          356         STA   Subtemp5
8290: 60                357         RTS
                        358 *-------------------------------------------
8291: AD 69 94          359 Pack8   LDA   Subtemp2    ;Carry clear=more scr to process
8294: C9 28             360         CMP   #40         ;Carry set=finished
8296: F0 02             361         BEQ   :2
8298: 18                362 :1      CLC
8299: 60                363         RTS
829A: AD 68 94          364 :2      LDA   Subtemp1
829D: C9 06             365         CMP   #$06
829F: D0 F7             366         BNE   :1
82A1: A9 00             367         LDA   #$00
82A3: 8D 6A 94          368         STA   Subtemp3
82A6: 8D 6B 94          369         STA   Subtemp4
82A9: 8D 6C 94          370         STA   Subtemp5
82AC: 38                371         SEC
82AD: 60                372         RTS
                        373
                        374 * Reloc - This routine moves the RUNTIME module up to the
                        375 *         address where it belongs.
                        376
                        377 RelPtr1 EQU   $00         ;Index pointers
                        378 RelPtr2 EQU   $02
                        379
82AE: A9 47             380 Reloc   LDA   #<Start3    ;Define source location
82B0: 85 00             381         STA   RelPtr1
82B2: A9 63             382         LDA   #>Start3
82B4: 85 01             383         STA   RelPtr1+1
82B6: A9 60             384         LDA   #<Control
82B8: 85 02             385         STA   RelPtr2
82BA: A9 8F             386         LDA   #>Control
82BC: 85 03             387         STA   RelPtr2+1
82BE: A0 00             388         LDY   #$00
82C0: B1 00             389 :1      LDA   (RelPtr1),Y ;Get from source
82C2: 91 02             390         STA   (RelPtr2),Y ;Store in new address
82C4: E6 00             391         INC   RelPtr1
82C6: D0 02             392         BNE   :2
82C8: E6 01             393         INC   RelPtr1+1
82CA: E6 02             394 :2      INC   RelPtr2
82CC: D0 F2             395         BNE   :1
82CE: E6 03             396         INC   RelPtr2+1
82D0: A5 03             397         LDA   RelPtr2+1
82D2: C9 96             398         CMP   #$96        ;Finished?
82D4: D0 EA             399         BNE   :1
82D6: 60                400         RTS
                        401
                        402 * Overlay - This routine displays all windows by drawing
                        403 *           frames at each location.  Call again to remove.
                        404
                        405 CtrlPtr EQU   $00         ;Used for indexing
                        406
82D7: 20 15 94          407 Overlay JSR   Savereg     ;Save registers first
82DA: A9 E9             408         LDA   #<Data      ;Set up CtrlPtr for indexing
82DC: 85 00             409         STA   CtrlPtr
82DE: A9 8F             410         LDA   #>Data
82E0: 85 01             411         STA   CtrlPtr+1
82E2: A0 00             412         LDY   #$00        ;Start at first entry
82E4: A2 00             413 :1      LDX   #$00        ;Start temporary counter
82E6: B1 00             414 :2      LDA   (CtrlPtr),Y ;Transfer data
82E8: 9D 1C 93          415         STA   WTop,X
82EB: C8                416         INY
82EC: E8                417         INX
82ED: E0 04             418         CPX   #$04
82EF: D0 F5             419         BNE   :2
82F1: C8                420         INY               ;Move Y-Reg to next entry
82F2: C8                421         INY
82F3: C8                422         INY
82F4: AD 1C 93          423         LDA   WTop        ;See if no more
82F7: C9 FF             424         CMP   #$FF
82F9: F0 06             425         BEQ   :3
82FB: 20 20 93          426         JSR   Frame       ;Display this frame
82FE: 4C E4 62          427         JMP   :1
8301: 20 42 94          428 :3      JSR   Restreg     ;Restore registers and exit
8304: 60                429         RTS
                        430
                        431 * Audio - This routine simply produces a tone as defined in
                        432 *         AudioDur (duration) and AudioPit (pitch).
                        433
8305: 20 15 94          434 Audio   JSR   Savereg     ;Save registers as always
8308: AD 30 C0          435 :1      LDA   $C030       ;Make a small click
830B: 88                436 :2      DEY
830C: D0 05             437         BNE   :3
830E: CE 27 60          438         DEC   AudioDur    ;Finished with duration?
8311: F0 09             439         BEQ   :4
8313: CA                440 :3      DEX
8314: D0 F5             441         BNE   :2
8316: AE 26 60          442         LDX   AudioPit    ;Reset pitch for next iteration
8319: 4C 08 63          443         JMP   :1
831C: 20 42 94          444 :4      JSR   Restreg     ;Restore and exit
831F: 60                445         RTS
                        446
                        447 * Filter - This routine filters out all lower-case chars
                        448 *          & translates to upper-case for our II Plus
                        449 *          friends.
                        450
8320: A2 3A             451 Filter  LDX   #<:2        ;Set up intercept
8322: A0 83             452         LDY   #>:2
8324: AD 00 BF          453         LDA   $BF00       ;Check for ProDOS
8327: C9 4C             454         CMP   #$4C
8329: D0 07             455         BNE   :1          ;DOS 3.3
```

```
832B: 8E 30 BE          456         STX   $BE30       ;Global page output vector
832E: 8C 31 BE          457         STY   $BE31
8331: 60                458         RTS
8332: 86 36             459 :1      STX   $36
8334: 84 37             460         STY   $37
8336: 20 EA 03          461         JSR   $03EA       ;Let DOS know all is fine
8339: 60                462         RTS
833A: C9 E1             463 :2      CMP   #"a"        ;Lower-case A?
833C: 90 06             464         BLT   :3          ;Below that...all right
833E: C9 FB             465         CMP   #"z"+1      ;Above lower-case Z?
8340: B0 02             466         BGE   :3          ;Yes, no problem
8342: 29 DF             467         AND   #$DF        ;Convert to upper-case
8344: 4C F0 FD          468 :3      JMP   $FDF0       ;Continue on
                        469
                        470 Start3  EQU   *
                        471
                        472         ORG   $8F60       ;Runtime module execute location
                        473
                        474 * Control - This is the main routine handling code for the
                        475 *           runtime module.  It retrieves all coordinates
                        476 *           and other info from the Data table, sets up
                        477 *           any necessary calling parameters, then branches
                        478 *           around to the animation subroutines.  This
                        479 *           cycle continues until any key is pressed.
                        480
8F60: 20 15 94          481 Control JSR   Savereg     ;Save everything first
8F63: 2C 50 C0          482         BIT   $C050       ;Display Hi-Res page 2
8F66: 2C 57 C0          483         BIT   $C057
8F69: 2C 55 C0          484         BIT   $C055
8F6C: 2C 52 C0          485         BIT   $C052
8F6F: 20 71 93          486         JSR   YGen        ;Create Ytables first
8F72: 20 99 93          487         JSR   Unpack      ;Unravel the picture onto screen
8F75: A9 E9             488 :1      LDA   #<Data      ;Point CtrlPtr at the Data table
8F77: 85 00             489         STA   CtrlPtr
8F79: A9 8F             490         LDA   #>Data
8F7B: 85 01             491         STA   CtrlPtr+1
8F7D: A0 00             492 :2      LDY   #$00        ;Start at param #0 for this wind
8F7F: B1 00             493         LDA   (CtrlPtr),Y
8F81: C9 FF             494         CMP   #$FF        ;If $FF then there are no more
8F83: F0 57             495         BEQ   :6
8F85: 8D 1C 93          496         STA   WTop        ;Put data elements into position
8F88: C8                497         INY
8F89: B1 00             498         LDA   (CtrlPtr),Y
8F8B: 8D 1D 93          499         STA   WBottom
8F8E: C8                500         INY
8F8F: B1 00             501         LDA   (CtrlPtr),Y
8F91: 8D 1E 93          502         STA   WLeft
8F94: C8                503         INY
8F95: B1 00             504         LDA   (CtrlPtr),Y
8F97: 8D 1F 93          505         STA   WRight
8F9A: C8                506         INY
8F9B: B1 00             507         LDA   (CtrlPtr),Y
8F9D: 48                508         PHA               ;Type of animation is on stack
8F9E: C8                509         INY
8F9F: B1 00             510         LDA   (CtrlPtr),Y
8FA1: 8D 68 94          511         STA   Subtemp1    ;Speed of this window
8FA4: C8                512         INY
8FA5: B1 00             513         LDA   (CtrlPtr),Y
8FA7: CD 68 94          514         CMP   Subtemp1    ;Ready to activate?
8FAA: F0 09             515         BEQ   :3          ;Yes
8FAC: 18                516         CLC
8FAD: 69 01             517         ADC   #$01        ;Counter + 1
8FAF: 91 00             518         STA   (CtrlPtr),Y
8FB1: 68                519         PLA
8FB2: 4C CC 8F          520         JMP   :5
8FB5: A9 00             521 :3      LDA   #$00        ;Reset counter
8FB7: 91 00             522         STA   (CtrlPtr),Y
8FB9: 68                523         PLA
8FBA: 0A                524         ASL
8FBB: AA                525         TAX
8FBC: BD 48 91          526         LDA   JumpTab1,X  ;Find out where to go
8FBF: 8D CA 8F          527         STA   :4+1        ;And self-modify code
8FC2: E8                528         INX
8FC3: BD 48 91          529         LDA   JumpTab1,X
8FC6: 8D CB 8F          530         STA   :4+2
8FC9: 20 FF FF          531 :4      JSR   $FFFF       ;And do the animation
8FCC: A5 00             532 :5      LDA   CtrlPtr     ;Advance to next window
8FCE: 18                533         CLC
8FCF: 69 07             534         ADC   #$07
8FD1: 85 00             535         STA   CtrlPtr
8FD3: A5 01             536         LDA   CtrlPtr+1
8FD5: 69 00             537         ADC   #$00
8FD7: 85 01             538         STA   CtrlPtr+1
8FD9: 4C 7D 8F          539         JMP   :2          ;And keep going
8FDC: 2C 00 C0          540 :6      BIT   Keyboard    ;If no keypress then start over
8FDF: 10 94             541         BPL   :1
8FE1: 8D 10 C0          542         STA   Strobe
8FE4: 20 42 94          543         JSR   Restreg     ;Restore everything and exit
8FE7: 60                544         RTS
                        545
                        546 * Data is the list of user-defined windows.  Each entry is
                        547 * coded in the following manner:
                        548 *
                        549 *    DFB n1,n2,n3,n4,n5,n6,n7
                        550 *
                        551 *        n1 = Window top (0-190)
                        552 *        n2 = Window bottom (1-191)
                        553 *        n3 = Window left (0-38)
                        554 *        n4 = Window right (1-39)
                        555 *        n5 = Window type (0-6)
                        556 *        n6 = Window speed (0 fast to 9 slow)
                        557 *        n7 = Window counter (keeps track of speed)
                        558
8FE8: 00                559         HEX   00          ;Place holder for saving wind file
8FE9: 00 00 00  560 Data    DS    351         ;Limit of 50 windows +1 Extra
8FEC: 00 00 00 00 00 00 00 00 00
8FF4: 00 00 00 00 00 00 00 00 00
8FFC: 00 00 00 00 00 00 00 00 00
9004: 00 00 00 00 00 00 00 00 00
900C: 00 00 00 00 00 00 00 00 00
9014: 00 00 00 00 00 00 00 00 00
901C: 00 00 00 00 00 00 00 00 00
9024: 00 00 00 00 00 00 00 00 00
902C: 00 00 00 00 00 00 00 00 00
9034: 00 00 00 00 00 00 00 00 00
```

Listing 2 for WindoWorks

WINDOWORKS.ML

```
983C: 00 00 00 00 00 00 00 00
9844: 00 00 00 00 00 00 00 00
984C: 00 00 00 00 00 00 00 00
9854: 00 00 00 00 00 00 00 00
985C: 00 00 00 00 00 00 00 00
9864: 00 00 00 00 00 00 00 00
986C: 00 00 00 00 00 00 00 00
9874: 00 00 00 00 00 00 00 00
987C: 00 00 00 00 00 00 00 00
9884: 00 00 00 00 00 00 00 00
988C: 00 00 00 00 00 00 00 00
9894: 00 00 00 00 00 00 00 00
989C: 00 00 00 00 00 00 00 00
98A4: 00 00 00 00 00 00 00 00
98AC: 00 00 00 00 00 00 00 00
98B4: 00 00 00 00 00 00 00 00
98BC: 00 00 00 00 00 00 00 00
98C4: 00 00 00 00 00 00 00 00
98CC: 00 00 00 00 00 00 00 00
98D4: 00 00 00 00 00 00 00 00
98DC: 00 00 00 00 00 00 00 00
98E4: 00 00 00 00 00 00 00 00
98EC: 00 00 00 00 00 00 00 00
98F4: 00 00 00 00 00 00 00 00
98FC: 00 00 00 00 00 00 00 00
9904: 00 00 00 00 00 00 00 00
990C: 00 00 00 00 00 00 00 00
9914: 00 00 00 00 00 00 00 00
991C: 00 00 00 00 00 00 00 00
9924: 00 00 00 00 00 00 00 00
992C: 00 00 00 00 00 00 00 00
9934: 00 00 00 00 00 00 00 00
993C: 00 00 00 00 00 00 00 00
9944: 00 00 00 00
                      561
                      562  * JumpTabl is used to access the seven animation
                      563  * types without unnecessary code processing.
                      564
9948: CE 91           565  JumpTabl DA    ScrlUp
994A: 08 92           566           DA    ScrlDn
994C: 68 92           567           DA    ScrlLf
994E: C1 92           568           DA    ScrlRt
9950: 56 91           569           DA    Inverse
9952: 7E 91           570           DA    Color
9954: 20 93           571           DA    Frame
                      572
                      573  * Inverse - This routine performs the inv wind function.
                      574  *          The #Top, #Bottom, #Left and #Right values
                      575  *          must be set before entering and nothing is
                      576  *          saved upon exit.
                      577
                      578  InvrsPtr EQU   $02          ;Zero page pointer for indexing
                      579
9956: AE 1C 93        580  Inverse  LDX   #Top         ;Start at top
9959: BD 32 95        581  :1       LDA   YHiresL,X    ;Setup pointer for indexing
995C: 85 02           582           STA   InvrsPtr
995E: BD 72 94        583           LDA   YHiresH,X
9961: 85 03           584           STA   InvrsPtr+1
9963: AC 1F 93        585           LDY   #Right       ;Start at right and move left
9966: B1 02           586  :2       LDA   (InvrsPtr),Y ;Get the screen byte
9968: 49 7F           587           EOR   #$7F         ;Inverse the byte
996A: 91 02           588           STA   (InvrsPtr),Y ;Replace on screen
996C: CC 1E 93        589           CPY   #Left        ;Left side yet?
996F: F0 03           590           BEQ   :3           ;Yes
9971: 88              591           DEY
9972: 10 F2           592           BPL   :2
9974: EC 1D 93        593  :3       CPX   #Bottom      ;At bottom yet?
9977: F0 04           594           BEQ   :4           ;Yes
9979: E8              595           INX
997A: 4C 59 91        596           JMP   :1           ;Nope, keep going
997D: 60              597  :4       RTS
                      598
                      599  * Color - This routine merely toggles the high bit of all
                      600  *         bytes in the window to produce a color change
                      601  *         effect.
                      602
997E: AE 1C 93        603  Color    LDX   #Top         ;Start at top
9981: BD 32 95        604  :1       LDA   YHiresL,X    ;Setup pointer for indexing
9984: 85 02           605           STA   InvrsPtr
9986: BD 72 94        606           LDA   YHiresH,X
9989: 85 03           607           STA   InvrsPtr+1
998B: AC 1F 93        608           LDY   #Right       ;Start at right and move left
998E: B1 02           609  :2       LDA   (InvrsPtr),Y ;Get the screen byte
9990: 49 80           610           EOR   #$80         ;Toggle the high bit
9992: 91 02           611           STA   (InvrsPtr),Y ;Replace on screen
9994: CC 1E 93        612           CPY   #Left        ;Left side yet?
9997: F0 03           613           BEQ   :3           ;Yes
9999: 88              614           DEY
999A: 10 F2           615           BPL   :2
999C: EC 1D 93        616  :3       CPX   #Bottom      ;At bottom yet?
999F: F0 04           617           BEQ   :4           ;Yes
99A1: E8              618           INX
99A2: 4C 81 91        619           JMP   :1           ;Nope, keep going
99A5: 60              620  :4       RTS
                      621
                      622  * ScrlUp - This routine scrolls the window defined by #Top,
                      623  *          #Bottom, #Left, and #Right upw with wraparound.
                      624  *          Uses ScrlPtr1 and ScrlPtr2 but does not restore
                      625  *          their contents.
                      626
                      627  ScrlPtr1 EQU   $02          ;Used by all scroll routines
                      628  ScrlPtr2 EQU   $04          ;Used by all scroll routines
99A6: 00 00 00        629  ScrlBufr DFB   0,0,0,0,0,0,0,0 ;A 40 byte buffer
99A9: 00 00 00 00 00
99AE: 00 00 00  630           DFB   0,0,0,0,0,0,0,0 ; to save scroll lines
99B1: 00 00 00 00 00
99B6: 00 00 00  631           DFB   0,0,0,0,0,0,0,0
99B9: 00 00 00 00 00
99BE: 00 00 00  632           DFB   0,0,0,0,0,0,0,0
99C1: 00 00 00 00 00
99C6: 00 00 00  633           DFB   0,0,0,0,0,0,0,0
99C9: 00 00 00 00 00
                      634
99CE: AE 1C 93        635  ScrlUp   LDX   #Top         ;Start by saving top ln for later
99D1: 20 42 92        636           JSR   HorSub
99D4: E8              637  :1       INX                ;Move and set up next line
99D5: 20 5D 92        638           JSR   HorSubA
99D8: AC 1F 93        639           LDY   #Right       ;Move the entire ln down to (s
99DB: B1 04           640  :2       LDA   (ScrlPtr2),Y
99DD: 91 02           641           STA   (ScrlPtr1),Y
99DF: CC 1E 93        642           CPY   #Left
99E2: F0 03           643           BEQ   :3
99E4: 88              644           DEY
99E5: 10 F4           645           BPL   :2
99E7: EC 1D 93        646  :3       CPX   #Bottom      ;Finished?
99EA: F0 0B           647           BEQ   :4           ;Yes, go to :4
99EC: A5 04           648           LDA   ScrlPtr2     ;Transfer pointer and continue
99EE: 85 02           649           STA   ScrlPtr1
99F0: A5 05           650           LDA   ScrlPtr2+1
99F2: 85 03           651           STA   ScrlPtr1+1
99F4: 4C D4 91        652           JMP   :1
99F7: AC 1F 93        653  :4       LDY   #Right       ;Restore the line saved above
99FA: B9 A6 91        654  :5       LDA   ScrlBufr,Y
99FD: 91 04           655           STA   (ScrlPtr2),Y
99FF: CC 1E 93        656           CPY   #Left
9A02: F0 03           657           BEQ   :6
9A04: 88              658           DEY
9A05: 10 F3           659           BPL   :5
9A07: 60              660  :6       RTS                ;Finished with Up scroll
                      661  *-------------------------------------------------
                      662  * ScrlDn - Another scroll routine which uses all the
                      663  *          same variables and tables as defined in ScrlUp
                      664
9A08: AE 1D 93        665  ScrlDn   LDX   #Bottom      ;First save bottom ln for later
9A0B: 20 42 92        666           JSR   HorSub
9A0E: CA              667  :1       DEX                ;Move to next ln & set up for
move
9A0F: 20 5D 92        668           JSR   HorSubA
9A12: AC 1F 93        669           LDY   #Right       ;Start move from right side
9A15: B1 04           670  :2       LDA   (ScrlPtr2),Y
9A17: 91 02           671           STA   (ScrlPtr1),Y
9A19: CC 1E 93        672           CPY   #Left
9A1C: F0 03           673           BEQ   :3
9A1E: 88              674           DEY
9A1F: 10 F4           675           BPL   :2
9A21: EC 1C 93        676  :3       CPX   #Top
9A24: F0 0B           677           BEQ   :4           ;Finished?  yes, go to :4
9A26: A5 04           678           LDA   ScrlPtr2     ;Transfer pointer for next move
9A28: 85 02           679           STA   ScrlPtr1
9A2A: A5 05           680           LDA   ScrlPtr2+1
9A2C: 85 03           681           STA   ScrlPtr1+1
9A2E: 4C 0E 92        682           JMP   :1
9A31: AC 1F 93        683  :4       LDY   #Right       ;Move that saved line back now
9A34: B9 A6 91        684  :5       LDA   ScrlBufr,Y
9A37: 91 04           685           STA   (ScrlPtr2),Y
9A39: CC 1E 93        686           CPY   #Left
9A3C: F0 03           687           BEQ   :6
9A3E: 88              688           DEY
9A3F: 10 F3           689           BPL   :5
9A41: 60              690  :6       RTS                ;Finished with DOWN scroll
                      691  *-------------------------------------------------
                      692  * HorSub (& HorSubA) - A common subroutine to save space
                      693
9A42: BD 32 95        694  HorSub   LDA   YHiresL,X
9A45: 85 02           695           STA   ScrlPtr1
9A47: BD 72 94        696           LDA   YHiresH,X
9A4A: 85 03           697           STA   ScrlPtr1+1
9A4C: AC 1F 93        698           LDY   #Right
9A4F: B1 02           699  :1       LDA   (ScrlPtr1),Y
9A51: 99 A6 91        700           STA   ScrlBufr,Y
9A54: CC 1E 93        701           CPY   #Left
9A57: F0 03           702           BEQ   :2
9A59: 88              703           DEY
9A5A: 10 F3           704           BPL   :1
9A5C: 60              705  :2       RTS
                      706
9A5D: BD 32 95        707  HorSubA  LDA   YHiresL,X
9A60: 85 04           708           STA   ScrlPtr2
9A62: BD 72 94        709           LDA   YHiresH,X
9A65: 85 05           710           STA   ScrlPtr2+1
9A67: 60              711           RTS
                      712  *-------------------------------------------------
                      713  * ScrlLf - Left scroll routine.  It also uses the
                      714  *          same variables defined in ScrlUp.
                      715
9A68: AE 1C 93        716  ScrlLf   LDX   #Top         ;Start with top line
9A6B: BD 32 95        717  :1       LDA   YHiresL,X    ;Setup screen pointer
9A6E: 85 02           718           STA   ScrlPtr1
9A70: BD 72 94        719           LDA   YHiresH,X
9A73: 85 03           720           STA   ScrlPtr1+1
9A75: AC 1E 93        721           LDY   #Left        ;Save bits 0 & 1 from left byte
9A78: B1 02           722           LDA   (ScrlPtr1),Y
9A7A: 29 03           723           AND   #$03
9A7C: 0A              724           ASL
9A7D: 0A              725           ASL
9A7E: 0A              726           ASL
9A7F: 0A              727           ASL
9A80: 0A              728           ASL
9A81: 8D A8 91        729           STA   ScrlBufr+2   ;Saved here
9A84: B1 02           730  :2       LDA   (ScrlPtr1),Y ;Get a byte to process
9A86: 29 80           731           AND   #$80         ;Get only high bit (color)
9A88: 8D A6 91        732           STA   ScrlBufr     ;And save it for later
9A8B: B1 02           733           LDA   (ScrlPtr1),Y
9A8D: 29 7F           734           AND   #$7F         ;Remove color
9A8F: 4A              735           LSR                ;Shift two bits out
9A90: 4A              736           LSR
9A91: 0D A6 91        737           ORA   ScrlBufr     ;Replace color
9A94: 8D A7 91        738           STA   ScrlBufr+1   ;Save for temporary
9A97: CC 1F 93        739           CPY   #Right       ;At the right edge?
9A9A: F0 13           740           BEQ   :3           ;Yes
9A9C: C8              741           INY                ;Move to next byte
9A9D: B1 02           742           LDA   (ScrlPtr1),Y
9A9F: 29 03           743           AND   #$03         ;We need only two bits
9AA1: 0A              744           ASL                ;And shift the byte over
9AA2: 0A              745           ASL
9AA3: 0A              746           ASL
9AA4: 0A              747           ASL
9AA5: 0A              748           ASL
9AA6: 4D A7 91        749           EOR   ScrlBufr+1   ;Combine with former byte
```

```
92A9: 88        750           DEY
92AA: 91 02     751           STA   (ScrlPtr1),Y ;Replace on screen
92AC: C8        752           INY
92AD: 10 D5     753           BPL   :2           ;Keep going
92AF: AD A8 91  754  :3       LDA   ScrlBufr+2   ;Get the leftmost saved bits
92B2: 4D A7 91  755           EOR   ScrlBufr+1   ;Combine with current byte
92B5: 91 02     756           STA   (ScrlPtr1),Y ;And replace on screen
92B7: EC 1D 93  757           CPX   WBottom      ;Finished?
92BA: F0 04     758           BEQ   :4           ;Yes
92BC: E8        759           INX
92BD: 4C 6B 92  760           JMP   :1           ;No
92C0: 60        761  :4       RTS
                762  *--------------------------------------------------
                763  * ScrlRt - This is the right scroll routine.  Again it uses
                764  *          the same variables as defined in ScrlUp
                765
92C1: AE 1C 93  766  ScrlRt   LDX   WTop         ;Start at the top
92C4: BD 32 95  767  :1       LDA   YHiresL,X    ;Setup the left screen pointer
92C7: 85 02     768           STA   ScrlPtr1
92C9: BD 72 94  769           LDA   YHiresH,X
92CC: 85 03     770           STA   ScrlPtr1+1
92CE: AC 1F 93  771           LDY   WRight       ;Start fr right & proceed left
92D1: B1 02     772           LDA   (ScrlPtr1),Y
92D3: 29 60     773           AND   #$60         ;We need only save bits 5 & 6
92D5: 4A        774           LSR
92D6: 4A        775           LSR
92D7: 4A        776           LSR
92D8: 4A        777           LSR
92D9: 4A        778           LSR
92DA: 8D A8 91  779           STA   ScrlBufr+2   ;Save for later
92DD: B1 02     780  :2       LDA   (ScrlPtr1),Y
92DF: 29 80     781           AND   #$80         ;Save color bit for later
92E1: 8D A6 91  782           STA   ScrlBufr
92E4: B1 02     783           LDA   (ScrlPtr1),Y
92E6: 29 7F     784           AND   #$7F         ;Remove color for shifting
92E8: 0A        785           ASL
92E9: 0A        786           ASL
92EA: 29 7F     787           AND   #$7F         ;Remove color again
92EC: 0D A6 91  788           ORA   ScrlBufr     ;Restore original color status
92EF: 8D A7 91  789           STA   ScrlBufr+1   ;Save temporary
92F2: CC 1E 93  790           CPY   WLeft        ;At left already?
92F5: F0 13     791           BEQ   :3           ;Yes
92F7: 88        792           DEY
92F8: B1 02     793           LDA   (ScrlPtr1),Y ;Get next byte
92FA: 29 60     794           AND   #$60         ;Take only bits 5 & 6
92FC: 4A        795           LSR                ;Shift into position
92FD: 4A        796           LSR
92FE: 4A        797           LSR
92FF: 4A        798           LSR
9300: 4A        799           LSR
9301: 4D A7 91  800           EOR   ScrlBufr+1   ;Combine with original
9304: C8        801           INY
9305: 91 02     802           STA   (ScrlPtr1),Y ;Replace on screen
9307: 88        803           DEY
9308: 10 D3     804           BPL   :2
930A: AD A8 91  805  :3       LDA   ScrlBufr+2   ;Get rightmost bits
930D: 4D A7 91  806           EOR   ScrlBufr+1   ;Combine with present
9310: 91 02     807           STA   (ScrlPtr1),Y ;And place on screen
9312: EC 1D 93  808           CPX   WBottom      ;Finished?
9315: F0 04     809           BEQ   :4           ;Yes
9317: E8        810           INX
9318: 4C C4 92  811           JMP   :1
931B: 60        812  :4       RTS
                813
                814  * Frame - This subroutine will EOR (inverse) a single bit
                815  *         wide frame at WTop, WBottom, WLeft, and WRight.
                816  *         Call Frame again to erase with background left
                817  *         undisturbed.
                818
931C: 00        819  WTop     DFB   0            ;Frame Top
931D: 00        820  WBottom  DFB   0            ;Frame Temporary
931E: 00        821  WLeft    DFB   0            ;Frame Left
931F: 00        822  WRight   DFB   0            ;Frame Right
                823  FramPtr1 EQU   $02          ;Zero page to aim at Hi-Res page
                824
9320: 20 15 94  825  Frame    JSR   Savereg      ;Save all registers and zero page
9323: AE 1C 93  826           LDX   WTop
9326: 20 55 93  827           JSR   :3           ;Do the solid line at top
9329: 4C 48 93  828           JMP   :2
932C: BD 32 95  829  :1       LDA   YHiresL,X    ;Set FramPtr1 to Hi-Res vertical
932F: 85 02     830           STA   FramPtr1
9331: BD 72 94  831           LDA   YHiresH,X
9334: 85 03     832           STA   FramPtr1+1
9336: AC 1F 93  833           LDY   WRight       ;Right coordinate
9339: B1 02     834           LDA   (FramPtr1),Y ;Get screen byte
933B: 49 40     835           EOR   #$40         ;Reverse leftmost pixel
933D: 91 02     836           STA   (FramPtr1),Y ;And replace it
933F: AC 1E 93  837           LDY   WLeft        ;Now get the left byte
9342: B1 02     838           LDA   (FramPtr1),Y
9344: 49 01     839           EOR   #$01         ;This time reverse leftmost pixel
9346: 91 02     840           STA   (FramPtr1),Y
9348: E8        841  :2       INX                ;Go till bottom row
9349: EC 1D 93  842           CPX   WBottom
934C: D0 DE     843           BNE   :1
934E: 20 55 93  844           JSR   :3           ;Do the solid line at bottom
9351: 20 42 94  845           JSR   Restreg
9354: 60        846           RTS
9355: BD 32 95  847  :3       LDA   YHiresL,X    ;Routine inverses line at X-Reg
9358: 85 02     848           STA   FramPtr1
935A: BD 72 94  849           LDA   YHiresH,X
935D: 85 03     850           STA   FramPtr1+1
935F: AC 1F 93  851           LDY   WRight
9362: B1 02     852  :4       LDA   (FramPtr1),Y
9364: 49 7F     853           EOR   #$7F
9366: 91 02     854           STA   (FramPtr1),Y
9368: CC 1E 93  855           CPY   WLeft
936B: F0 03     856           BEQ   :5
936D: 88        857           DEY
936E: 10 F2     858           BPL   :4
9370: 60        859  :5       RTS
                860
                861  *--------------------------------------------------
                862  * This routine generates the YHires tables which hold the
                863  * Low & high bytes of 192 leftmost bytes of the Hi-Res
                864  * vertical screen locations.
                865

9371: A9 40     866  YGen     LDA   #$40
9373: 85 E6     867           STA   $E6          ;Page 2
9375: A9 BF     868           LDA   #191         ;Start on bottom line
9377: 8D 68 94  869           STA   Subtemp1
937A: A0 00     870  :1       LDY   #0
937C: 20 11 F4  871           JSR   $F411        ;Use AppleSoft ROM to find line
937F: AE 68 94  872           LDX   Subtemp1     ;Vertical line # for index
9382: A5 26     873           LDA   $26          ;Low byte
9384: 9D 32 95  874           STA   YHiresL,X    ;Store in Low table
9387: A5 27     875           LDA   $27          ;High byte
9389: 9D 72 94  876           STA   YHiresH,X    ;Store in High table
938C: CE 68 94  877           DEC   Subtemp1     ;Move up a line
938F: AD 68 94  878           LDA   Subtemp1
9392: C9 FF     879           CMP   #$FF         ;Finished with top line yet?
9394: D0 E4     880           BNE   :1           ;Nope, keep going
9396: 60        881           RTS                ;Finished
                882
                883  * Unpack - This routine unpacks the table at PackBufr onto
                884  *          Hi-Res page 1.  Follows the same standard coding
                885  *          convention as described in Pack.
                886
                887  * Uses the following variables as defined in Pack:
                888
                889  * PackPtr1 :Temporary zero page pointer for indexing
                890  * PackPtr2 :  "     "    "    "    "    "
                891  * Subtemp1 :For temporary storage
                892  * Subtemp2 :  "     "    "
                893
9397: 00 00     894  LocPack  DFB   0,0
                895
9399: 20 15 94  896  Unpack   JSR   Savereg      ;Save registers and zero page locs
939C: A9 00     897           LDA   #$00         ;Set vert and horiz to $00
939E: 8D 68 94  898           STA   Subtemp1
93A1: 8D 69 94  899           STA   Subtemp2
93A4: AD 97 93  900           LDA   LocPack      ;Set PackPtr1 to data location
93A7: 85 00     901           STA   PackPtr1
93A9: AD 98 93  902           LDA   LocPack+1
93AC: 85 01     903           STA   PackPtr1+1
93AE: AD 69 94  904  :8       LDA   Subtemp2     ;Check if finished (horiz = 40)
93B1: C9 28     905           CMP   #40
93B3: D0 04     906           BNE   :1
93B5: 20 42 94  907           JSR   Restreg      ;Restore everything and exit
93B8: 60        908           RTS
93B9: 20 DA 93  909  :1       JSR   Unpack6      ;Get command byte fr data table
93BC: C9 80     910           CMP   #$80         ;If high bit set then goto :3
93BE: B0 0C     911           BGE   :3
93C0: AA        912           TAX                ;X-Reg counts unique bytes
93C1: 20 DA 93  913  :2       JSR   Unpack6      ;Get a byte
93C4: 20 E5 93  914           JSR   Unpack7      ;Put on screen
93C7: CA        915           DEX
93C8: 10 F7     916           BPL   :2           ;Go till finished
93CA: 30 E2     917           BMI   :8
93CC: 29 7F     918  :3       AND   #$7F         ;Convert to low ascii
93CE: AA        919           TAX                ;and use x-reg as counter
93CF: 20 DA 93  920           JSR   Unpack6      ;Get the repeating byte
93D2: 20 E5 93  921  :4       JSR   Unpack7      ;Put on screen
93D5: CA        922           DEX
93D6: 10 FA     923           BPL   :4           ;Go till finished
93D8: 30 D4     924           BMI   :8
                925  *--------------------------------------------------
93DA: A0 00     926  Unpack6  LDY   #$00         ;Get a byte from data table
93DC: B1 00     927           LDA   (PackPtr1),Y
93DE: E6 00     928           INC   PackPtr1
93E0: D0 02     929           BNE   :1
93E2: E6 01     930           INC   PackPtr1+1
93E4: 60        931  :1       RTS
                932  *--------------------------------------------------
93E5: 48        933  Unpack7  PHA                ;Routine to put a byte on screen
93E6: AC 68 94  934           LDY   Subtemp1     ;Vertical
93E9: B9 32 95  935           LDA   YHiresL,Y
93EC: 85 02     936           STA   PackPtr2
93EE: B9 72 94  937           LDA   YHiresH,Y
93F1: 85 03     938           STA   PackPtr2+1
93F3: AC 69 94  939           LDY   Subtemp2     ;Horizontal
93F6: 68        940           PLA
93F7: 48        941           PHA
93F8: 91 02     942           STA   (PackPtr2),Y
93FA: AD 68 94  943           LDA   Subtemp1     ;Increment vertical twice
93FD: 18        944           CLC
93FE: 69 02     945           ADC   #$02
9400: C9 C0     946           CMP   #$C0
9402: 90 0C     947           BLT   :1
9404: 38        948           SEC
9405: E9 BF     949           SBC   #$BF
9407: C9 02     950           CMP   #$02
9409: D0 05     951           BNE   :1
940B: A9 00     952           LDA   #$00
940D: EE 69 94  953           INC   Subtemp2
9410: 8D 68 94  954  :1       STA   Subtemp1
9413: 68        955           PLA
9414: 60        956           RTS
                957
                958  * Savereg - This small routine just pushes the A, X, and Y
                959  *          registers onto the stack for later retrieval by
                960  *          Restreg below.  Generally called by subroutines
                961  *          that want to leave registers unaltered.
                962
9415: 8D 70 94  963  Savereg  STA   Specireg+2
9418: 68        964           PLA
9419: 8D 6E 94  965           STA   Specireg
941C: 68        966           PLA
941D: 8D 6F 94  967           STA   Specireg+1
9420: AD 70 94  968           LDA   Specireg+2
9423: 48        969           PHA
9424: 8A        970           TXA
9425: 48        971           PHA
9426: 98        972           TYA
9427: 48        973           PHA
9428: 8E 71 94  974           STX   Specireg+3
942B: A2 07     975           LDX   #$7
942D: B5 00     976  :1       LDA   $0000,X
942F: 48        977           PHA
9430: CA        978           DEX
9431: 10 FA     979           BPL   :1
9433: AE 71 94  980           LDX   Specireg+3
```

Listing 2 for WindoWorks

WINDOWORKS.ML *(continued)*

```
9436: AD 6F 94  981           LDA   Specireg+1
9439: 48        982           PHA
943A: AD 6E 94  983           LDA   Specireg
943D: 48        984           PHA
943E: AD 70 94  985           LDA   Specireg+2
9441: 60        986           RTS
                987
                988 * Restreg - This is the counterpart to Savereg (above) and
                989 *           merely pulls the A, X, and Y registers off the
                990 *           stack.
                991
9442: 68        992 Restreg   PLA
9443: 8D 6E 94  993           STA   Specireg
9446: 68        994           PLA
9447: 8D 6F 94  995           STA   Specireg+1
944A: A2 00     996           LDX   #$00
944C: 68        997 :1        PLA
944D: 95 00     998           STA   $0000,X
944F: E8        999           INX
9450: E0 08     1000          CPX   #$08
9452: D0 F8     1001          BNE   :1
9454: 68        1002          PLA
9455: A8        1003          TAY
9456: 68        1004          PLA
9457: AA        1005          TAX
9458: 68        1006          PLA
9459: 8D 70 94  1007          STA   Specireg+2
945C: AD 6F 94  1008          LDA   Specireg+1
945F: 48        1009          PHA
9460: AD 6E 94  1010          LDA   Specireg
9463: 48        1011          PHA
9464: AD 70 94  1012          LDA   Specireg+2
9467: 60        1013          RTS
                1014
                1015 * These variables are globally available for subroutines to
                1016 * use and may be destroyed at will, but not passed between
                1017 * routines.  Please use special variables for that.
                1018
9468: 00        1019 Subtemp1 DFB   0
9469: 00        1020 Subtemp2 DFB   0
946A: 00        1021 Subtemp3 DFB   0
946B: 00        1022 Subtemp4 DFB   0
946C: 00        1023 Subtemp5 DFB   0
946D: 00        1024 Subtemp6 DFB   0
                1025
                1026 * These are special variables for Savereg and Restreg and
                1027 * should never be used.
                1028
946E: 00 00 00  1029 Specireg DFB   0,0,0,0
9471: 00
                1030
                1031 YHiresH  EQU   *
                1032 YHiresL  EQU   *+192
```

END OF LISTING 2

```
                KEY PERFECT 5.0
                    RUN ON
                WINDOWORKS.ML

=======================================
   CODE-5.0    ADDR# - ADDR#    CODE-4.0
   --------    -------------    --------
   248D4065     6000 - 604F        2529
   698287FC     6050 - 609F        285B
   D22E73D0     60A0 - 60EF        2579
   644EBAB1     60F0 - 613F        27FC
   D6934E3E     6140 - 618F        26AE
   F5B97F36     6190 - 61DF        27B8
   99AB3B1B     61E0 - 622F        2753
   C274EE0E     6230 - 627F        2725
   0412B27D     6280 - 62CF        21F0
   8026CA4F     62D0 - 631F        250B
   B172789D     6320 - 636F        2B08
   93594FC5     6370 - 63BF        293E
   2B8517C9     63C0 - 640F        25EC
   5678BE35     6410 - 645F          00
   5678BE35     6460 - 64AF          00
   5678BE35     64B0 - 64FF          00
   2BEC8CBE     6500 - 654F        107E
   9B4EE38F     6550 - 659F        2657
   65E18528     65A0 - 65EF        1BB1
   EB031A19     65F0 - 663F        22EE
   AA3CA749     6640 - 668F        1BE2
   6395BFF8     6690 - 66DF        2713
   D6A9E970     66E0 - 672F        2275
   DE2810B8     6730 - 677F        27CE
   E5AA8758     6780 - 67CF        289A
   C2904378     67D0 - 681F        2991
   7C239160     6820 - 6858        1D05
   CAB28195 = PROGRAM TOTAL =      0859
```

Listing 3 for WindoWorks

WINDOWORKS.DEMO

```
1   REM ++++++++++++++++++++++
2   REM +  WINDOWORKS.DEMO   +
3   REM +  by  Bob Thrasher  +
4   REM + Copyright (C) 1987 +
5   REM + by MicroSPARC, Inc +
6   REM + Concord, MA  01742 +
7   REM ++++++++++++++++++++++
10  TEXT : PRINT  CHR$ (12) CHR$ (21): HOME :
      PRINT : VTAB 10: HTAB 12: PRINT "WINDOW
    ORKS.DEMO": VTAB 12: HTAB 10: PRINT "by
    Bob Thrasher": VTAB 14: HTAB 1: PRINT "C
    opyright (C) 1987 by MicroSPARC, Inc.": VTAB
    23: HTAB 8: PRINT "Please wait ...";
20  HIMEM: 15360: FOR A = 774 TO 814: READ C:
      POKE A,C: NEXT :D = 49200
30  B$ = "/////39@N@>@>@>=@0@=>@@@>N0>0NN0>00M
    0N@M@@N@@N333NN7785@868@0000N@JD@@@BDH@=
    @@@=>@>00=@D8E>@>8@M0=?>N3333@@@@=@@@93@
    @DJ@@939@@@933N731N"
40  D$ =  CHR$ (4): DIM A(18),B(26,4): ONERR  GOTO
    60
50  PRINT : PRINT D$"BLOADWINDOWORKS.ML": GOTO
    70
60  HOME : INVERSE : PRINT "FATAL ERROR";: NORMAL
    : PRINT " -": PRINT : PRINT : PRINT "THI
    S DEMO REQUIRES THE FILE": PRINT : INVERSE
    : PRINT "WINDOWORKS.ML": NORMAL : PRINT
    : PRINT "ON THE DISKETTE.": PRINT : END

70  ONERR  GOTO 270
80  FOR A = 0 TO 26: FOR C = 0 TO 4:B(A,C) =
      ASC ( MID$ (B$,A * 5 + C + 1,1)) - 47: NEXT
    : NEXT : FOR A = 0 TO 18:A(A) = ( PEEK (
    24576 + A * 2) + 256 *  PEEK (24577 + A *
    2)): NEXT : CALL A(8): CALL A(18): POKE
    A(5),96
90  VTAB 23: HTAB 8: PRINT "Press Return to c
    ontinue";: GET Z$: PRINT : HGR2 :G = 1:H
     = 4:A$ = "WINDOWORKS DEMONSTRATION PROG
    RAM": GOSUB 170: HCOLOR= 3: HPLOT 0,8 TO
    279,8:I = 0: GOSUB 240: GOSUB 250: GOSUB
    260: GOSUB 210
100 I = 0: GOSUB 180: GOSUB 240: GOSUB 250: GOSUB
    260: FOR A = 0 TO 5: HCOLOR= A + 1: FOR
    C = A * 7 TO A * 7 + 6: HPLOT C + 49,88 TO
    C + 49,143: NEXT : NEXT : GOSUB 210
110 I = 0: GOSUB 180: GOSUB 240:K = 28:L = 11
    1:M = 16:N = 71: GOSUB 200:K = 168:L = 2
    51: GOSUB 200:M = 88:N = 143: GOSUB 200:
    K = 28:L = 111: GOSUB 200
120 GOSUB 250: GOSUB 260: FOR A = 0 TO 5: HCOLOR=
    A + 1: FOR C = A * 7 TO A * 7 + 6: HPLOT
    C + 189,112 TO C + 189,143: NEXT : NEXT
    : GOSUB 210
130 I = 0: GOSUB 180: FOR W = 0 TO 7:P = W *
    5 + 1:Q = P + 2:R = 2:S = 17: GOSUB 220:
    H = W * 5 + 2:G = 80:A$ = "O": GOSUB 170
    :P = W * 5 + 2:Q = P:R = 2:S = 17:E = 0:
    F = W: GOSUB 230: NEXT
140 H = 1:G = 160:A$ = "AN EXAMPLE OF THE EIG
    HT SPEED SETTINGS": GOSUB 170: GOSUB 210
150 I = 0: GOSUB 180: GOSUB 240:K = 28:L = 97
    :M = 63:N = 96: GOSUB 200:K = 182:L = 25
    1: GOSUB 200: GOSUB 250: GOSUB 260: GOSUB
    210
160 I = 0: GOSUB 180: GOSUB 240:M = 40:N = 55
    :K = 140:L = 210: GOSUB 200: GOSUB 250: GOSUB
    260: HCOLOR= 3: FOR A = 7 TO 32: HPLOT A
     * 7 + 3,134: NEXT : GOSUB 210: TEXT : HOME
    : END
170 FOR A = 1 TO  LEN (A$): POKE 1,H:B =  ASC
    ( MID$ (A$,A,1)) - 64:B = B * (B > 0 AND
    B < 27): FOR C = 0 TO 4: POKE 768 + C,B(
    B,C):T =  PEEK (D): NEXT : POKE 0,G: CALL
    774:H = H + 1: NEXT : RETURN
180 HCOLOR= 0: FOR A = 191 TO 9 STEP  - 1: HPLOT
    0,A TO 279,A:T =  PEEK (D): NEXT : RETURN

190 HCOLOR= 3: HPLOT K,M TO L,M TO L,N TO K,
    N TO K,M: RETURN
200 HCOLOR= 0: FOR A = M TO N: HPLOT K,A TO
    L,A: NEXT : RETURN
210 G = 183:H = 7:A$ = "PRESS RETURN TO CONTI
    NUE": GOSUB 170:M = 180:N = 190:K = 45:L
```

```
    = 225: GOSUB 190: POKE 49152,0: CALL A(
    10): RETURN
220 K = P * 7 - 1:L = (Q + 1) * 7:M = R * 8 -
    1:N = (S + 1) * 8: GOSUB 190: RETURN
230 V = A(17) + I * 7: POKE V,R * 8: POKE V +
    1,S * 8 + 7: POKE V + 2,P: POKE V + 3,Q:
     POKE V + 4,E: POKE V + 5,F: POKE V + 6,
    0:I = I + 1: POKE A(17) + I * 7,255: RETURN

240 READ J: FOR W = 1 TO J: READ P,R,Q,S: GOSUB
    220: NEXT : RETURN
250 READ J: FOR W = 1 TO J: READ H,G,A$:G =
    G * 8: GOSUB 170: NEXT : RETURN
260 READ J: FOR W = 1 TO J: READ P,R,Q,S,E,F
    : GOSUB 230: NEXT : RETURN
270 TEXT : HOME :Z =  PEEK (222): IF Z = 255
     THEN  PRINT "Please do not press Contro
    l-C": PRINT : PRINT "Type RUN to restart
     demo": PRINT : END
280 PRINT "An error exists in the program li
    sting.": PRINT : PRINT "Please recheck t
    he listing.  Perhaps": PRINT : PRINT "th
    e error is in LINE # "; PEEK (218) +  PEEK
    (219) * 256: PRINT : END
290 DATA 169,4,141,5,3,165,0,24,109,5,3,170,
    189,50,149,141,39,3,189,114,148,141,40,3
    ,164,1,174,5,3,189,0,3,153,255,255,206,5
    ,3,16,221,96
300 DATA 4,4,2,15,8,24,2,35,8,4,11,15,17,24,
    11,35,17,6,9,5,UP,28,5,LEFT,8,14,DOWN,28
    ,14,RIGHT,7,19,"A SAMPLE OF THE FOUR BAS
    IC",7,20,"TYPES OF SCROLLING WINDOWS"
310 DATA 4,9,2,10,8,0,0,24,5,35,5,2,0,8,11,1
    1,17,1,0,24,14,35,14,3,0,2,14,2,25,8,4,1
    1,15,17,5,16,5,INVERSE,27,14,FRAME,4,10,
    "COLOR CHANGE"
320 DATA 7,19,"THESE ARE THE THREE BASIC",11
    ,20,"NON-MOVING WINDOWS",4,14,2,25,8,4,2
    ,4,11,15,17,5,1,24,11,35,17,6,3,0,2,2,17
    ,2,0
330 DATA 8,4,2,15,8,24,2,35,8,4,11,15,17,24,
    11,35,17,5,3,16,9,25,3,36,9,5,12,16,18,2
    5,12,36,18
340 DATA 10,8,4,UP,8,5,LEFT,28,4,UP,28,5,RIG
    HT,8,13,DOWN,7,14,INVERSE,27,12,DOWN,27,
    13,COLOR,3,20,"SUPERIMPOSING TWO OR MORE
     WINDOWS",3,21,"CAN PRODUCE OTHER ANIMAT
    ION TYPES"
350 DATA 8,4,2,15,8,0,1,4,2,15,8,2,1,24,2,35
    ,8,0,1,24,2,35,8,3,1,7,11,13,17,1,0,4,11
    ,15,17,4,1,27,11,32,17,1,0,27,11,32,17,5
    ,3,4,4,3,13,16,2,8,15,11,26,3,35,16,24,8
    ,37,11,7,8,5,AB,8,6,CD,30,5,AB,30,6,CD
360 DATA 0,18,"LARGE WINDOW IS UP    LARGE W
    INDOW IS UP",0,19,"SMALL MOVES UP ALSO
     SMALL MOVES RIGHT"
370 DATA 2,21,"EXAMPLES OF PARTIAL OVERLAP W
    INDOWS",4,8,3,9,16,0,0,8,8,9,11,0,0,26,3
    ,35,16,0,0,26,8,35,11,3,0,4,7,5,32,6,23,
    4,24,7,25,4,26,7,7,16,32,16,6,6,4,"LARGE
     MOVES LEFT",19,9,DOWN,27,9,UP,15,6,AB
380 DATA 9,15,"MOVING RIGHT AND UPWARD",0,20
    ,"TWO LAST EXAMPLES OF ADVANCED TECHNIQU
    ES",5,7,5,32,6,2,0,23,5,24,6,1,0,25,5,26
    ,6,0,0,7,16,32,16,3,0,7,16,32,16,0,0
```

END OF LISTING 3

```
            KEY PERFECT 5.0
                RUN ON
            WINDOWORKS.DEMO
==========================================
   CODE-5.0    LINE# - LINE#    CODE-4.0
   --------    -------------    --------
   2F3F056E        1 -    30        E67B
   3A19C820       40 -   130      017BE4
   F2136D1F      140 -   230      0130E7
   BBE9CFBF      240 -   330      01CAF0
   128FBE0E      340 -   380      018356
   6EF2DF1C = PROGRAM TOTAL =      0E2C
```