 Apple Lisa Computer
Technical Information

Apple Lisa Computer: Workshop Review (DTC 1988)

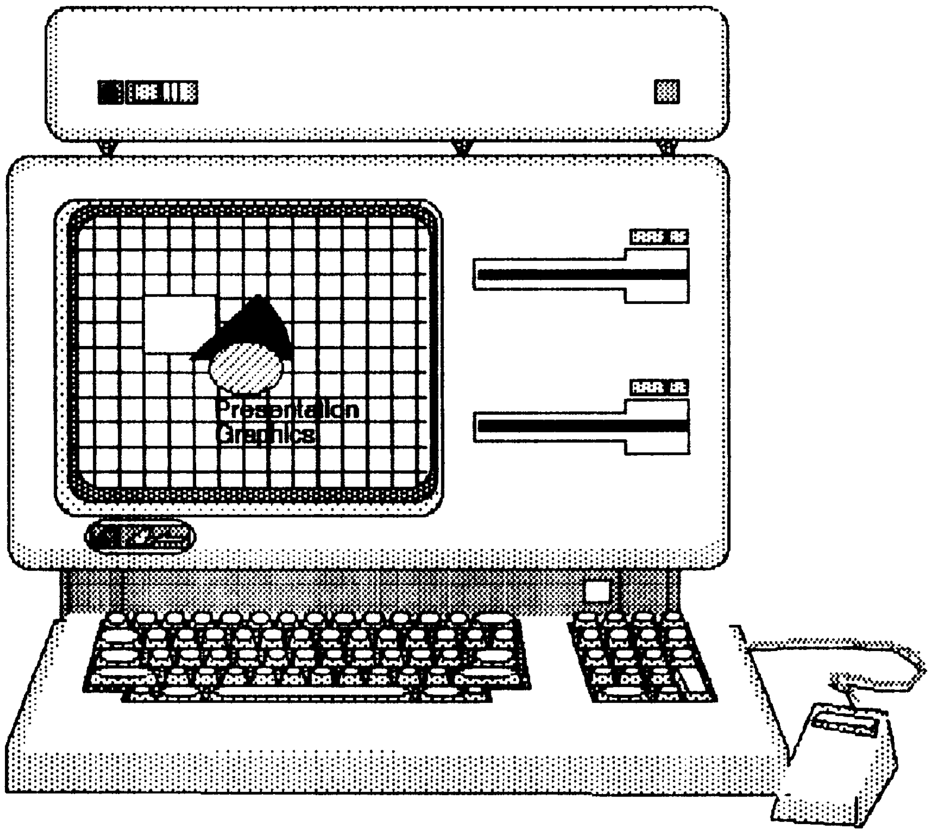
Lisa Computer: 1983 - 1985

Printed by: Macintosh Picture Printer 0.0.5 1999-01-11 Printed: 1999-02-01 17:34:11

Apple Lisa Personal Computer

1983 to 1985

Lisa WorkShop Review by DTC



David T. Craig - 736 Edgewater, Wichita, Kansas 67230 - (316) 733-0914

"LisaWkShopReviewDTCOct88 0.PICT" 256 KB 1999-02-01 dpi: 360h x 363v pix: 2546h x 3236v

A Review of Apple's Lisa WorkShop

by
David Craig
736 Edgewater, Wichita KS 67230

(October 12, 1988)

INTRODUCTION

The WorkShop was Apple's software development environment for the Lisa during the years 1981 to 1985. From 1981 to 1983 this environment produced all the code for the Lisa. After Apple's Macintosh introduction in 1984 Macintosh developers used the WorkShop since no native Macintosh development environment existed.

The WorkShop is a command line shell similar in appearance to Apple's older Apple // and /// UCSD-like Pascal shells. The main command line provides access to two other command lines, the File Manager and the System Manager, and to several programming tools. The programming tools consist of an editor, Pascal compiler, 68000 assembler, and a linker. Even though the WorkShop supports various languages it is really tailored toward Pascal since most of its utilities are for Pascal source code and object files. The main command line appears as

```
{V3.9} WORKSHOP: FILE-MGR, SYSTEM-MGR, Edit, Run, Debug, Pascal, Basic, Quit, ?
```

with the other half of this line being

```
Assemble, Generate, MakeBackground, Link, TransferProgram
```

FILE MANAGER

The File Manager provides access to files which are stored either on 400K microdiskettes, hard disks, or tape backup systems. The Lisa Operating System supports a hierarchical file structure which the WorkShop also supports. The command line for the File Manager appears as

```
FILE-MGR:  Backup, Copy, Delete, List, Online, Prefix, Rename, Transfer, Quit, ?
```

with the other half of this line being

```
AddCatalog, Equal, FileAttributes, Initialize, Mount, Names, Scavenge, Unmount
```

With this command line you could backup, copy, and delete files. Wildcard characters within file names are supported for easier file selection. For a directory listing the List command displays a listing complete with file sizes, creation and modification dates, and file attributes. A sample follows:

```
Filename                     Size Psize  Last-Mod-Date   Creation-Date  Attr
---------                    ---- -----  -------------   -------------  ----
CARTOG.OBJ                   2560     5  10/02/88-10:53  10/02/88-10:52 SC
CARTOG.TEXT                 15360    30  09/29/88-18:20  12/03/87-23:40 0
CARTOG/backup.TEXT           2048     4  10/01/88-11:22  09/28/88-22:01
CARTOG/Map_Util.OBJ         34304    67  10/09/88-15:26  10/09/88-15:25
CARTOG/Map_Util.TEXT        50176    98  10/08/88-16:33  09/11/88-17:05 P
CARTOG/Map_UtilX.TEXT        4096     8  09/29/88-22:17  09/11/88-17:37
CARTOG/UNewMapMgr.OBJ       47104    92  10/09/88-15:22  10/09/88-15:22
CARTOG/UNewMapMgr.TEXT      82944   162  10/09/88-15:20  09/28/88-09:48
CARTOG/UOldMapMgr.OBJ       15872    31  10/09/88-15:21  10/09/88-15:21
CARTOG/UOldMapMgr.TEXT      32768    64  10/08/88-16:34  09/11/88-18:21
CARTOGX.TEXT                 2048     4  09/29/88-14:41  09/08/88-23:37
```

```
565 total blocks for files listed
94 blocks of OS overhead for volume and files listed
1115 blocks free out of 17418
```

The attributes field specifies the attributes of a file. These are defined as follows:

```
O - File is currently open
C - File was closed by the Operating System
S - File has its Safety flag set
P - File is Protected from copying
```

The **Online** command displays a list of the currently mounted devices as follows:

```
DevName DevAlias    VolumeName        VolSize  FreeBlks  Files  Open  Attr
------- --------    ----------        -------  --------  -----  ----  ----
#12     UPPER       AOS 3.0             17418      1113    244    42  MOP
#13     LOWER       lisa diskette         772       762      4     0  M
#15#1   ALTCONSOLE                          0         0      0     0  M
#15#2   MAINCONSOLE                         0         0      0     1  M
#10#1   RS232A      <printer>               0         0      0     0  M
#2#1    SLOT2CHAN1  ProFile              9690      2714    189     0  M
```

The **AddCatalog** command creates a new subdirectory, **Equal** compares files for equality, and **Initialize** formats disk volumes. The **Scavenge** command analyses disk volumes for irregularities and repairs them if any problems exist. **Mount** and **Unmount** make external devices visible to the Lisa Operating System and the WorkShop. The **FileAttributes** command produces the following command line:

```
File Attributes: ClearAttributes, Protect, Safety, AddPassword, RemovePassword, Quit
```

This command line deletes or specifies new file attributes. The **Safety** command marks a file as non-deletable so that if you try and delete it the delete will fail. This attribute is very useful for important files. **Protect** marks a file as protected so that the file can never be copied. This attribute was used by the Lisa Office System (a.k.a. Lisa 7/7) to turn programs into "protected masters". **AddPassword** and **RemovePassword** allow file password protection.

SYSTEM MANAGER

The System Manager provides access to several low-level features of the Lisa. Its command line is

```
SYSTEM-MGR:  ManageProcess, OutputRedirect, Preferences, Time, Quit, ?
```

with the second half appearing as

```
Console, FilesPrivate, Validate, DefaultPrinter
```

ManageProcess lists all the current system processes and can terminate executing processes. **OutputRedirect** redirects all console output to a text file whose name you specify. **Preferences** runs the Lisa Preferences tool which appears as a window with buttons that are mouse controlled. This window allows you to specify several hardware parameters of the Lisa such as the screen brightness, speaker volume level, and keyboard and mouse sensitivity. **Time** displays the current clock date and time. **Console** allows the WorkShop's main console I/O to originate from either the alternate console or an external terminal connected to one of the Lisa's serial ports. **FilesPrivate** enables the File Manager to have access to special Office System files whose names start with "{". Generally WorkShop programs should not access these files since their integrity is vital for correct operation of the Office System.

The **Validate** command controls several of what are called "paranoia" settings, such as whether file transfers will be verified and whether file selections will require user confirmation for File Manager operations. This allows you to tailor the system to your level of confidence in it and in yourself.

PROGRAMMING TOOLS

The Lisa WorkShop supports several powerful programming tools. Access to these is available through the main command line. The commands in the line for the tools are

Edit, Debug, Pascal, Basic, Assemble, Generate, Link

The first tool that is usually used is the source code editor. The editor, called **LisaEdit**, provides full window, menu bar, and mouse support. Menus exist in a menu bar which supports pull-down menus. These handle file creation, opening, saving, and printing. In an early version of LisaEdit the print menu supported underlining of Pascal keywords. Unfortunately, later versions eliminated this handy feature. Multiple overlapping and resizable windows are available for editing source code files with file size limited only by the amount of memory your Lisa has. With my 1 Mbyte system I have had been able to work with about a dozen large programs. Text resizing is available through the Type Style menu. You can have very small text with about 150 characters per line (cpl) to very large text with about 60 cpl. Cutting and pasting is done with the mouse and the Edit menu which supports the commands Cut, Copy, and Paste. The Lisa's arrow keys on the keypad are supported for cursor movement. The Undo menu command provides the ability to "undo" your last operation. For example, if you Cut some text that was not supposed to be cut Undo will undo the Cut. When you leave LisaEdit and return to the WorkShop main command line the opened windows in LisaEdit remain active. Later, when you reenter LisaEdit the windows appear automatically. This is very handy for modifying a program, leaving LisaEdit to compile the program, and returning to LisaEdit.

The **Pascal Compiler** is the heart of the Lisa WorkShop. For a detailed review of this compiler's features see my paper titled "A Review of Apple's Lisa Pascal". This compiler generates I-code (intermediate code) which is actually only standard UCSD Pascal P-code. If an error occurs during a compilation the editor is run, the source file is loaded, the cursor is placed over the offending statement, and a specific error message is displayed. The **Code Generator**, which is run automatically by the Compiler, takes an I-code file and produces 68000 object code. The **Linker** links different files and libraries to create executable object files. The Compiler and Code Generator display a lot of compilation statistics as the following example shows:

```
Lisa Pascal Compiler V3.76 (05-Apr-85)                10:50:46 10-Oct-88
(c)1981 SVS, Inc.  (c)1983, 1984 Apple Computer, Inc.

Input file - [.TEXT] RangeTester
List file - [.TEXT]
Output file - [RangeTester] [.OBJ]

[242109 words] RANGE_TE

Elapsed time: 5.901 seconds.
Compilation complete - no errors found.  9 lines.
```

```
Lisa Pascal MC68000 Code Generator V3.65 (20-Mar-85) 10:51:00 10-Oct-88
(c)1981 SVS, Inc.  (c)1983, 1984 Apple Computer, Inc.

Input file - $I+
Input file - [.I] RangeTester
Output file - [RangeTester] [.OBJ] RangeTester

RANGE_TE - RANGE_TE    Code size =     88

Elapsed time: 2.580 seconds.
Total code size = 088


Linker - M68000 Object Code  v0.9.3.1 08-Apr-85 15:26:15
Copyright Apple Computer, Inc. 1985

Beginning memory:            269656
After initial allocation:    233146
Input file [.OBJ] ? ?
Options ? ?
Options are:
    Option Value Description:
    ------ ----- ------------
    +A -A   '-'  Alphabetical Listing
    +C MODNAME SegName    Copy module into segment
    +F -F   '-'  For domain 0
    -H num   Initial Stack Swap Area:    4096
    +I -I   '+'  Interfaces Flag
    +L -L   '-'  Location Ordered Listing
    +M fromName toName       Segment Name Mapping
    +O -O   '+'  Emit O.S. Data record
    +P -P   '-' Physical Link, machines w/out MMU's.
    +R -R   '-' MODNAME Do partial link
    If MODNAME is provided, dead code will be stripped using it as the root
    +S num   Start Dynamic Stack Size:   10240
    +T num   Top Dynamic Stack Size:    131072
    +W Which directory file: -#12-INTRINSIC.LIB
    +X -X   '-' Cross-develop to MAC
Options ?
Input file [.OBJ] ? RangeTester
Input file [.OBJ] ? IOSPASLIB
Input file [.OBJ] ?
Listing file [-CONSOLE] / [.TEXT]
Output file ? [.OBJ] RangeTester
Reading file: RangeTester.OBJ
Reading file: IOSPASLIB.OBJ
Input summary:
    2 Files      , max =   100
    6 Segments   , max =  4096
   19 Modules    , max = 32768
   11 Entries    , max = 65536
    4 Ref. Lists, max = 65536
    4 References, max = 65536
Linking Main Program.
Reading Library Directory: -#12-INTRINSIC.LIB
Active : 1 of 19 read.
Visible: 1 of 11 read.
Global data: $000016
Common data: $000000
Number of segments in file = 1, number of Jump Table entries = 1
Linking segment:          file (JT) seg:   1 size:       88
0 Errors detected.

RangeTester.OBJ is an executable program file.

Elapsed time: 24.210 seconds.
That's all Folks!
```

The Pascal Compiler supports many directives which modify its behavior. For example, you can control integer and sub-range checking, short-circuit boolean evaluation, code optimization control (i.e., off, old scheme, new scheme), program routine name inclusion into the object code for debugging with LisaBug, assembly language listing, and conditional compilation. The last item is implemented using the following directives:

{$DECL var_name}	declares var_name to be a conditional variable
{$SETC var_name := expression}	sets var_name to a boolean or an integer
{$IFC expression}	tests if expression is true, if true includes following code
{$ELSEC}	the else part in the if-then-else-endif construct
{$ENDC}	the endif part in the if-then-else-endif construct

The assembly listing shows the generated assembly code listed after each source line. The listing for a simple Pascal program follows:

```
Lisa Pascal Compiler V3.76 (05-Apr-85)                          10:51:45 10-Oct-88
Lisa Pascal MC68000 Code Generator V3.65 (20-Mar-85)            10:51:57 10-Oct-88

1       1 --            PROGRAM Range_Tester; {$ASM+}
2       2 --
3       3 --              VAR a : ARRAY [0..9] OF INTEGER;
4       4 --                  i : 0..10;
5       5 --
6       6 0-              BEGIN
          000000 4EBA 0000        RANGE_TE JSR      %_BEGIN
          000004 4E56 0000                 LINK     A6,#$0000
          000008 2C5F                      MOVE.L   (A7)+,A6
          00000A 4E55 FFEA                 LINK     A5,#$FFEA
          00000E 9FED 0010                 SUBA.L   $0010(A5),A7
          000012 4EBA 0000                 JSR      %_INIT
7       7 --                FOR i := 0 TO 10 DO
          000016 4220 FFEB                 CLR.B    $FFEB(A5)
          00001A 601A                      BRA.S    L0001          ; 00000036
8       8 --                  a[i] := i;
          00001C 1020 FFEB        L0002    MOVE.B   $FFEB(A5),D0
          000020 4880                      EXT.W    D0
          000022 41BC 0009                 CHK      #$0009,D0
          000026 E340                      ASL.W    #$1,D0
          000028 1220 FFEB                 MOVE.B   $FFEB(A5),D1
          00002C 4881                      EXT.W    D1
          00002E 3B81 00EC                 MOVE.W   D1,$EC(A5,D0.W)
          000032 5220 FFEB                 ADDQ.B   #$1,$FFEB(A5)
          000036 0C20 000A FFEB L0001      CMPI.B   #$000A,$FFEB(A5)
          00003C 6FDE                      BLE.S    L0002          ; 0000001C
          00003E 4EBA 0000                 JSR      %_TERM
          000042 4E5D                      UNLK     A5
          000044 4EBA 0000                 JSR      %_END
          000048 4E75                      RTS
          00004A 4E5E                      UNLK     A6
          00004C 4E75                      RTS

          00004E D241 4E47 455F            .WORD    $D241,$4E47,$455F ; ".ANGE_"
          000054 5445                      .WORD    $5445             ; "TE"

          000056 0000             CstSize  .WORD    Last-CstSize-2
          000058                  Last
9       9 -0              END.

Elapsed compilation time: 3.197 seconds.
Compilation complete - no errors found.  9 lines.
Elapsed code generator time: 4.075 seconds.
Total code size = 088
```

The **68000 Assembler** is used to create assembly routines which will later be linked to a Pascal program. Assembly is really used only for time or space critical code.

The Lisa WorkShop supports several other languages which must be purchased separately. A **C Compiler** was created shortly after the Lisa was introduced, but little programming effort seems to have been done with C due to the Lisa's overall Pascal perspective. **LisaBasic** was a rewrite of DEC's Basic-PLUS language, but since

LisaBasic was an interpreter its use never caught on with programmers. **LisaCOBOL** appeared but was shortly forgotten. One interesting language which Apple developed for the Lisa but was never announced or released was **Magic/L** (pronounced "magical"). This language was a combination between Pascal and Forth, which I assume Apple Inc. found too exotic even for their revolutionary Lisa.

Debugging Lisa programs is accomplished in either of two ways. The preferred method for Pascal and other high level languages is to write debugging data to the Lisa's alternate console. This console, whose device name is "-ALTCONSOLE", is displayed when the Right-Option and Keypad-Enter keys are simultaneously pressed. While a program is running it displays debugging data on the alternate console using Pascal's WRITELN. The program's execution continues even when this console is displayed. To debug assembler or compiled code **LisaBug** exists. LisaBug uses the alternate console for both input and output. To debug a program use the Debug command from the main command line and you will be asked for the name of the file to debug. A breakpoint is set at the first instruction of the program and you can then single step, trace, or disassemble the program using various LisaBug commands. Routine names or symbols are available if the compiler's debugging directive, {$D+}, was enabled. When a runtime error occurs in a WorkShop program LisaBug is invoked. For example, the following Pascal program contains a runtime range error:

```
PROGRAM Range_Tester;

VAR a : ARRAY [0..9] OF INTEGER;
    i : 0..10;

BEGIN
  FOR i := 0 TO 10 DO   { <--- Runtime error occurs when i = 10 }
  a[i] := i;
END.
```

When this program runs the Lisa displays the following information on the alternate console and makes this console visible:

```
Level 7 Interrupt
RANGE_TE+0022 E340                          ASL.W #$1,D0
PC=0002002E SR=0000   0 US=00F7FAE6 SS=00C8FEE0 D0=1 P#=0008
D0=0000000A D1=00000009 D2=0000FFFF D3=00000001
D4=00000004 D5=00CC9054 D6=00F7080C D7=00CC91E2
A0=A0220E1A A1=00CC91E2 A2=00CE004C A3=00CC9042
A4=00CC9042 A5=00F7FC4A A6=00F7FC4A A7=00F7FAE6

CHK RANGE ERROR in process of gid     8
SI =        0  pc =   131118
 saved registers at 13369278
Going to Lisabug, type g to continue
>
```

You are now in LisaBug and can issue LisaBug commands. For example, to see the program type "IL RANGE_TE" and the following appears:

```
>IL RANGE_TE
RANGE_TE+0000 4E56 0000      RANGE_TE LINK A6,#$0000
RANGE_TE+0004 2C5F                    MOVE.L (A7)+,A6
RANGE_TE+0006 4E55 FFEA               LINK A5,#$FFEA
RANGE_TE+000A 9FED 0010               SUBA.L $0010(A5),A7
RANGE_TE+000E A022 0238               IUJSR $00220238          ; 00220238
RANGE_TE+0012 422D FFEB               CLR.B $FFEB(A5)
RANGE_TE+0016 601A                    BRA.S *+$001C            ; 0002003E
RANGE_TE+0018 102D FFEB               MOVE.B $FFEB(A5),D0
RANGE_TE+001C 4880                    EXT.W D0
RANGE_TE+001E 41BC 0009               CHK #$0009,D0
RANGE_TE+0022 E340           PC       ASL.W #$1,D0
```

```
RANGE_TE+0024 1220 FFEB           MOVE.B $FFEB(A5),D1
RANGE_TE+0028 4881                EXT.W D1
RANGE_TE+002A 3B81 00EC           MOVE.W D1,$EC(A5,D0.W)
RANGE_TE+002E 5220 FFEB           ADDQ.B #$1,FFEB(A5)
RANGE_TE+0032 0C2D 000A FFEB      CMPI.B #$000A,$FFEB(A5)
RANGE_TE+0038 6FDE                BLE.S *-$0020          ; 00020024
RANGE_TE+003A A022 0290           IUJSR $00220290        ; 00220290
RANGE_TE+003E 4E5D                UNLK A5
RANGE_TE+0040 A022 0220           IUJSR $00220220        ; 00220220
RANGE_TE+0044 4E75                RTS
RANGE_TE+0046 4E5E                UNLK A6
RANGE_TE+0048 4E75                RTS
```

This assembly listing shows in line RANGE_TE+0022 that the program crashed with a range error (CHK RANGE ERROR) since variable i was greater than 9, the last index in array a.

LisaBug can also be invoked at any time by pressing the "—" key on the keypad. This is useful when a program is in an infinite loop situation and nothing will make it stop. Once in LisaBug you can terminate the errant program and return to the WorkShop main command line.

For Macintosh programmers Apple developed the WorkShop Macintosh Software Supplement. This consisted of a set of Lisa diskettes containing Macintosh ToolBox and OS interfaces which allowed total access to every Macintosh feature. Various Macintosh oriented tools for the WorkShop were also provided. This included a resource editor and a resource compiler. For more advanced Macintosh programming Apple released several preliminary versions of MacApp, an object-oriented programming environment centered around the Object Pascal language. MacApp was based upon Apple's work with ToolKit/32, an ambitious object-oriented system based upon Clascal, the predecessor to Object Pascal. ToolKit/32 and Clascal were never supported by Apple since when both of these were near completion Apple diverted its resources toward the Macintosh and away from the Lisa.

EXEC FILES

The program development cycle for a typical application involves a lot of typing. To ease programmers of this burden the WorkShop allows command line input to come from special files, called Exec files. These files are similar to Apple's older /// Pascal Exec facility, but the Lisa version allows Exec files to be created with the editor. The Lisa Exec facility evolved into a very sophisticated Pascal-like language that supports variables, boolean expressions, loops, comments, and file I/O. A simple example follows which compiles two units & a main program, links these items, and runs the final executable file:

```
$EXEC
$
$SET %0 TO 'CARTOG/Map_Util'    { main program file }
$SET %1 TO 'CARTOG/UNewMapMgr'  { new map file manager unit }
$SET %2 TO 'CARTOG/UOldMapMgr'  { old map file manager unit }
$
P{ascal} %2                   { compile the old map manager unit }
{}
{}
P{ascal} %1                   { compile the new map manager unit }
{}
{}
P{ascal} %0                   { compile the main program }
{}
{}
$
L{ink}   %0                   { link the main program }
%1                { link the new map manager unit }
```

```
%2              { link the old map manager unit }
Lisa/UVM        { link my virtual memory unit }
Lisa/UDebug     { link my runtime debugger unit }
IOSPasLib       { link the Lisa Pascal runtime unit }
{}
{}
%0              { send Linker output to the main program }
$
R{un}    %0  { run the main program }
$
$ENDEXEC  { That's all, Folks ... }
```

LANGUAGE UTILITIES

The WorkShop contains many useful utilities which assist programmers greatly. These utilities disassemble compiled programs, edit files at the block level, generate program cross-references, etc... The following table summarizes the more important utilities:

CodeSize	shows the code sizes and names of program segments
DumpObj	disassembles compiled program code
DumpPatch	edits files at the block level
FileDiv	divides large files into smaller files
FileJoin	joins files divided by FileDiv
Find	searches text files for strings
MacCom	transfers files to and from Lisa and Macintosh microdiskettes
PasMat	formats Pascal source files using many options
ProcNames	shows the names and lexical levels of Pascal program routines
RMaker	compiles a resource file for Macintosh programs
SegMap	shows the code segments in a program
ShowInterface	shows the interface portion of a Pascal Unit object file
UXRef	shows the routine relationships in a compiled program
XRef	shows the variable/routine cross-references in a source file

Two other features make the WorkShop well rounded. The **TransferProgram** command in the main command line runs a mouse- and window-based telecommunication's program. This is ideal for accessing a programming BBS and downloading source code. The **MakeBackground** command allows programs to be run as a background process. With this feature you can run several programs at once, but with several programs running concurrently system performance can become degraded.

CONCLUSION

For the past decade I have worked with many different computers, from programmable hand-held calculators to mainframes and from 1984 onward I have used a Lisa for programming mainly in Pascal. Overall, I have found the Lisa WorkShop to be a very powerful development system for creating, compiling, and debugging large programs. Even when compared to the other machines that I have worked with the Lisa WorkShop still comes out as being both a professional and powerful development environment.

<<< That's all, Folks ... >>>