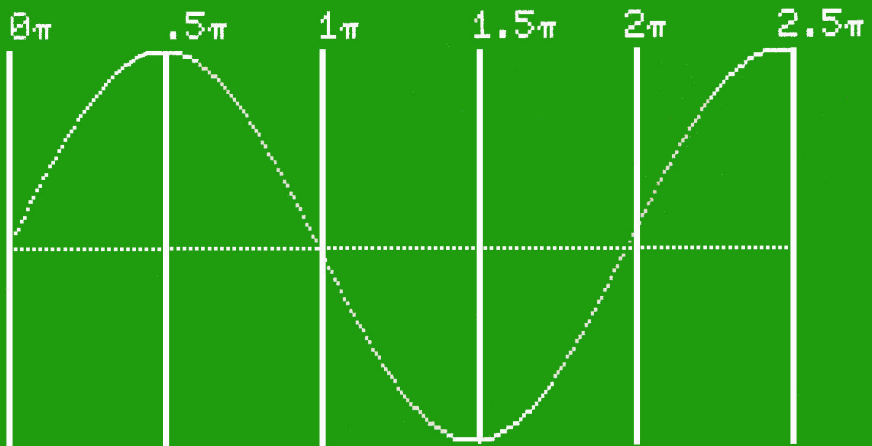# apple® software bank

CONTRIBUTED PROGRAMS
VOLUMES 3-5

BONUS ISSUE

Sine function Y=sin(X) from 0π to 2.5π.

BONUS ISSUE

NOTE: In this BONUS ISSUE we have put together some unusually
        good programs which we believe will be of particular
        value to our users. Because of the fine quality of these
        programs, we felt they warranted more extensive documentation
        than is normally included in our User Contributed Software Bank
        Publications.

TABLE OF CONTENTS

APPLE SOFTWARE BANK CONTRIBUTED PROGRAMS

APPLE Computer Inc. is happy to present a broad selection of Contributed
Programs to please you and your APPLE. Most of these programs were written
by APPLE owners and submitted to the APPLE SOFTWARE BANK. To keep the cost
to a minimum, and to provide the widest selection of programs possible,
these Contributed Programs are not supported by APPLE COMPUTER INC. That
means that APPLE and its dealers will not correct any errors that the
authors might have made, or provide information beyond what is presented in
this document.

You can obtain these Contributed Programs from any APPLE dealer. A
Contributed Program release is made about every two months. The programs in
each release are on diskettes which are distributed to the dealers.

It works like this. Bring your own diskette or tape to your dealer (or buy
one there), and choose the programs you want from this catalog of programs.
You LOAD one of the desired programs into the dealer's APPLE, and then SAVE
it on your disk or tape. If you save the programs on cassette tape, it is
often a good idea to bring your own tape recorder to the dealer's showroom,
to assure compatibility of the resulting cassette. Compatibility is not a
problem with diskettes. Your dealer can show you how to LOAD and SAVE
programs if you are unfamiliar with the procedure.

Additional copies of this document are available at your dealer.

HOW TO OBTAIN PROGRAMS FROM THESE VOLUMES

This section tells you how to transfer individual items from the dealer's
diskette to your diskette or cassette. You may also copy the dealer's
entire diskette to your diskette, if you so desire.

Boot the Software Bank disk in Drive 1 of the dealer's system. If there is
only one drive, it is Drive 1. If you don't know how to use Disk II, see
the DISK II manual or ask the dealer to operate the system. When the disk
is booted, the following message should appear:

APPLE SOFTWARE BANK
CONTRIBUTED PROGRAMS: VOLUME <volume number>

You can type the command

CATALOG

to see the list of available programs. If the list is too long to fit on
the screen, the prompt character (>) will not appear; press the space bar to
see the rest of the catalog.

The programs HELLO, COPY and COPY.OBJ are not part of the offering, although
there is no objection to your copying them.   As you could tell by LISTing
it, the HELLO program is just two statements. The COPY programs are
supplied with every disk drive, so that if you have DISK II, you already
have the COPY programs. The program APPLESOFT is the standard version of
APPLESOFT BASIC that is supplied with every DISK II.

1

When you have read the catalog and have made your first selection, you should type

LOAD <program name>

You must type the program name exactly as it is shown in the catalog.

When the program is LOADed, a process that takes only a few seconds (1Ø seconds for a very long program), you may transfer it to your tape or diskette according to the instructions below. If the program is SAVEd immediately following LOAD, it will RUN fine in any APPLE that does contain an APPLESOFT card. However, if the program that you just LOADed is in APPLESOFT, but the dealer's computer you are using does not have an APPLESOFT card, you must type the command

CALL 3314

before RUNning or LISTing the program on this computer. You can tell that a program is in APPLESOFT if its name is preceded by an "A" in the catalog.

If the computer from which you SAVEd the program did not have an APPLESOFT card and you typed CALL 3314 before SAVEing it, the program will run fine on an APPLE that does not contain an APPLESOFT card. However, it will be necessary to type

CALL 54514

before you can RUN or LIST the program on an APPLE that is equipped with an APPLESOFT card.

TRANSFERRING PROGRAMS TO TAPE OR DISKETTE

a) TAPE

Once the desired program has been LOADed:
Place a cassette in your recorder, and press the RECORD and PLAY levers simultaneously. On the APPLE, type the command

SAVE

and then press the APPLE's RETURN key. The program will commence being saved on cassette tape. Saving is complete when the cursor and prompt character return.

b) DISK (One-drive systems)

Once the desired program has been LOADed:
Remove the Software Bank diskette from the drive, place your diskette in the drive, and type

SAVE program name, VØ

In a few seconds, the program will have been transferred to your diskette.

c) DISK (Two-drive systems)

Once the desired program has been loaded:
Place your diskette in drive 2, and type

SAVE program name, VØ, D2

In a few seconds, the program will have been transferred to your diskette.

On a two-drive system, the entire Software Bank diskette may be quickly
copied at once. Any information stored on your diskette will be lost.
Type

RUN COPY

and follow the instructions on the screen.

NOTE: RUNNING THE INTEGER BASIC PROGRAMS

Many of the Integer BASIC programs in these volumes have LOMEM:s and HIMEM:s
that are set internally by the programs themselves. If you are using a
cassette system, and you receive a ***MEM FULL ERR when LOADing a program
that should fit, press the RESET key and type

ctrl B   (while holding down the CTRL key, type B)

Then try LOADing the program again.

If you are using a disk system, and you receive a ***DISK: PROGRAM TOO LARGE
ERROR when you are LOADing a program that should fit, type

LOMEM:2Ø48

Then set the proper HIMEM: according to the following chart:

IF YOUR SYSTEM SIZE IS	YOU SHOULD TYPE
16K	HIMEM: 5632
2ØK	HIMEM: 9728
24K	HIMEM: 13824
32K	HIMEM: 22Ø16
36K	HIMEM: 26112
48K	HIMEM: -27136

Then try LOADing the program again.

The following pages contain concise descriptions and operating instructions for each of the programs on the diskette. Have fun.

The purchaser of any of these programs accepts and uses them AT HIS OR HER OWN RISK, in reliance solely upon his or her own inspection of the program material and without reliance upon any representation or description concerning the program material.

Program Name: INTERCEPT
Volume Number: 3
Software Bank Number: ØØ137
Submitted By: Jo and Charlie Kellner
Program Language: Integer BASIC
Minimum Memory Size: 12K Bytes

You say it's early evening, and you want to just sit back and relax?
Saturday afternoon, and the living is easy? Late at night, and you've just
made your cocoa? THEN DON'T PLAY INTERCEPT!

BUT, if you've had your shots, and you're not afraid of uncrated Bleedles,
then have we got a game for you! INTERCEPT, the game that pits you, the
person, against APPLE, the phantasmagorical game machine, in a game so
ridiculously easy it can be played by a 1Ø year old! (An 11 year old will
have some difficulty; a 23 year old will find it highly troublesome; if
you're over 3Ø, we recommend Whist.) Just flex your flibberty digits and
plow right in. Soon you'll be racking up points at a rate that will make
your head spin. (Unfortunately, head-spinning is against the rules and will
cause you immediately to forfeit the round.) With a little practice, you
can be INTERCEPT champion of your neighborhood, your community, your entire
block! Now, put down that cocoa and let's get cracking!

INSTRUCTIONS

LOAD this program in Integer BASIC and then RUN it. You will perceive a
display which indicates the number of degrees Kellner and poses the burning
question, "INSTRUCTIONS?"

Hustle is a dynamic chase game. The object of the game is to accumulate
points by intercepting colored blocks as they appear at random on the
playing field. You intercept the blocks by directing the head of a line
which is constantly growing on the screen. The direction of your line's
growth can be controlled from the keyboard using the keys "U" for up, "D"
for down, "L" for left, and "R" for right. If you wish to exit the game
during a round in progress, press ESC. Any other key will be ignored. Do
not press RETURN after typing the letter for your direction; APPLE will be
reading your entry on the fly.

Scoring is determined by the color of the block you intercept with your
line; the value for each color block is shown on the scoreboard. Caution:
the gray blocks have a variable value and may at times decrease your score!

A round is ended when your line hits either the border or itself, so be sure
you neither turn back on yourself nor box yourself in. A game consists of
five rounds. The scoreboard displays the score of the current game as well
as the high score for the day.

At the end of a game, or upon pressing ESC, you will be asked if you wish to
play another game. Just press RETURN to indicate a yes answer; APPLE will
adjust the skill level of the next game depending on your current score.
Answering "N" or "NO" will end play. If another person wishes to play,
answer "NO" and then RUN the program again to restore initial skill level;
high game score will remain in memory and is not cleared by RUNning the
program again.

PROGRAMMER'S CORNER

The professionalism displayed by the Kellners in this program is stunning.
It is a model game, both in terms of game theory and construction. If you
are interested in writing games or simply wish to study good programming
technique, INTERCEPT is an ideal subject. Because of speed constraints, REM
statements have been used sparingly. Therefore, we offer the following
augmentation. Play the game before analyzing it, then LIST the sections
surrounding the lines as you read about them. In general, list by
increments of 50 lines. The analysis will note where there are gaps in
numbering.

Start of program:

40	POKE -16298,0 insures that high-resolution graphics mode is off; if the last program used high-resolution graphics, GR might turn on the high-resolution graphics display.
45,46	A subroutine used to make screen white, for printing the directions. The PRINT statement on 46 is one space less than that of 45 so line 24 won't scroll. POKE 2039,32 pokes a white space into the last screen position.
55	Determines if game has been played since power-up, by seeing if a flag has been set by line 56,
56	which POKEs two arbitrary numbers, 163 and 12, into memory.
57	Zeros the HI-GAME score locations if above flag has not been set (first game of the day).
60	Fills matrix C with block colors.
70	Selects valid keyboard characters.
80	POKEs tone routine into memory : opening sound : titles.
100,195	Sets up playing field and score-board.
171, 175	See notes for line 9020.
182,184	Decides first direction that line will move. Either DX1 or DY1 will be +1 or -1; whichever is not will be 0. They are then added to X1 and Y1 to extend the line in the chosen direction.
190,195	Initializes variables described elsewhere at the start of each round.
200,290	The main program loop, off of which the various subroutines branch.
230,240	Increments the speed of the line periodically.
250,270	Decides whether to draw or erase one of the three possible blocks. "IF NOT RND(DB) THEN...." is the same as saying, "IF RND(DB) = 0 THEN...." branch to the block-drawing routine.

There are no lines from 300 to 499.

6

500,590	These form one of the three block drawing routines.
510	Tests whether the color of block (CB) is 0, or black. If the color is not black,
520	will set it to black and jump to the draw routine. If the color is black,
530	will set it to a color not being used (not equal to CC or CD). It will also set the odds for line 250 to 100 to 1. (Line 520 will set the odds of drawing a new block to 8 to 1.)
560	Finds a random location for the new block and then checks to see if it is occupied. If so, it keeps searching.
570	Draws the new block or erases the old.
580	Jumps to a tone routine if flag T1 <> 0.
600,690, 700,790	Virtually identical subroutines for the other blocks.

There are no lines from 800 to 899.

900,990	End of game routine.
915,916	Changes the HI-GAME score if the new score is higher.
916	Sets the user's level of expertise: "EX=SC(ore)/32" so that if another game is played, the initial speed of the lines will be higher, based on the player's skill. See line 195

There are no lines from 1000 to 2099.

2100,2155	Checks the keyboard to see if a valid key has been pressed. See line 70. KE=155, the ASCII number for ESC.
2160	Sets new position. See notes for lines 182,184.
2170	Checks new position to see if screen is black. If not, the program branches to line 2500. Otherwise, line 2180
2180	plots the new position and line 2190
2190	returns to main loop.

There are no lines from 2200 to 2499.

2500	Program gets here by line striking an object: a block, the border, or the line itself.
2510	If the color of the screen (CS) where the line is about to move is equal to the color of a block, then go to scoring routines. Otherwise, lines 2580 to 2590
2580,2590	increment the Round and end the game if there have been five Rounds.
2600,2690	These are the scoring routines.
2610	See notes for line 60. If the screen color not gray, "I" will be left equal to that color's score/100. If the color is gray, "I" will equal a number between −5 and 10.
2620	First sets the flag, NI (negative I), equal to "true", or 1, if I<0. If I was positive, then it is false that I<0 and NI will be 0. The second statement will leave I a positive number.

2640   Adds 1ØØ to the score (see notes on line 9ØØØ) for
     every unit of I. Flag T1 is now shut-off.
266Ø,2675 Erases the block that has been struck without
     making any sound because (see line 58Ø) T1 is off.
268Ø   Plots the new line position, and line 269Ø
269Ø   returns program to main loop.

There are no lines from 27ØØ to 7999.

List all at once from 8ØØØ to 8999.

8ØØØ,88ØØ These are the audio routines. Read REM on line 32ØØØ.
831Ø   The routine which "bleedles," or jumps the dot
     all over the screen at the beginning of the game
     prior to its settling down and forming a line.

88ØØ   The actual call used to make the sounds. POKE Ø,P
     sets the pitch; POKE 1,D sets the duration. The
     routine is documented in the <u>Apple II</u> <u>Reference</u>
     <u>Manual</u>.

List all at once from 9ØØØ, 9999.

9ØØØ,9Ø9Ø These are the scoring routines.
9Ø1Ø   See notes for lines 262Ø and 264Ø. SC stores the score
     in the current game; if NI = 1, then each time line
     9Ø1Ø is accessed, 1 will be subtracted from the score.
     Otherwise, 1 will be added. Then, line 9Ø2Ø
9Ø2Ø   prints out the score. Right-justification is provided
     by tabbing over one position for every "truth". (SC<1ØØØ,
     SC<1ØØ, etc.) After printing SC, two zeros are printed,
     thus making scores far larger than 32767 possible.

At 1ØØØØ, begin listing by 5Ø lines again.

Lines 1ØØ2Ø,1ØØ4Ø contain the title. While this kind of titling does
require careful planning, it is not as hopelessly difficult as it seems:
After typing PRINT, type spaces over to the very last column and print your
quotation mark; this will put the cursor at the first position on the next
line. You may then type your titles in the actual screen positions where
they will appear when PRINTed. If you are careful with the first line, the
rest are easy. But don't LIST until you are through, or you'll be staring
at gibberish.

The balance of the program contains the directions and, at line 32ØØØ, the
machine language POKEs for the sound.

If you are just learning programming, analyzing properly-written programs
such as this is an excellent learning tool. After you understand the
program, try playing around with it. Try making the block stay on the
screen for a time that is relative to its point-value. This would be done
on lines 54Ø, 64Ø, and 74Ø. How about making the program speed up and slow
down randomly? This project is a little harder; you'll have to really
understand lines 195 and 22Ø,24Ø.

Why not change the rules? Make it legal to turn back on yourself. Make colors other than gray (perhaps randomly selected) capable of making you lose points.

Use your imagination and have fun; one of the real joys of programming is its inherent creativity. When you've played with INTERCEPT for a while, write a program of your own; you'll find yourself using your new-found knowledge to solve programming problems you never had before -- because you'll be doing things you never realized could be done.

Program Name: AIRFOIL
Volume Number: 3
Software Bank Number: ØØ399
Submitted By: J. Raskin
Program Language: APPLESOFT II
Minimum Memory Size: 16K Bytes with ROM card, 24K Bytes without ROM card

In the 193Ø's, the NACA (National Advisory Committee on Aeronautics)
developed a series of aircraft wing sections called the "four-digit" wing
sections. These airfoils have been used and are still being used on many
aircraft, both prototypes and small-scale flying models. Performance data
for these airfoils is readily available.

AIRFOIL is a program that generates NACA four-digit wing sections, given
three parameters that determine the curvature and thickness of the wing.
The wing section is then displayed on the APPLE's high-resolution graphics
screen.

The formulae and definitions used in this program can be found in Chapter 6
of Abbott and Von Doenhoff, Theory of Wing Sections, Dover
Publications, N.Y. 1959.

INSTRUCTIONS

LOAD the program in APPLESOFT II and RUN it. You will be asked to specify
three parameters described below, and then the program will draw the wing
section you have defined, on the high-resolution screen.

First, a very brief definition of terms is in order. If a straight line is
drawn from a wing section's leading point to its trailing point, the length
of this line is the wing section's "chord". The "meanline" of a wing
section also connects the leading point and the trailing point, but lies
halfway between the upper surface and the lower surface. "Thickness" is the
distance between the upper surface and the lower surface. "Camber" is the
maximum deviation of the meanline from the straight chord-line.

A wing section in the NACA 4-digit series is described by three numbers:

1) The amount of camber of the meanline, expressed as a percentage of the
wing's chord. Since most practical airfoils have less than 1Ø% camber, a
single digit suffices to specify this parameter.

2) The position of the highest point in the camber, measured from the
wing's leading edge and expressed as a percentage of the wing's chord. The
position of maximum camber can vary from the leading edge of the wing to the
trailing edge. Since small differences in this position are relatively
unimportant, a single digit can be used to specify maximum camber positions
from 1Ø% to 9Ø% of the wing's chord.

3) The maximum thickness of the section, expressed as a percentage of the
chord. Practical maximum wing thicknessess typically vary from 2% to 25% of
the wing's chord, with a difference of a few percent being quite important.
Therefore two digits are required to specify maximum thickness.

For example, an airfoil called an NACA 6412 would have
1) 6% maximum camber at
2) 40% of the chord from the leading edge of the wing.
The maximum thickness would be
3) 12% of the chord.

Thus, given any NACA four-digit airfoil, you can find the proper constants
to operate the program. To allow scaling of the wing for different chords,
you can specify a non-standard drawing. In this case you will be asked to
give the chord, expressed as a percentage of screen width. The program
cannot give absolute sizes since it has to operate with many different size
television screens. You can also specify the number of points to be
plotted. Use a small number if you want a quick plot, and a large number if
you want to build a wing section from the plot.

One technique that modelers might find useful is to put a piece of paper on
the TV screen and trace the wing sections. The unevenness caused by the
digital nature of the plot can be remedied by a bit of filing or sanding of
the template or rib constructed from this program.

Program Name: MICROLISP
Volume Number: 3
Software Bank Number: ØØ398
Submitted By: Ole Anderson
Program Language: APPLESOFT II
Minimum Memory Size: 32K Bytes

The following discussion assumes the user is already familiar with the
principles and technical vocabulary of the LISP language.. The user should
consider this document more as reference material on how MICROLISP differs
from other versions of LISP, and what features it implements. A good
reference for learning LISP is Weissman's LISP 1.5 Primer (see
Bibliography). We also can recommend Friedman's The Little LISPer as an
enjoyable introduction.

MICROLISP is a cassette-based microprocessor implementation of an
interpreter for LISP, a programming language useful for symbol processing.
The MICROLISP interpreter is written in APPLESOFT II BASIC. More complete
documentation will soon be available when Hayden Press publishes a book on
Microlisp by Ole Anderson. This book will include an annotated source
listing of the interpreter, an explanation of its theory of operation, a
detailed reference manual, an application section describing how to do
graphics from MICROLISP using BASIC, a short LISP primer for the
inexperienced user, and a bibliography of LISP literature useful to the
MICROLISP user.

Many programming languages (BASIC for example) have been created to deal
primarily with numerical data, and programs written in those languages can
solve a large range of problems. But there is an equally interesting area
of programming that deals with symbolic data.

If you want to be "in" with the artificial intelligence crowd, you might
consider learning a symbolic manipulation language. Symbolic data might be
chess positions, equations (treated as equations, not as expressions to be
evaluated), or natural-language sentences. LISP is a language for list
processing (hence the name: LISt Processor). Lists are the data items of
LISP, just as numbers are the data items in BASIC. Because list elements
are symbols not restricted to numbers, LISP makes it possible to solve many
types of symbolic manipulation problems as easily as simple numeric problems
are solved in BASIC. LISP is used primarily for artificial intelligence
applications such as processing natural-language input, performing symbolic
differentiation and integration of numerical formulas, and proving
mathematical theorems. MICROLISP is a version of LISP that will allow you
to learn the elements of LISP by writing and executing LISP programs.

BASIC goes a long way toward making the power of the personal computer
available to the average person, but it lacks several critical features
found in many other high-level languages: naming of subroutines and calling
them with parameters, use of variable names with more than the first two
characters significant, recursive subroutines and dynamic allocation of
variables. These features are all found in MICROLISP.

Implementing MICROLISP in BASIC means that LISP programs do not execute very
rapidly. However, it makes it possible to see how the language has been
implemented. On the other hand, APPLESOFT II supports real (floating point)
arithmetic, string functions and trigonometric functions, and these are all
available to the MICROLISP user (often they are not available in larger
versions). Another convenient feature in APPLESOFT II is the ability to

12

PEEK and POKE memory locations, and to CALL machine-language routines from BASIC. This is often useful for interfacing special hardware and for applications where speed is critical.

Another reason MICROLISP was written in BASIC is for tutorial purposes. Many personal computer users desire to learn about the techniques involved in implementing high-level computer languages. This program is an example of a complete language processor which accepts characters typed from the keyboard, builds symbols from them, and processes these symbols according to the syntax of the LISP language. Thus, it uses many of the important algorithms employed by all other interpreters, compilers and language processors. MICROLISP can also be used as an educational tool for introducing elementary list-processing concepts.

PROGRAMMING WITH THE MICROLISP MONITOR

LOADING AND RUNNING THE INTERPRETER

LOAD and RUN MICROLISP in APPLESOFT II. MICROLISP will respond by displaying this message:

MICROLISP/ <version date>
COPYRIGHT 1978 APPLE COMPUTER INC
LIST ELEMENTS= <list elements>

The first line indicates the current version date of MICROLISP. The third line tells how much space MICROLISP has reserved for list structures, a number dependent on how much memory is available at initialization time. MICROLISP stores most of its data as list elements.

The user must then wait while initialization is completed. This time depends on how much memory is present and may take as long as 2 minutes. When initializing is complete, MICROLISP displays its prompt character:

?

At this point, MICROLISP is ready to accept commands.

In MICROLISP, all programs and data are in the form of symbolic expressions called s-expressions. Only memory constraints limit the length of s-expressions, and they have a tree structure which gives MICROLISP its power in handling different types of symbolic data.

The most elementary type of s-expression is an atomic symbol or atom. Atoms may be numeric or non-numeric. In MICROLISP, numeric atoms are real numbers, like real numbers in APPLESOFT II. Non-numeric, or literal, atoms are made up of a string of characters not containing right or left parentheses " () "or dots " . " or spaces " " as these have special meanings in LISP, and not beginning with a digit or a minus signum " - " as these begin a numeric atom. Because MICROLISP literal atoms (or literals) are implemented as BASIC strings, the maximum length of each is 255 characters.

Non-atomic s-expressions are written in either <u>dot notation</u> or <u>list notation</u>. They are built of atomic symbols and the four special characters: left parenthesis, right parenthesis, dot, and space. An s-expression written in dot notation is either an atom, a <u>dotted pair</u> of atoms written as

(\<atom 1> . \<atom 2>)

 or a dotted pair of s-expressions written as

(\<s-expr1> . \<s-expr2>)

where \<atom1> and \<atom2> are any atoms and \<s-expr1> and \<s-expr2> are any s-expressions. List notation is a version of dot notation invented to simplify the reading and writing of s-expressions. For example, the s-expression

(A B)

in list notation is equivalent to the same s-expression

(A . (B . NIL))

written in dot notation. The atoms A and B are called the <u>elements</u> of the list (A B)

When we talk to the MICROLISP interpreter, we are communicating with a supervisor program which accepts symbols typed from the keyboard and processes them according to certain rules. MICROLISP is a version of LISP 1.6, in which the supervisor, EVAL, expects a single s-expression input, expressed in either dot or list notation. (The LISP 1.5 supervisor, EVALQUOTE, differs in that it expects two inputs: a function name and an argument list for the function. The two supervisors are equivalent in computational ability and differ only in syntax.)

PREDEFINED PROPERTY LISTS

This section describes each identifier which exists in MICROLISP after initialization of the property lists has occurred. Six identifiers have special meanings when they occur on property lists: VALUE, SUBR, FSUBR, EXPR, FEXPR, LAMBDA. Property lists can be dynamically defined or removed by the functions DEFPROP and REMPROP.

A VALUE is an identifier which has an s-expression on its property list with the property name, VALUE. VALUEs are evaluated by returning the s-expression property. The two identifiers with predefined VALUE property lists are T and NIL.

A SUBR is an identifier which has a number on its property list with property name, SUBR. SUBRs are evaluated by first evaluating their arguments and passing the results to the corresponding internal interpreter subroutine.

An FSUBR is an identifier which has a number on its property list with property name FSUBR. FSUBRs are evaluated by passing the unevaluated actual argument list to the internal MICROLISP routine.

An EXPR is an identifier which has a LAMBDA expression on its property list with property name EXPR. EXPRs are evaluated by binding the values of the actual arguments to their corresponding dummy variables. If there are more actual arguments than dummy variables, the excess arguments are evaluated but ignored. If there are more dummy variables than actual arguments, an error will be generated.

An FEXPR is an identifier which has a LAMBDA expression on its property list with the property name FEXPR. FEXPRs are evaluated by binding the actual argument list to the single dummy variable without evaluating any arguments. Only one argument is allowed for an FEXPR; more than one will generate an error.

The form, (LAMBDA "ARGLIST" "BODY") defines a function by specifying an argument list, which is a list of identifiers which are to serve as dummy variables, and a body, which is an s-expression. LAMBDA expressions are evaluated by binding actual arguments to the dummy variables, then evaluating BODY with the current dummy-variable bindings.

PREDEFINED FUNCTIONS

Each function below is presented with its parameter-calling order. Following the conventions of many LISP manuals, double quotes (") surrounding an argument to a functional s-expression mean that the argument is implicitly quoted. Otherwise, the arguments are evaluated before being passed and used.

(AND P1 P2 ... Pn)
Returns the value of Pn if all Pi are non-NIL; otherwise, it returns NIL. AND scans its arguments from left to right until either NIL is found, in which case the remaining arguments are not scanned, or until the last argument is scanned. Note that (AND) returns T.

(ATOM X)
Returns T if X is not a list; i.e. if X is either an identifier or a number. Otherwise, the value of ATOM is NIL.

(CAR X)
Returns the first half of the dotted pair. If X is a list, this results in returning the first element of X. The CAR of an atom is illegal.

(CDR X)
Returns the latter half of the dotted pair X. If X is a list, this results in returning the list X with its first element removed. The CDR of an identifier returns the property list of that identifier. (CDR is pronounced "couder".)

(COND ("P1" "V1")
      ("P2" "V2")
      ...
      ("Pn" "Vn"))
In this call, the Pi are considered to be predicates which evaluate to truth values. They are evaluated consecutively until the first is found whose value is non-NIL. Then the corresponding Vi is evaluated and its value is returned as the value of COND. If no Pi is non-NIL, then COND issues a warning message and returns NIL.

15

(CONS X Y)
Creates a dotted pair with left half X and right half Y. If Y is a list,
this amounts to returning the list Y with X inserted as the first element.

(CONSP X)
Returns X if X is not an atom; otherwise, NIL.

(DEFPROP "ID" "PROPNAME" "PROPERTY")
Enters the property name, PROPNAME, with the property value, PROPERTY, into
the property list of the identifier, ID. The new property name and its
value are placed on the beginning of the property list. DEFPROP returns ID.
 Improper use of DEFPROP may result in two or more different values for
identical property names, in which case the first takes precedence. To
replace a property, delete it first with REMPROP. (The parameter order
differs from most other implementations of LISP.)

(EQ X Y)
Returns T if X and Y are the same pointer of internal address. Identifiers
have unique addresses and therefore EQ will be true if X and Y are the same
identifier. EQ cannot be used to compare equivalent floating point numbers.
 For non-atomic s-expressions, EQ is T if X and Y are the same pointers.

(EVAL X)
Evaluates the value of the s-expression X

(GET PROPERTY ID)
Searches the property list of the identifier ID looking for the property
name, PROPERTY. If such a property name is found, the property name-value
pair is returned as the value of GET; otherwise, NIL is returned. (This
differs from other LISPs in parameter order.)

(GC)
Causes a garbage collection to be performed.

(GO "ID")
Causes a sequence of control within a PROG to be transferred to the next
statement following the label ID. ID must be atomic or an error is issued.
GO cannot transfer into or out of a PROG.

(LOAD)
Restores a previously saved set of property lists from cassette tape. This
also has the effect of restoring the screen memory to the state it was in
when the (SAVE) was performed.

(LITATOM X)
Returns T if X is an identifier; i.e., a non-numeric atom.

(NCONC X Y)
Modifies list structure by replacing the last element of X with a pointer to
Y. The value of NCONC is the modified list X, which is the concatenation of
X and Y.

(NOT X)
Returns T if X is NIL and NIL if X is non-NIL.

(NUMBERP X)
Returns T if X is numeric; NIL otherwise.

(OR P1 P2 ... Pn)
Returns the value of the first non-NIL Pi, or NIL if the values of all the
Pi are NIL. OR scans its arguments from left to right until a non-NIL
argument is found, leaving the remaining arguments unscanned. Note that
(OR) returns NIL.

(PRINT X)
Prints the ASCII representation of the s-expression S and returns X.

(PAIRLIS X Y)
Returns the list of dotted pairs of corresponding elements of the lists X
and Y, in reverse order.

(PROG "VARLIST" "S1" "S2" ... "Sn")
Specifies a list of program variables, VARLIST. (which are initialized to
NIL when the PROG is entered), and a sequence of non-atomic statements and
atomic labels. PROG evaluates its statements in sequence until either a
RETURN or a GO is evaluated or the list of statements is exhausted, in which
case the value of PROG is that of the last statement evaluated.

(PROGN "S1" "S2" ... "Sn")
Evaluates all the expressions Si and returns the value of Sn.

(QUOTE "X")
Returns the s-expression X without evaluating it. An equivalent shorter
form is (´"X")

(READ)
Causes the next s-expression to be read from the keyboard or another
selected input device, and returns the internal representation of the
s-expression. READ guarantees that references to the same identifier are
alike (via EQ).

(REMPROP "ID" "PROPERTY")
Removes the property, PROPERTY, from the property list of identifier ID and
returns ID.

(RETURN X)
Causes the current PROG to be exited with the value X.

(RPLACA X Y)
Replaces the CAR of X with Y and returns the modified value of X.

(RPLACD X Y)
Replaces the CDR of X with Y and returns the modified value of X.

(SAVE)
Saves the current set of property lists on cassette tape. This is performed
by writing the entire memory image onto tape. (SAVE) is reversed by (LOAD).

17

(SET ID X)
Changes the value of the identifier specified by the expression, ID, to X
and returns X. Both arguments are evaluated. Note that SET and SETQ modify
the association list and are, therefore, valid only inside PROGs. DEPROP
should be used at the top level of MICROLISP to set constant values by
modifying property list VALUEs.

(SETQ "ID" X)
Changes the value of ID to X and returns X. SETQ evaluates X but not ID.
SETQ, like SET, is valid only inside PROGs.

FUNCTIONS
The following functions perform the real arithmetic and are the logical
comparison operators that are used in MICROLISP. They correspond to the
same operators in BASIC. Because "-" is used as a unary operator (to sign
or signum numbers), SUB is used for real subtraction.

```
( +   X Y)          add
(SUB  X Y)          subtract
( *   X Y)          multiply
( /   X Y)          divide
( ^   X Y)          raise to the power of
( <   X Y)          less than
( >   X Y)          greater than
( =   X Y)          equals
( <=  X Y)          less than or equal
( >=  X Y)          greater than or equal
( <>  X Y)          not equal
```

BIBLIOGRAPHY

∅. Friedman, D. P., The Little LISPer, Science Research Associates, Inc.,
    ISBN: ∅-574-19165-8

        A delightful, informal book. The notation differs only
        slightly from MICROLISP.

1. McCarthy, J., et al., LISP 1.5 Programmer's Manual, The M.I.T. Press,
    Cambridge, Massachusetts

        McCarthy invented LISP. This book is right from the
        horse's mouth. (Pardon the expression, John...)

2. Minsky, M., (ed.), Semantic Information Processing, The M.I.T. Press,
    Cambridge, Massachusetts, 1968

        What can be done (in one field) with LISP.

3. Weissman, C., LISP 1.5 Primer, Dickenson Publishing Co.,
    Belmont California, 1968

        A useful introduction, more formal than the LISPer,
        and more detailed.

Program Name: SHOOTOUT
Volume Number: 3
Software Bank Number: ØØ221
Submitted By: Andrew Rose,
Program Language: Integer BASIC
Minimum Memory Size: 8K Bytes

The following cover letter was received with this program:

Dear Sirs,
    This is the first time I've submitted. If you think the form is to messy
send it back and a blank one and I will try again. If you don't want the
program just say so and I'll send you my other one. This program was
written by me and my friend who doesn't know how to program wants to learn.

                        Sencirly,   ·
                        Andrew Rose

P.S. I'm going on 12 in March.

P.S.S. You can shot through the cactus when the other guy is out of bullets.

NOTE: This program was written by two people, A. Rose and J. Duggar. J.
Duggar does not have an Apple II, so has agreed to let A. Rose have the
certificate and work it out from there.

INSTRUCTIONS

LOAD the program in Integer BASIC and RUN it. After brief instructions, the
game will begin. The board consists of two guns, two cacti, and 12 bullets
(6 each). Each player uses one game control to maneuver a gun up and down,
firing at the opponent by pressing down on the game control's pushbutton.
The cacti cannot be penetrated as long as the opponent has bullets
remaining.

Each game ends following five successful shots by one opponent. At that
point, APPLE will perform the necessary rituals and ask if you wish to play
again. To continue, press the RETURN key; to end, type an N and press the
RETURN key. The winner will be announced and presented with a medal of
valor.

Program Name: HIGH-RESOLUTION CHARACTER GENERATOR
Volume Number: 3
Software Bank Number: 00405
Submitted By: Christopher Espinosa
Program Language: Demonstration Program -- APPLESOFT II
                  Subroutine Package and Character Set -- Machine Language
Minimum Memory Size: Demonstration Program -- 32 K Bytes
                     Subroutine Package and Character Set -- 16K Bytes

<u>This program generated the illustration on the front cover.</u>

HIGH-RESOLUTION CHARACTER GENERATOR gives your APPLE the ability to mix text
with high-resolution graphics, anywhere on the high-resolution graphics
screen. You can display up to 960 upper and lower-case letters, numbers,
symbols, or special characters in three display modes. You can even define
up to 160 characters of your own, if you wish. The program will work with
Integer BASIC, APPLESOFT II BASIC, or machine language, and it can be put at
any convenient location in memory.

INSTRUCTIONS

At your dealer, LOAD the program in APPLESOFT II BASIC. If the dealer's
computer does not have an APPLESOFT II Firmware Card, type

CALL 3314

and press RETURN. Then RUN the program. The program will BLOAD
(BinaryLOAD) the subroutine package and the character set and then reproduce
the picture on the front cover of this manual. When it is complete, type

PR#0: TEXT

and press RETURN. This will return you to the normal text screen. You may
now SAVE the program as follows:

SAVING ON CASSETTE TAPE

Machine language loading and saving is a little confusing if you have never
done it before. It can be helpful to understand exactly what you are doing.
In a machine-language load command such as

6000.6C00 R

the number 6000 is the hexadecimal starting address for the program, and
6C00 is the program's hexadecimal ending address. The computer can measure
the program's length by subtracting the starting address from the ending
address, and thus knows when the entire program has been saved or loaded.
Machine-language commands use the period much as BASIC uses the comma or
dash in LIST statements (e.g.: LIST 100, 300). The R stands for READ and
means the same as LOAD in BASIC. The W stands for WRITE and is equivalent
to SAVE in BASIC.

You do not have to tell BASIC the length of a program because the program itself contains that information. LOADing a BASIC program causes the APPLE to beep twice: once at the beginning and once at the end. The APPLE beeps twice because the computer LOADs two programs. If you listen to a BASIC program, you will hear a short blip in the continuous tone at the beginning, a blip that coincides with the first beep. The blip is a very short program that tells APPLE how long the main program is. After the blip, the tone begins again, and APPLE, armed with the new information, LOADs the main program into memory. A second beep signals the end of .the main program. Loading a machine-language program, however, causes the APPLE to beep only once, at the end.

First SAVE the demo program by typing

SAVE   (do not press RETURN, yet)

Start the cassette recorder in "record" mode, and press the APPLE's RETURN key. When the flashing cursor returns, stop the recorder (but do <u>not</u> rewind the tape). Press APPLE's RESET key and type

6ØØØ.61ØØ W   (do not press RETURN, yet)

Again start the cassette recorder in "record" mode, and press APPLE's RETURN key. When the flashing cursor returns, stop the recorder (again, do <u>not</u> rewind the tape). Press APPLE's RESET key and type

68ØØ.6CØØ W

Yet again, start the cassette recorder in "record" mode, and press APPLE's RETURN key. When the flashing cursor returns, stop the recorder again. Your cassette now contains all three portions of the program, recorded one portion after the other. The cassette tape may now be rewound back to the beginning.

To LOAD the program back into your APPLE, once you are home, LOAD the first portion (written in APPLESOFT II BASIC) in the usual way. After the second beep, stop the recorder but do <u>not</u> rewind the tape. Press the APPLE's RESET key and type

6ØØØ.61ØØ R

Start the recorder in "play" mode, and press APPLE's RETURN key to load the machine-language "Hi-Res Character Generator" portion of the program into memory. After the first beep, stop the recorder but, again, do not rewind the tape. Press the APPLE's RESET key, and type

68ØØ.6CØØ R

Again start the recorder in "play" mode and press APPLE's RETURN key to load the "Character Table" portion of the program into memory. After the first beep, all three portions of the program are in your APPLE. You may then return to BASIC using ctrl C (type C while holding down the CTRL key) followed by pressing the RETURN key, and proceed to RUN the program.

SAVING ON DISKETTE

After LOADing and RUNning the program in your dealer's APPLE, asdescribed earlier in this section, insert your disk and type

SAVE HI-RES CHARACTER DEMO, V∅

Then type

BSAVE HI-RES CHARACTER GENERATOR, A$6∅∅∅, L$1∅∅

and press RETURN. Finally, type

BSAVE CHARACTER TABLE, A$68∅∅, L$4∅∅

PROGRAMMER'S CORNER

Use and Care of the HIGH-RESOLUTION CHARACTER GENERATOR

The program itself takes up 256 bytes of memory. It also needs a Character Table which uses 1K bytes of memory. You can store these two machine-language program portions in any area of memory you wish. The recommended place for the Character Generator portion of the program is $6∅∅∅, and for the Character Table, $68∅∅.

Load the machine-language portions as explained above. You may now LOAD or write a program in either specie of BASIC that will draw on the high-resolution graphics screen. When you are ready to PRINT a legend, title, or sentence on the screen, type

POKE 54, ∅
POKE 55, 96

From this point on, anything your program PRINTs will be displayed on the high-resolution screen! To return to the normal text screen, type

PR#∅
TEXT

and if you have been using high-resolution graphics Page 2 in Integer BASIC, also type

POKE -163∅∅, ∅

Normal text-screen output will resume for subsequent PRINT statements.

Your program, while it is in the high-resolution graphics PRINT mode, will respond to Integer BASIC's VTAB and TAB, and APPLESOFT's HTAB, VTAB, SPC (X), and TAB (X). HOME and CALL -936 will still move the PRINT position to the upper-left corner of the screen, but will not clear the screen. When you reach the end of the bottom line, the screen will not scroll, but will "wrap around" to the top line. The text window still operates normally.

That's not all it can do.

POKE 973, 255

will change the display mode to Inverse Video, printing black characters,
each on a small white background.

POKE 973, Ø

returns you to Normal Video mode (white-on-black), and

POKE 973, 1

puts you in Exclusive-Or mode, which makes whatever you PRINT show up as the
complement of the screen color at that position (white becomes black, black
becomes white). And finally, typing or PRINTing a ctrl L will clear the
screen to black if you are in Normal or Exclusive-Or mode, or to white if
you are in Inverse mode. To avoid a ***SYNTAX ERR message after typing ctrl
L or any control character not recognized by the Monitor, press the
left-arrow key once before pressing RETURN.

If your program uses high-resolution graphics Page 2, but you are not in
APPLESOFT II BASIC with HGR2 invoked, use

POKE 974, 64

to direct the PRINTing to that page.

POKE 974, 32

will return you to writing on Page 1.

If you would like to use the APPLE DOS while PRINTing characters on the
high-resolution graphics screen, there will be no problem. All DOS commands
are still active in the high-resolution graphics character mode, until
keyboard input is requested. At that point the DOS will turn the
High-Resolution graphics mode off. In order to get around this, RUN this
short program:

```
1Ø PR#Ø : IN#Ø : REM  TURN DOS OFF.
2Ø POKE 54, Ø : POKE 55, 96 : REM  ACTIVATE SCREEN PRINTING.
3Ø CALL 976 : REM  TURN DOS BACK ON AND END PROGRAM.
```

Then you may type a ctrl L to clear the screen.

Now you're in high-resolution graphics PRINT mode but not in a program. Try
LISTing the program a couple of times and watch it wrap around the screen.
Change to Inverse Video mode with a POKE 973, 255 and then clear the screen
with a ctrl L. To return to the normal text screen, type

PR#Ø
TEXT

Voila! Back in the real world. To re-enter the high-resolution graphics Print mode, type

POKE 54, 60
POKE 55, 96

Tight on Space?

If you have only a 16K or 20K system, then you might not want to use up valuable memory space at the normal locations for this program. Don't worry, the program is completely portable. The CHARACTER GENERATOR portion will fit in almost any 256 bytes you happen to have available in memory, provided that the starting address in hexadecimal is an even $100 ($2700, $300, $1200, etc.). And the CHARACTER TABLE will go in almost any 1K cubbyhole that is free, provided that the starting address in hexadecimal is an even $800 or $1000 ($6000, $1800, $4800, etc.). On a small system, using Integer BASIC and the RAM high-resolution graphics routines supplied with the HI-RES Demo tape which load at $C00.$FFF, a good place to put the CHARACTER GENERATOR program and the CHARACTER TABLE is from $1000 to $14FF.

If you are using a cassette, locate the HI-RES CHARACTER GENERATOR program (the second portion, in machine language) immediately following the HI-RES CHARACTER DEMO (the first portion, in APPLESOFT) on the tape. Then load the GENERATOR portion into hexadecimal memory locations $1400 through $14FF by pressing APPLE's RESET key and typing

1400.14FF R

Start the cassette recorder in "play" mode and then press the APPLE's RETURN key. After the first beep, the GENERATOR portion has been loaded, so stop the recorder. Now load the CHARACTER TABLE portion into location $1000 through $13FF. Press APPLE's RESET key, and then type

1000.13FF R

Again, start the cassette recorder in "play" mode and then press APPLE's RETURN key. After the first beep, you may re-enter BASIC.

If you are using the program from diskette, type

BLOAD HI-RES CHARACTER GENERATOR, A$1400, V0
BLOAD CHARACTER TABLE, A$1000

In order to tell the program where you have put the CHARACTER TABLE, you must first PRINT at least one character, then POKE location 972 with the high byte of the TABLE's starting address. In this case, the TABLE starts at hexadecimal address $1000, whose high byte is $10 (16, in decimal). So, type

POKE 54, 0
POKE 55, 20
PRINT " ";
POKE 972, 16

24

Now you may PRINT to your heart's content. To exit the high-resolution graphics PRINT, and return to normal text mode, execute the commands

PR#Ø
TEXT

and to re-enter the high-resolution graphics text mode with the CHARACTER GENERATOR portion of the program located at $14ØØ, execute a

POKE 54, 6Ø
POKE 55, 2Ø

The number which is POKEd into location 55 is the high byte of the CHARACTER GENERATOR program's beginning address. The high byte of $14ØØ is $14 (the first two digits). $14 in hexadecimal is 2Ø in decimal. (A handy chart for HEX - DEC conversions may be found in the Token Table included in the discussion of the ILLEGAL STATEMENT WRITER elsewhere in this document.)

HIGH-RESOLUTION GRAPHICS GETTING FANCY

There are 128 characters available to you in high-resolution graphics PRINT mode. Control characters will PRINT on the screen as special symbols, such as musical notes, card symbols, or math symbols. If you decide that you don't want some of these, you may change the characters to ones of your own liking. The following section explains how to do it.

First, on graph paper, draw a box that is 8 squares high and 7 squares wide. Then draw in the character you want, blackening the squares that are to make up the character and leaving blank those squares that will be background. For example, a "Paragraph" symbol might look like this:
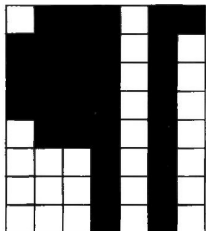
Now you are ready to convert from squares to data. Look at each square in the top row, starting at the right-hand corner square and moving from right to left. Next to that row, write (from left to right as you examine the squares from right to left) a number for each of the seven squares: 1 if the dot is filled and Ø if it is not. When you finish, you should have seven 1's or Ø's in a line next to the row, like this:

1 1 Ø 1 1 1 Ø

25

Do the same for each of the eight rows. You will then have eight rows of
1's and Ø's forming a complete mirror image of the dots and spaces they
represent. Then draw a vertical line separating the numbers into two
columns, the left column having 3 digits, the right column, 4 digits, as
follows:

```
1 1 Ø 1 1 1 Ø
Ø 1 Ø 1 1 1 1
Ø 1 Ø 1 1 1 1
Ø 1 Ø 1 1 1 1
Ø 1 Ø 1 1 1 Ø
Ø 1 Ø 1 Ø Ø Ø
Ø 1 Ø 1 Ø Ø Ø
Ø 1 Ø 1 Ø Ø Ø
```
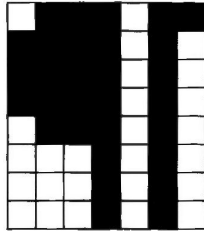
For each of the two groups in each row, convert the three binary digits or
four binary digits into one hexadecimal digit. Write the resulting
two-digit hexadecimal number next to each row:

Conversion Table

Binary Data							Data Converted to Hexadecimal		
1	1	Ø	1	1	1	Ø	=	6	E
Ø	1	Ø	1	1	1	1	=	2	F
Ø	1	Ø	1	1	1	1	=	2	F
Ø	1	Ø	1	1	1	1	=	2	F
Ø	1	Ø	1	1	1	Ø	=	2	E
Ø	1	Ø	1	Ø	Ø	Ø	=	2	8
Ø	1	Ø	1	Ø	Ø	Ø	=	2	8
Ø	1	Ø	1	Ø	Ø	Ø	=	2	8

Bin.	Hex	Bin.	Hex
ØØØØ = Ø		1ØØØ = 8	
ØØØ1 = 1		1ØØ1 = 9	
ØØ1Ø = 2		1Ø1Ø = A	
ØØ11 = 3		1Ø11 = B	
Ø1ØØ = 4		11ØØ = C	
Ø1Ø1 = 5		11Ø1 = D	
Ø11Ø = 6		111Ø = E	
Ø111 = 7		1111 = F	

Hex: Digit 1   Digit 2

You are now ready to enter the data into your Table. First, load the
CHARACTER TABLE into a known location, such as $68ØØ. Then press RESET.
Now, decide which keys you want to press in order to produce the new
character on the screen. Since this is a Paragraph symbol, let's use ctrl
P. In the CHARACTER TABLE, the definition for the character produced by
ctrl @ (ASCII Ø) starts at location $68ØØ, the character definition for ctrl
A (ASCII 1) starts at $68Ø8, the character definition for ctrl B (ASCII 2)
starts at $681Ø, and so on. The APPLESOFT manual has a table giving the
ASCII code for each character.

(In hexadecimal, or base 16, we use 16 digits:
Ø,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. $1Ø in hexadecimal = 16 in decimal. For
each [hexa]decimal place you multiply by 16, not 1Ø as in decimal. So $68Ø8
is 8 bytes less than $681Ø.)

The existing character already defined for ctrl P (ASCII 16) must start at
... $688Ø! That's $1Ø (the hexadecimal equivalent of ASCII 16) times $8
(the number of hexadecimal bytes per character-definition), added to $68ØØ
(the Table's starting address).

26

Press RESET to enter the Monitor, and store your new character definition
into memory starting at location $6880. Type

6880: 6E 2F 2F 2F 2E 28 28 28

Press the RETURN key and you're done. Now you may go back to your program
(use ctrl C, not ctrl B, to re-instate BASIC) and RUN it again. This time,
whenever you PRINT a ctrl P, a paragraph symbol will appear. You may now
SAVE the revised character set.

If you're a real hot-shot, you may create one hundred and twenty-eight
more characters, storing them just after the normal character set
(starting at $6C00). Then, by doing a

POKE 975, 255

you may switch to that character set, in place of the usual set!
POKE 975, 127

will get you back to the regular set. If you want to get really fancy, you
may have any number of character tables in memory and switch between them
simply by POKEing 972 with the decimal equivalent of high byte of the chosen
table's starting address. (The table must start at an even 2K boundary
($800 or $1000), like $6800 or $7000.) With a 48K system, you can have
thiry-five hundred and eighty-five distinct characters to use on both
high-resolution graphics screens!

(The person given the task of checking out this particular claim completed
only 2437 distinct characters before being ushered, while screaming, "ones
and zeros, ones and zeros", into a nice, warm room with soft walls and a
picture of a window. We apologize for the appalling breach in our code of
standards in going ahead with the publication of this claim without further
substantiation -- we couldn't seem to find another volunteer....)

Summary of high-resolution graphics PRINT Mode

To enter high-resolution graphics PRINT mode, type

POKE 54, <addresslow>
POKE 55, <addresshigh>

where <addresslow> and <addresshigh> are the decimal equivalents of the low
and high hexadecimal bytes of the HI-RES CHARACTER GENERATOR program's
starting address. This starting address must end with $00, so <addresslow>
must be 0.

To exit the high-resolution graphics PRINT mode, and return to normal text
mode, type

PR#0
TEXT

and in Integer BASIC when using high-resolution graphics Page 2, also type

POKE -16300, 0

To re-enter the high-resolution graphics PRINT mode, type

POKE 54, 6Ø
POKE 55, <addresshigh>

To change parameters, after at least one character has been PRINTed:

POKE 972, <tablehigh> where <tablehigh> is the decimal equivalent of the
                              high hexadecimal byte of the CHARACTER TABLE's
                              starting address. This starting address must
                              end with either $8ØØ or $ØØØ.

POKE 973, Ø   for Normal Video mode
POKE 973, 255   for Inverse Video mode
POKE 973, 1   for Exclusive-Or mode

POKE 974, 32   for high-resolution graphics Page 1
POKE 974, 64   for high-resolution graphics Page 2

POKE 975, 127   for 128-character set
POKE 975, 255   for 256-character set

Typing a ctrl L will clear the screen and HOME the cursor. Then press the
left-arrow to clear the ctrl L.

Typing TEXT will turn on the regular text screen, but all PRINTing will
continue to occur on the high-resolution graphics screen, which will not be
in view. This mode is to be studiously avoided.

Program Name: APPLE-VISION
Volume Number: 3
Software Bank Number: ØØ1Ø6
Submitted By: Robert J. Bishop
Program Language: Integer BASIC
Minimum Memory Size: 16K Bytes

One never forgets the first time one sees Bob Bishop's APPLE-VISION. While
some are prone to carry away the impression of a light-hearted vaudeville
act, those of a more sophisticated bent will see through to the profound
philosophical implications in the recursive theme of a television within a
television. (This writer is not so bent but rather is reporting the
experiences of those who are.) But regardless of one's position,
APPLE-VISION stands as one of the best computer-animations ever to have been
done on the Apple II.

INSTRUCTIONS

Set HIMEM: to 16384 and LOAD the program in Integer BASIC. Then RUN. If
you wish to SAVE this program on cassette or another disk, you must not RUN
it until after it has been SAVEd. If you LIST the program before RUNning
it, do not be alarmed to see that it is not your usual BASIC, as the first
section is in machine language.

PROGRAMMER'S CORNER

There shall be no attempt to analyze this program; it is very complex and
the major (and most interesting) parts are done in machine code. He would
say, however, that this program shines as an example of finely polished
work. It takes two or three times as long to write a cosmetically perfect
program as it does to knock out one with unclear input requests, ragged
flashes between graphics and text screens, or music scores with
some-notes-a-little-off-but-no-one-will-notice-and-everybody-does.
Flashiness is the mark of a clever programmer; perfection is the mark of a
professional programmer. Bob Bishop's programs consistently display both.

Program Name: INTERNAL-COMBUSTION ENGINE SIMULATION
Volume Number: 3
Software Bank Number: ØØ123
Submitted By: Paul Lutus
Program Language: Integer BASIC
Minimum Memory Size: 16K

In 1876 Otto and Langen invented a new gasoline-powered device known
(euphemistically?) as The Otto Silent Engine. Throughout the last 1ØØ years
it has remained the overwhelmingly dominant choice of power plant for the
world's automobiles.

For generations, American parents have tried to explain the engine's inner
workings to their children, using a combination of halting phrases and
flailing body language worthy of a contortionist, only to find their
children staring at them in intensifying puzzlement. Apple reduces the
strain on both the child's mind and the parent's body by allowing viewers to
look inside a working engine and see the cycles taking place.

INSTRUCTIONS

Set HIMEM: to 16384, LOAD the program in Integer BASIC and RUN it. In a few
moments time, a single cylinder of an automobile-type engine will begin
running in high-resolution graphics animation. To step through the four
strokes in the cycle while following the description below, press the M key
(M for "manual") and then press RETURN repeatedly to move through each
stroke in sequence.

DESCRIPTION

STROKE 1: INTAKE STROKE -- While the inlet valve on the left is open, the
descending cylinder draws fresh gasoline-and-air mixture through the supply
tube on the left from the carburator (the carburator is not shown).

STROKE 2: COMPRESSION STROKE -- With both valves closed, the rising piston
compresses the mixture to a pressure of 8 to 12 times normal atmospheric
pressure. (The ratio between normal atmosphere and the fully-compressed
mixture is called the "compression-ratio" of the engine.)

STROKE 3: POWER STROKE -- at maximum compression, when the volume of the
combustion chamber is at a minimum, the mixture is ignited by the spark plug
(top, center). The mixture burns, and, with the valves still closed, the
pressure of the expanding gases of combustion drives the piston downwards,
powering the engine.

STROKE 4: EXHAUST -- During the final stroke, the exhaust valve on the right
is opened and the piston moves upward, driving the exhausted gases from the
combustion chamber, leaving it ready to begin stroke 1 of the next cycle.

Out of the four strokes of the cycle, only one produces any power. This
makes the cycling of the individual cylinder very rough. But by staggering
the firing of the various cylinders, so that, for example, in a
four-cylinder engine, one cylinder is firing while another is exhausting
while a third is intaking while the fourth is compressing, the overall
engine runs very smoothly.

Program Name: FILE CABINET
Volume Number: 3
Software Bank Number: ØØ4ØØ
Submitted By: J. Apple Sede
Program Language: APPLESOFT II
Minimum Memory Size: 16K -- Requires Disk II

FILE CABINET is a disk-based Information Storage and Retrieval System that
can be used for such diverse applications as:

A Personal Phone Directory:
        -Quickly find business or personal phone
        numbers by Name, Number, or Address!

A Reference Guide For Your Home:
        -Store planting dates and gardening hints
        for seeds and bulbs.
        -Keep track of Birthdays, Anniversarys, and
        other special occasions.
        -Jot down auto or boat service scheduals --
        When they were serviced and how much it cost.
        [Great for tax time!]

Financial Help:
        -Automatically sort and total your checks.
        Keep track of where all that money went!
        -Use it as a simple inventory system with
        Item number, part description, quantity on hand,
        and reorder levels -- search on any key word.

The uses of FILE CABINET roll on endlessly.... Play with it and see!

INSTRUCTIONS

When you LOAD and RUN this APPLESOFT BASIC program for the first time on a
new diskette, you will be asked to name your first DATA BASE file. Give it
a name that will help you remember the contents of the file later. For
example, if you wish to create a list of telephone numbers, you could give
this file the name "TELEPHONE LIST":

        NAME FOR NEW DATA BASE FILE: TELEPHONE LIST

Next you will be asked for the HEADER FOR COLUMN NUMBER 1. As you add
records to your new data base file, the different items of information will
appear in various appropriate columns. At the top of each column is the
column's name, or header. HEADER FOR COLUMN NUMBER 1: is a request to name
the first data column for all subsequent records in this data base. For
instance, you might respond with a column name such as "NAME". You will
continue to be prompted for additional column headers until you press RETURN
as the only response. A possible example is on the next page:

```
HEADER FOR COLUMN NUMBER 1: NAME
HEADER FOR COLUMN NUMBER 2: STREET ADDRESS
HEADER FOR COLUMN NUMBER 3: CITY
HEADER FOR COLUMN NUMBER 4: STATE
HEADER FOR COLUMN NUMBER 5: ZIP CODE
HEADER FOR COLUMN NUMBER 6: PHONE NUMBER
HEADER FOR COLUMN NUMBER 7: COMMENT
HEADER FOR COLUMN NUMBER 8: <press RETURN here, to end column names>
```

You will next be presented with a "menu" of further options:

```
    1   SELECT DATA BASE
    2   SEARCH AND/OR CHANGE DATA
    3   ADD RECORDS
    4   DELETE RECORDS
    5   REPORT
    6   SORT (TAKES APPROX. n MINUTES)
    7   TURN ON PRINTER
    8   TURN OFF PRINTER
    9   LIST DATA BASE
   1Ø   QUIT
```

The following is an explanation of each option in detail:

1 SELECT DATA BASE

This option displays for you a list of the available data base files and gives you the option of selecting an existing data base, starting a new data base, or deleting a data base. NOTE! A data base, once deleted, cannot be recovered!

2 SEARCH AND/OR CHANGE DATA

Here you have a chance to locate Aunt Sally's phone number - or update it to reflect her move to Miami Beach.

SEARCHING:

The APPLE will display a numbered list of catagories under which you might wish to search. The list starts with RECORD NUMBER, and then continues through the various column headers (such as NAME or PHONE NUMBER).

If you want to look at a particular record, and you know its record number, type a Ø. When you press RETURN, the APPLE will ask you to type the number of the record you wish to see.

If, on the other hand, you want your APPLE to search through the data base for a certain key word, type the number for the column which should contain that word. In the above example you might type a 1, to specify the column headed NAME. When you press RETURN, the APPLE will ask you to type the NAME that you want to find: SALLY . You do not have to type a full NAME: any character or series of characters, including numbers, may be searched for. However, the search will find the specified characters only if they are the first characters of that column's entry. When you press RETURN this time, the APPLE will display all the records in which the NAME entry begins with the characters SALLY . Finally, you will be asked to choose between 1) do more searches, and 2) make changes. To return to the main menu, press RETURN.

CHANGING DATA:

One of the selections listed under SEARCH will be MAKE CHANGES. Type the number of this selection if you wish to modify any of the data your file. You will be asked first to specify the record number, then to specify the column number, and finally to make the necessary change. (NOTE: You must know the record number of the item you wish to change. If you don't know this number, use the SEARCH feature to find the appropriate record. Each searched-for record will be displayed along with its record number.)

3 ADD RECORDS

This is the one you've been waiting for... entering data. As you add data, APPLE will ask you to enter something under each of the headers you specified when you created the file. In our example above, APPLE will first ask for the NAME, then the STREET ADDRESS, then the CITY, and so forth. Simply type the information and press RETURN. At the end of each new record, you will be given the choice of returning to the main menu or continuing to add more new data records.

4 DELETE RECORDS

With this command you can delete whole records within your data base. As you delete records, the remaining records are renumberd to maintain the order. For example, if you delete record number 1, record number 2 becomes record number 1, record number 3 becomes record number 2, and so on.

5 REPORT

Now that you have all of your data typed in, you would like to be able to print it out nicely, right? Right! REPORT is just the command your're looking for.

When you first use REPORT, you will be asked if you wish to create a report format file. Answer NO and you will return to the main menu. Answer YES and you will begin the first part of REPORT: the design phase. You will be given a list of the column headers used in the current data base. Specify the total number of headers you wish to have printed. In other words, if you wish only NAMEs and PHONE NUMBERs to be printed, type 2 .

Now, for each column to be printed you must specify:
   a) The number of that column's header.
   b) The tab position where the printing of that column should start. (This must be a number from 1, the left edge, up to the character-width of your printer. This number must not exceed 132.)
   c) Do you wish the column's numbers added, with a grand total printed on the bottom line? (If this column does not contain numeric data, you should answer N for No.)

Finally, you will be asked to specify the tab position for printing a column of horizontal totals. After each record, this column will show the sum of all the numbers from that record which you specified for vertical summation. If you simply press RETURN, this last column will not be printed.

Reports are difficult to do properly the first time. Trying it once will give you a better idea of how to set the format for best appearance.

Next you must specify which parts of your data base you wish printed. You will be asked, "SELECT RECORDS BY WHICH HEADER?". To select all entries in the data base, simply press RETURN. To print out only a portion of the data base, respond as if you were setting up a SEARCH. For instance, you might type the number for the header CITY. When you press RETURN, the APPLE will first ask you to specify a SECOND HEADER. Press RETURN to ignore this option. Then the APPLE will ask you to type the character or characters it is to look for under the column headed CITY. You might type the word MIAMI BEACH, as shown below:

    SELECT RECORDS WHERE CITY= MIAMI BEACH

The APPLE will then print out all the records in which the CITY entry begins with the characters MIAMI BEACH.

If you do not immediately press RETURN in response the request for a SECOND HEADER, you can specify a second header and select records by TWO keys. For instance, you could print only those records that contain MIAMI BEACH listed under CITY and RELATIVE listed under COMMENTS.

6 SORT

You can sort your data file based on any KEY. Enter the header that you want the sort based on. Entering NAME, for instance, will order your data base so that all of the records are in alphabetical order by name.

If there are items that you would prefer sorted numerically, instead of alphabetically, a numeric sort option is provided. The numeric sort uses the VAL(X) function to determine the actual value of the data entry string. This means that 55C21 will be treated as the number 55, because VAL only evaluates the first characters, stopping at the first non-numeric character. Any string that does not have a digit in the first location will evaluate to zero.

7 TURN ON PRINTER

After this command, APPLE will prompt you for the character-width of your printer (4∅, 8∅, or 132 column) and will adjust the output accordingly. With the printer option on, any command that normally causes output to be displayed on the screen (such as PRINT or LIST) will also send that output to the printer.

8 TURN OFF PRINTER

Turns off the printer option, so that APPLE's output is sent only to the TV screen.

9 LIST

LIST will display and print the entire contents of the current data base file, one item per line. If the printer option is not on, only one screen-full will be displayed at a time. Pressing RETURN will cause the listing to continue.

10 QUIT

Quitting will return you softly back to APPLESOFT BASIC. We hope you have enjoyed your trip....

Program Name: INTEGER HI-RES
Volume Number: 3
Software Bank Number: ØØ224
Submitted By: J. Apple Sede
Program Language: Machine Language
Minimum Memory Size: 16K

INTEGER HI-RES is a set of machine-language subroutines which allow you to
do high-resolution graphics from machine-language programs or from Integer
BASIC. These routines give Integer BASIC the same power found in APPLESOFT.
 With INTEGER HI-RES, you can plot 28Ø points across the screen, by 16Ø or
192 points down the screen, in four colors. What's more, you can color
backgrounds, draw lines between any two points on the screen, and draw any
predefined shape in many different sizes and rotations. If you have not
used the APPLE's high-resolution graphics before, you will be amazed and
delighted at the detailed, speedy images you can produce.

INSTRUCTIONS

In Integer BASIC, to load INTEGER HI-RES from diskette, type

BLOAD INTEGER HI-RES, A$CØØ

When the BASIC prompt (>) returns, type

HIMEM:8192

To save INTEGER HI-RES onto another diskette, type

BSAVE INTEGER HI-RES, A$CØØ, L$4ØØ

To save INTEGER HI-RES onto tape cassette, press APPLE's RESET key to enter
the Monitor (prompt character: *), and type

CØØ.FFF W

Start your tape recorder running in "record" mode; then press APPLE's RETURN
key.

To load INTEGER HI-RES from cassette tape, press APPLE's RESET key to enter
the Monitor (prompt character: *), and type

CØØ.FFF R

Start your tape recorder running in "playback" mode; then press APPLE's
RETURN key. When APPLE beeps and the prompt character returns, type ctrl B
(type B while holding down the CTRL key) and press the RETURN key to enter
Integer BASIC. Then, if your computer has 2ØK Bytes of memory or less, type

HIMEM: 8192

Once the INTEGER HI-RES program is loaded, the user has access to seven
high-resolution graphics subroutines, from Integer BASIC (in
immediate-execution mode or in programs), or from machine-language and
assembly-language programs. These subroutines allow the user to set
high-resolution graphics mode, clear the graphics screen, and draw or
position dots, lines, or predefined shapes on the screen.

In describing how to use the high-resolution graphics subroutines from
BASIC, it is assumed that the user is only moderately familiar with Integer
BASIC. But the descriptions of how to use these subroutines from
assembly-language programs are geared to those users who are quite
experienced in assembly-language programming.

It is suggested that BASIC users try all of the examples which illustrate
the following subroutine descriptions. This is the only way to gain
familiarity with the use of these subroutines from Integer BASIC. All
memory locations are in decimal notation except when preceded by a dollar
sign ($), which will denote hexadecimal (base 16) notation. Integer BASIC
uses numbers in decimal notation only. From the Monitor, numbers expressed
in hexadecimal notation are more convenient.

INIT Initializes high-resolution Graphics mode.

INIT must be CALLed before using any other high-resolution graphics CALL

   From BASIC: CALL 3Ø72
   From assembly language: JSR $CØØ

This subroutine sets high-resolution Graphics mode with a 28Ø X 16Ø matrix
of dots in the upper portion of the screen and four lines of text in the
lower portion of the screen. INIT also clears the graphics portion of the
screen to black.

To see the effect of CALL 3Ø72, first type

VTAB 23

and press the RETURN key. This ensures that what you type will appear at
the bottom of the screen, in what will soon be the four-line text area. Now
type

CALL 3Ø72

and press the RETURN key. Notice that the upper portion of the screen is
cleared of all text, and that your typing shows up in the bottom four lines
of the screen. To get out of high-resolution Graphics mode, type

TEXT

and press the RETURN key. The screen is immediately returned to its
original condition.

To use full-screen high-resolution graphics, with not text, after CALLing
3Ø72, type:

POKE -163Ø2,Ø

From now on, it will be assumed that you know to press the RETURN key after
each command you type.

PLOT Plots a point on the screen.

     From BASIC: CALL 3780
     From assembly language: JSR $C7C

This subroutine plots a single point on the screen. The X and Y coordinates
of this point, and also the color of the point, are selectable by the user.
To do this from Integer BASIC, you will use the POKE command, as follows.

To set the color of the point to be plotted, simply type

POKE 812, 255

The number 812 is the address of a location in memory. All high-resolution
graphics subroutines except the "LINE" subroutine, look at the contents of
memory location 812 to determine the color for plotting. (The "LINE"
routine looks at 28.) The second number, 255, represents the color white.
POKE 812,255 has stored the value 255 in memory location 812.

You can verify that the value 255 has been stored in memory location 812, by
typing

PRINT PEEK(812)

The number 255 will be displayed below the line you just typed, because that
is the value of the contents of memory location 812.

To change the color of the plot to black, type

POKE 812, 0

To change the color to violet, type

POKE 812, 85

And to change the color to green, type

POKE 812, 170

You can select the Y (vertical) coordinate of the point to be plotted by
another use of the POKE command. This time you will POKE an integer between
0 and 159 into location 802. The integer 0 represents the top of the
graphics screen, and 159 represents the bottom of the graphics screen. (If
you are using full-screen graphics, the number 191 will represent the bottom
of the graphics screen.) To set the Y coordinate halfway between the top
and the bottom of the graphics screen, type

POKE 802, 79

Another POKE command selects the X (horizontal) coordinate, but the value of
the X coordinate will be shared between two memory locations. The value you
choose for the X coordinate must be an integer from 0 to 279. The integer 0
represents the left side of the graphics screen; 279 represents the right
side of the screen. To set the X coordinate halfway between the right and
left sides of the screen, type

POKE 800, 139 MOD 256
POKE 801, 139/256

To set the X coordinate to 279, type

POKE 800, 279 MOD 256
POKE 801, 279/256

In general, to set the X coordinate, type

POKE 800, x MOD 256
POKE 801, x/256

where x is an integer from 0 to 279.

Now, in order to plot a white point in the middle of the graphics screen,
type the following commands:

CALL 3072
POKE 812, 255
POKE 802, 79
POKE 800, 139 MOD 256
POKE 802, 139/256

The first line sets high-resolution graphics mode. The second line sets the
plotting color to white. The third line sets the Y coordinate to 79. And
the last two lines set the X coordinate to 139. Now, type

CALL 3780

and a white dot will appear in the middle of the screen.

To draw another white dot in the lower right corner of the screen, type

POKE 802, 159
POKE 800, 279 MOD 256
POKE 801, 279/256
CALL 3780

and a white dot will appear in the lower right corner of the graphics
screen. Now, type

TEXT

to get out of graphics mode.

To use the PLOT subroutine in an assembly-language program, deposit the
color value in location $32C, the Y coordinate in the A register, and share
the X coordinate between the X and Y registers. The hexadecimal color
values are:

Color	Hex Value	Decimal Value
BLACK	$ 0	0
VIOLET	$55	85
GREEN	$AA	170
WHITE	$FF	255

CLEAR   Clears the graphics screen.

   From BASIC: CALL 3086
   From assembly language: JSR $C0E

This subroutine clears the high-resolution graphics screen to black, without resetting the high-resolution graphics mode.

To clear the graphics screen of the two dots plotted in the previous example, type

CALL 3086

The two dots will disappear.

POSN   Positions a point on the screen.

   From BASIC: CALL 3761
   From assembly language: JSR $C26

This subroutine does all the calculations for a PLOT, but does not draw a dot on the screen (it leaves the screen unchanged). This is useful when establishing a starting-point for LINE or SHAPE (described later). To use this subroutine in Integer BASIC, set up the X and Y coordinates just as you did for PLOT.

To use this subroutine in assembly language, set up the X and Y coordinates just as you did for PLOT, using the A, X, and Y registers.

LINE   Draws a line on the screen.

   From BASIC: CALL 3786
   From assembly language: JSR $C95

This subroutine draws a line starting from the last point PLOTted or POSitioNed and ending at the point whose coordinates are currently specified. To draw a violet line from the center of the graphics screen to the lower right corner, type

CALL 3086
POKE 812, 85
POKE 802, 79
POKE 800, 139 MOD 256
POKE 801, 139/256
CALL 3761

The first line clears the graphics screen. The second line sets the
plotting color to violet. The third line sets the Y coordinate to 79. The
fourth and fifth lines set the X coordinate to 139. And the last line
POSitioNs the point in the middle of the screen, without actually plotting
it.

Now, to draw the line to the lower right corner of the screen, type

```
POKE 8Ø2, 159
POKE 8ØØ, 279 MOD 256
POKE 8Ø1, 279/256
CALL 3786
```

The first line sets the Y coordinate to 159. The second and third lines set
the X coordinate to 279. And the last line draws a violet line from the
center of the screen to the lower right corner.

Either the POSN or the PLOT subroutine must be used prior to using the LINE
subroutine in order to set the starting-point of the line to be drawn,
unless the line is an extension of an existing line, in which case any color
change must be POKEd into 28($IC) instead of 812. The coordinates of the
line's end-point must then be POKEd before the LINE subroutine is CALLed.

To use this subroutine in an assembly-language program, POSitioN or PLOT the
starting-point, then set the coordinates of the end-point using the A, X,
and Y registers, and JSR to the LINE subroutine.

SHAPE Draws a predefined shape on the screen.

    From BASIC: CALL 38Ø5
    From assembly language: JSR $DBC

Before this subroutine can be used it is important that the user know how to
predefine a shape. The following paragraphs describe the steps neccessary
to predefine a shape by building a Shape Table.

Each byte in a Shape Table is divided into three sections, and each section
can specify a "plotting vector": whether or not to plot a point, and also a
direction to move (up, down, left, or right). The SHAPE subroutine steps
through each byte in the Shape Table section by section, from the Table's
first byte through its last byte. When a byte that contains all zeros is
reached, the Shape Table is complete.

This is how the three sections A, B and C are arranged within one of the
bytes that make up a Shape Table:

Section:	C		B			A		
Bit Number:	7	6	5	4	3	2	1	Ø
Specifies:	D	D	P	D	D	P	D	D

41
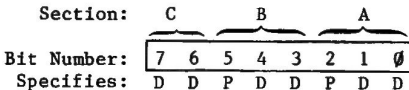
Each bit pair DD specifies a direction to move, and each bit P specifies
whether or not to plot a point before moving, as follows:

If DD = ØØ   move up
     = Ø1   move right   If P = Ø   don't plot
     = 1Ø   move down   = 1   do plot
     = 11   move left

Notice that the last section, C (the two most significant bits), does not
have a P field (by default, P=Ø), so section C can only specify a move
without plotting.

Each byte can represent up to three plotting vectors, one in section A, one
in section B, and a third (a move only) in section C.

The sections are processed from right to left (least significant bit to most
significant bit: section A, then B, then C). At any section in the byte, IF
ALL THE REMAINING SECTIONS OF THE BYTE CONTAIN ONLY ZEROS, THEN THOSE
SECTIONS ARE IGNORED. Thus, the byte cannot end with a move in section C of
ØØ (a move up, without plotting) because that section, containing only
zeros, will be ignored. Similarly, if section C is ØØ (ignored), then
section B cannot be a move of ØØØ as that will also be ignored. And a move
of ØØØ in section A will end your Shape Table unless there is a 1-bit
somewhere in section B or C.

Suppose you want to draw a shape like this:
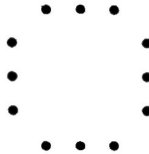
First, draw it on graph paper, one
dot per square. Then decide where
to start drawing the shape. Let's
start this one at the center. Next,
draw a path through each point in
the shape, using only 9Ø degree
angles on the turns:
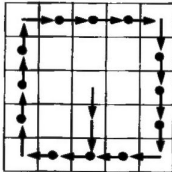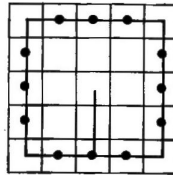
Next, re-draw the shape as a series
of plotting vectors, each one moving
one place up, down, right, or left,
and distinguish the vectors that
plot a point before moving (a dot
marks vectors that plot points).

42

Now "unwrap" those vectors and write them in a straight line:

↓ ↓ ↤↦ ↤↦ ↑ ↓ ↓ ↓ → ↤↦ ↤↦ ↤↦ ↓ ↑ ↑ ↑ ← ↤↦

Next, draw a table like the one in Figure 1, below:

Section	C	B	A		C	B	A			Vector	Code	
Byte Ø		↓	↓			Ø1Ø	Ø1Ø			↑	ØØØ	
1		↤↦	↑			111	111			→	ØØ1 or Ø1	Move
2		↓	↓			1ØØ	ØØØ			↓	Ø1Ø or 1Ø	Only
3	→	↓	↓		Ø1	1ØØ	1ØØ			←	Ø11 or 11	
4		↤↦	↤↦			1Ø1	1Ø1					
5		↓	↤↦			Ø1Ø	1Ø1			↓	1ØØ	
6		↓	↑			11Ø	11Ø			↤↦	1Ø1	Plot
7		←	↓			Ø11	11Ø			↕	11Ø	& Move
8			↤↦				111			↤↦	111	
9					ØØ	ØØØ	ØØØ	← Denotes End of Shape Table				

└─This Vector─┘
Cannot Plot
or Move Up
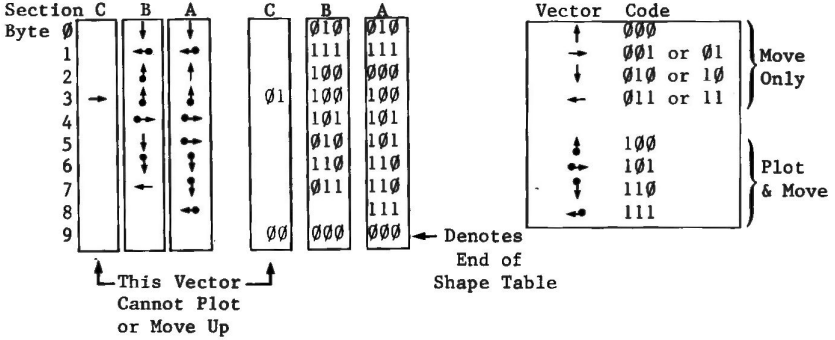
Figure 1

For each vector in the line, determine the bit code and place it in the next available section in the table. If the code will not fit (for example, the vector in section C can't plot a point), or is a ØØ (or ØØØ) at the end of a byte, then skip that section and go on to the next. When you have finished coding all your vectors, check your work to make sure it is accurate.

Now make another table, as shown in Figure 2, below, and re-copy the vector codes from the first table. Recode the vector information into a series of hexadecimal bytes, using the hexadecimal codes from Figure 3.

						Bytes Recoded in Hex			Codes		
Section:	C	B	A							Binary	Hex
Byte Ø	Ø Ø	Ø 1	Ø Ø 1 Ø	=	1 2				ØØØØ	=	Ø
1	Ø Ø	1 1	1 1 1 1	=	3 F				ØØØ1	=	1
2	Ø Ø	1 Ø	Ø Ø Ø Ø	=	2 Ø				ØØ1Ø	=	2
3	Ø 1	1 Ø	Ø 1 Ø Ø	=	6 4				ØØ11	=	3
4	Ø Ø	1 Ø	1 1 Ø 1	=	2 D				Ø1ØØ	=	4
5	Ø Ø	Ø 1	Ø 1 Ø 1	=	1 5				Ø1Ø1	=	5
6	Ø Ø	1 1	Ø 1 1 Ø	≈	3 6				Ø11Ø	=	6
7	Ø Ø	Ø 1	1 1 1 Ø	=	1 E				Ø111	=	7
8	Ø Ø	Ø Ø	Ø 1 1 1	=	Ø 7				1ØØØ	=	8
9	Ø Ø	Ø Ø	Ø Ø Ø Ø	=	Ø Ø ← Denotes End of Shape Table				1ØØ1	=	9
Hex:	Digit 1	Digit 2							1Ø1Ø	=	A
									1Ø11	=	B
									11ØØ	=	C
									11Ø1	=	D
									111Ø	=	E
									1111	=	F

Figure 2

Figure 3

The series of hexadecimal bytes that you arrived at in Figure 2 is the contents of your Shape Table.

To store this Shape Table in memory, type

TEXT

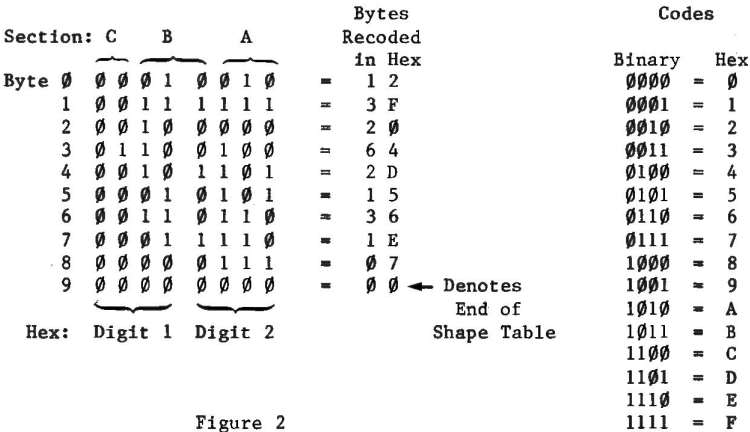to get out of graphics mode. Then press the RESET key on the Apple II keyboard to enter the Monitor (an asterisk will appear). Now, type

1C∅∅:12 3F 2∅ 54 2D 15 36 1E ∅7 ∅∅

and then press the RETURN key. You have just stored the Shape Table in Figure 2 into hexadecimal memory locations $1C∅∅ through $1C∅9 (decimal locations 7168 through 7177). Finally, type ctrl C (type the C key while you are holding down the CTRL key) and then press the RETURN key to get back into Integer BASIC.

Now, to draw this shape in the center of the screen, type

CALL 3∅72

to enter high-resolution graphics mode. Then, type

POKE 8∅2, 79
POKE 8∅∅, 139 MOD 256
POKE 8∅1, 139/256
CALL 3761

This POSitioNs a point at the center of the screen. To set the color for the shape, use the same command that sets the color for PLOT:

POKE 812, 255

The contents of memory location 812 define the color of the next shape drawn from a Shape Table. If location 812 contains the value 255, the shape to be drawn will be white (see PLOT for other color values). Now, type

POKE 8∅4, 7168 MOD 256
POKE 8∅5, 7168/256

These lines will tell the SHAPE subroutine where to find the Shape Table we want to draw (this Table started in memory location 7168). Now, type

CALL 38∅5

A square will appear in the center of the screen.

When storing Shape Table definitions in memory, be sure to find locations
that will not harm or be harmed by programs already in memory. A suggested
location is from $1000 or, in decimal notation, 4096. You may safely use
locations starting at $1C00 (7168 decimal) if you have a 16K system if you
protect the memory area from locations 7168 to 8192 by issuing the
immediate-execution BASIC command

HIMEM: 7168

before storing your Shape Table. Remember that decimal memory locations
8192 through 16384 are used for the high-resolution graphics display. Any
program or Shape Table stored in those locations will be erased when you
call the INIT subroutine.

MEMORY REQUIREMENTS

The INTEGER HI-RES Routines occupy a normally free area of memory between
the top of the variable table and the bottom of the program, from 3072 to
4096 ($C00 to $1000). As your program grows or your variable table size
increases, it is possible to write over part of the INTEGER HI-RES Routines
with program or variables. If, in the course of writing a program, it
suddenly ceases to function, check its size as follows:

PRINT PEEK(202)+PEEK(203)*256 bottom of program -- must be above 4096
PRINT PEEK(204)+PEEK(205)*256 top of variable table -- must be lower
                               than 3072

Program Name: HIGH-VELOCITY, HIGH-RESOLUTION KALEIDOSCOPE
Volume Number: 3
Software Bank Number: 00401
Submitted By: Mike Markkula
Program Language: Integer BASIC (as written, requires Programmer's Aid #1)
Minimum Memory Size: 16K Bytes

APPLE will do many wonderfully flashy, active, rigorous programs for you
when you feel flashy, active, and full of rigor. But during those times
when you are relaxed and comfortable, allow APPLE to create restful beauty
for your simple enjoyment. Kick off your shoes, put on some good music and
let KALEIDOSCOPE, with its ever-changing mosaic of shape and light, produce
a beautiful background to your evening. APPLE will love you for it....

INSTRUCTIONS

As written, the program requires PROGRAMMER'S AID #1, the Integer BASIC ROM
(Read-Only Memory) containing the APPLE high-resolution graphics routines.
If you know programming and do not have this ROM as yet, you may wish to
convert the program to use the high-resolution graphics routines on the
HIRES Demo tape, or convert the entire program to APPLESOFT II.

LOAD the program in Integer BASIC. When you RUN the program you will be
offered the choice of looking at the Kaleidoscope, examining the
documentation, or exiting the program. Type a 1 and press RETURN to see the
Kaleidoscope. When you've watched as long as you wish, press ESC to return
to the menu.

PROGRAMMER'S CORNER

The author has provided a line-by-line documentation section after the
Kaleidoscope portion of the program. To examine it, just pick
"documentation" from the menu. The reason the REMs follow the referenced
lines instead of being within them is simply for speed: when Integer BASIC
(and APPLESOFT II) is looking for a line following a GOSUB or GOTO, it
begins a linear search from the first line in the program. The more lines
it must search through to find the line it is after, the longer it takes.
The same is true of the variable table: placing your most often-used
variables early in your program will decrease their access time. The
construction of this program is an excellent example of how high-speed
programs should be written. Speed is also the reason for writing it in
APPLE Integer BASIC, the fastest BASIC in the Micro world.

Advanced programming note: a HIMEM:8192 has been inserted on line 32767 by
use of a one-line routine, Illegal Statement Writer, documented in THE
INFINITE NUMBER OF MONKEYS. Auto-HIMEM:'s must occur near the end (highest
line number) of a program, in order to avoid the program's writing over
itself and getting lost.

VOLUME 4

ENTITLED: THE APPLE MAGIC LANTERN -- SLIDE SHOW 2

Here at The Orchard, we just took delivery on a new cask of lamp oil. So,
flying into a dither, we powered up the APPLE Magic Lantern for another
picture show....

A grand adventure from childhood's Saturday afternoon Westerns, starring
Hopalong Cassidy and all the characters: the dancehall girl with a diamond
in her ear as big as all outdoors, so big it must be measured with a
macrometer (her diamond, not her ear!); the bad guy lookin' as if Wilkinson
Sword went out of business 3 weeks ago; and some of the strangest looking
horses since The Cavendish Gang Meets Wolfman (sixteen legs and the
biggest brand you ever laid eyes on). A traveling Englishman who looks
suspiciously like Winston Churchill and is constantly peering at his watch
(so he knows when to brush?) and a little girl who needs saving round out
the cast, followed by a preview of the coming attraction: What Ever
Happened to Baby Jane? And like the Westerns from those thrilling days of
yesteryear, there is absolutely no plot!!!

So, gather the children 'round, curl up by the fire, and set on some poppin'
corn as, with chimney chuffing, the show begins...

NOW APPEARING

1 RANDOM LADY   7 TIME MACHINE
2 LADY BE GOOD   8 CHURCHILL
3 MACROMETER   9 HOPALONG CASSIDY
4 DIP CHIPS   1Ø A GIRL'S BEST FRIENDS
5 TEX   11 BABY JANE
6 SQUEEZE

INSTRUCTIONS

Turn the "color" control on your television or monitor all the way down.
(If you have some version of a "magic button" on the set, you may also have
to turn that off.) It may be prudent to note the present setting of the
control so you may return it to that setting after viewing the slides.
These pictures are specifically designed to be viewed in Black and White.

Place the disk in the drive and, from BASIC, type

PR#6

as usual, to boot the disk. A menu will appear. You can choose to watch
the show, copy a slide onto another diskette (in either single or dual disk
systems), put a slide onto a cassette, or you can selectively view any
slide. Just follow the instructions as they appear on the screen. (After
SAVEing, you will be instructed how to view the slide on your system at
home.) To stop viewing a slide and go back to the menu, press the ESC key.

On systems with 48K of memory, where it is possible to use both the Page 1 and the Page 2 high-resolution graphics screens, APPLE will automatically switch between them so that you will not see the slides entering the screen buffer (the technical name for that portion of APPLE's memory used to display high-resolution graphics). If you wish to defeat this feature and have the slides overlay, then, before RUNning, type

HIMEM: 24000

PROGRAMMER'S CORNER

HOW TO PUT TOGETHER A DISK OF YOUR FAVORITE SLIDES

(Both SLIDE SHOW 1 and SLIDE SHOW 2 are driven by the program, THE APPLE MAGIC LANTERN. It is the "HELLO" program on each disk. SLIDE SHOW 2 has a subtly improved version.)

Because of the MAGIC LANTERN program's design, you can easily assemble your slides without tearing apart the program to change it. To make up your own disk, INITialize it naming the "HELLO" program with the name you want for your slide show. (For the benefit of the MAGIC LANTERN program, your slide show's name should not exceed 20 characters.) It is not necessary to have the MAGIC LANTERN in the computer at the time. A simple "HELLO" program such as,

10 END

will suffice, since you will soon replace this program. After INITializing,

LOAD SLIDE SHOW 2

and

SAVE <your "HELLO" program name>

You will now have a disk that will boot up into the MAGIC LANTERN program, under your newly appointed name.

Then, using the slide show program, copy your choice of slides from APPLE slide shows onto the new disk. Or, if you have created your own slides,

BSAVE <slide name>.PIC, A$2000, L$2000

It is important that all slide names end in .PIC   You may SAVE up to 11 slides on one disk. After you have assembled your slides,

LOAD <your "HELLO" program name>

and

LIST 1000, 1190

48

You will ·find, in order of appearance, the titles of the slides on SLIDE
SHOW 2, the disk title SLIDE SHOW 2, and the number of slides on the disk.
Change the slide titles to match those on your new disk. Do not include the
.PIC in these slide titles; this is done automatically by the program. If
you have fewer than eleven slides, delete those title lines left after you
have completed entering your slide titles. Next, change the disk title to
match the name of your slide show. Your disk title may be up to 2∅
characters in length. Also, be sure to insert the correct slide count.
Everything in the program will now conform to your new data -- your new
slide names will be displayed correctly. If you have only six slides, there
will be only six listed on menus. Users will not even be allowed to enter a
7 from the keyboard when a menu of slides is offered.

As the MAGIC LANTERN and 11 slides use up the disk with about 8∅ bytes to
spare, it is not impossible to run out of room (***DISK: DISK FULL ERROR)
when using long picture titles. The creators were interrogated to elicit a
solution. After several hours of "gentle persuasion", they broke from their
sullen recalcitrance and revealed to us a cleverly concealed byte bank,
consisting of lines ∅ through 14. DELeting these lines will free-up a
substantial amount of memory ("and," according to the leakily lamenting
LANTERN author, "break a poor mother's heart).

HOW THE SLIDES ARE MADE, OR DITHERING IN AN ORDERLY FASHION

The original photographs are first "scanned" by a newspaper facsimile
machine. This device consists of a rotating drum upon which the photos are
placed. The drum spins in front of a photocell which "reads" the relative
brightness of each spot ("pixel") on the picture as it slowly traverses the
length of the drum. The relative brightness is stored in computer disk
memory as a number between ∅ and 255, the brighter the pixel, the higher the
number. This process produces 5376∅ bytes of memory (28∅ X 192) and two big
problems: how to display 53K of picture in an 8K screen buffer made up only
of two-level elements; and how to display 256 shades of gray on a screen
which is black and white, on and off. (It will be pointed out by the
skeptics in the crowd that there is obviously gray in the picture, and the
argument is well taken, but from the standpoint of the programmer, each
point on the screen is either on or off. If two horizontally adjacent
points are on, the screen is white; if one point is on and those on either
side are off, the point that is on is gray. That is merely a characteristic
of APPLE's TV display.)

The two problems are solved by post-picture processing. RUN SLIDE SHOW 2
and select "4 LOOK AT A SINGLE SLIDE" and then select slide 1, "RANDOM
LADY". This slide was post-picture processed by the random number method:
each picture byte in turn is compared to a random number between ∅ and 255.
If the byte is greater than the random number, the screen element is turned
on; if not, the screen element is turned off. If an area, or single point,
of the picture has a brightness of 64, or 25% of 256, there is a 25% chance
that that screen pixel will be turned on; if, on the other hand, a given
area, or single point, has a brightness value of 256, there is a 1∅∅% chance
of the pixel being turned on. In this way, a picture with reasonable gray
scale definition and high spacial definition (sharpness) is built up.

RETURN to the menu, and select 2, "LADY BE GOOD". This slide was post-processed by a method known as Ordered Dithering. The writer feels he has dipped too deeply into his bag of humor to maintain much credibility when launching into a seemingly serious discussion of post-picture pixel processing based on such a wonderfully whimsical word as dithering. He deeply regrets his former misdeeds and begs your indulgence -- there really is an Ordered Dither.

Ordered Dithering is a half-tone technique developed by Bell Laboratories for the Picture Phone. It achieves the effect of continuous tone by overlaying a numeric matrix. The dither matrix used for "LADY BE GOOD" is an 8 X 8 matrix; we will examine a 2 X 2 matrix first, as it is easier to grasp the concept involved on a smaller scale.

Consider an extremely low-resolution picture made up of 4 elements laid out in a 2 X 2 pattern. We hold in memory, as we did for the random picture, a number between Ø and 255 for each of the four pixels. But rather than generate a random number to decide whether to turn on the screen at each of our 4 points, we shall turn it on according to whether it exceeds a pre-determined number in our matrix. The numbers are as follows:

	Ø	128
2 X 2 DITHER MATRIX:		
	192	64

If the first element of the picture is greater than Ø (highly likely) the screen will be turned on; if the second is greater than 128, it will be turned on, if the third is greater than 192, it will be turned on, and if the forth is greater than 64, it will be turned on.

This will give us 5 possible grey scale levels (Ø, 1, 2, 3, or 4 elements on in the matrix) in a much more orderly fashion than is achieved using the random method. To do a larger picture, one simply repeats the 2 X 2 matrix, checkerboard fashion, filling out the screen.

A 4 X 4 dither matrix (as was used with "TEX") is arranged as follows:

	Ø	128	32	16Ø
	192	64	224	96
4 X 4 DITHER MATRIX:				
	48	176	16	144
	24Ø	112	2Ø8	8Ø

It should be noted that the 2 X 2 matrix is nestled in the upper left-hand corner of the 4 X 4 matrix; this is a characteristic of dither matrices, the pattern continuing with the 4 X 4 appearing in the upper left hand corner of the 8 X 8, etc. The word, "dither", refers to the smooth distribution of the numbers through the matrix. Each number has been carefully placed to complement its neighbors as closely as possible -- both those within the matrix and within bordering matrices. This arrangement minimizes "noise", allowing the spacial definition to be that of the

original picture, not that of the dither matrix. If you examine "LADY BE GOOD", you will notice the evenness of the matrix as it moves from one gray level to another. Compare it with "RANDOM LADY", which is often clumped or rarified. The price paid in Dithering is spacial definition. Edges become softened and less distinct.

The larger the dither matrix used, and therefore the fewer repeats of the matrix, the higher the gray-scale definition and the lower the spacial definition. On APPLE, a 4 X 4 or 8 X 8 matrix, depending on the specific photograph being processed, seems the best compromise.

Bill Atkinson, one of Apple's resident geniuses, assembled and built the apparatus for making these pictures and processed the pictures themselves. He is a gentleman well known in the world of computer graphics.

Program Name: CHASER
Volume Number: 5
Software Bank Number: 00403
Submitted By: Charlie Kellner
Program Language: Integer BASIC (and machine language)
Minimum Memory Size: 12K Bytes

Ever tried catching a greased pig while wearing snowshoes? Want twice the
fun with half the mess? Welcome to CHASER, that delightful little color
graphics game where you target a great big square with your cross-hatch
sights and fire-when-ready. Easy. Except the square keeps sliding all over
the screen, and when you successfully plug it, it comes back smaller. (The
smallness is no problem -- just miss it a couple of times and it will get
larger. Of course you will lose about half your points.) They say that
once you hit the target when it's size is only one square, you win. This
writer wouldn't know; he cannot stand the tension long enough to find out.
Besides, snowshoes get to be uncomfortable after a spell....

INSTRUCTIONS

LOAD the program into Integer BASIC, then RUN it. You will be presented
with a menu offering you the game, documentation, or the way to get out.
See the PROGRAMMER'S CORNER for a discussion of the documentation.

Enter a 1 and press the RETURN key. You will now be given very complete
information on how to play the game. Please note that pressing the ESC key
will bring you back to the menu. Exiting with a CTRL C can leave poor
APPLE on text page 2 while you are busily typing on page 1. If this happens
to you, just RESET.

Having read the instructions, press RETURN and.... Well, good luck.

PROGRAMMER'S CORNER

This program is typically Kellner: it is fun to play, well-structured and
carefully documented. If you are a beginning programmer, you may learn much
from the techniques used here. If you are a more advanced programmer, the
"documentation" portion of this program should be of special interest. The
animation technique uses text Page 2 (as opposed to high-resolution graphics
Page 2), which occupies decimal memory locations 2048 through 3095, directly
after the memory used for text Page 1. The user is shown a display from
Page 2, while the program does its drawing and undrawing on Page 1, unseen.
When the computer is through drawing, the completed picture on Page 1 is
moved to Page 2 in less than 1/50th of a second, thus allowing a crisp
transition in movement. Within the documentation section, you will find
described a simple 1-wire hardware modification which makes the switch from
Page 1 to Page 2 occur when your TV isn't looking. (The switching is
synchronized to coincide with the TV's vertical retrace time.) At the end
of the documentation is an index to the various subroutines that make up
this program.

Program Name: CALIFORNIA DRIVER'S TEST
Volume Number: 5
Software Bank Number: 00404
Submitted By: Scot Kamins
Program Language: Integer BASIC
Minimum Memory Size: 20K Bytes

In the small community of Milpitas, just east of Apple along the shore of
San Francisco Bay, lives a gentleman by the name of John Scribblemonger.
Several months ago, John decided to leave Milpitas and travel to Alviso,
another small community just east of Apple. Unfortunately, the only route
was by car, and John had no driver's license. So John had to take the
driver's test.

Twenty-seven times John Scribblemonger took that test and twenty-seven times
he failed it. Would he never pass? Was John Scribblemonger doomed to
remain in Milpitas all his life? Not if APPLE could help it!

APPLE could help it. After fifty-three short hours of instruction by APPLE
on the rules of the road, John Scribblemonger went back to take his test.
The Mayor was there. Also the Fire Chief and the Chief of Police. They let
the local junior high out for the afternoon and the school band played as he
marched in for the fateful quiz. And when it was over, John had become a
licensed driver!

Proudly he marched to his waiting car, followed by his friendly dog, Tige.
He climbed inside as the band swelled into a Sousa March. The engine roared
to life as he backed into the Police Chief's car, which rolled back into the
Mayor's limousine, knocking the Fire Chief's car into the bay.

The Police Chief was soon overheard expressing his personal commitment to
John's future security and nutritional well-being as he remarked,
"Scribblemonger, I'm gonna know where you are and what you're eatin' for the
next 90 days!" And the Mayor pledged to personally drive John to Alviso the
very moment he gets out. The Fire Chief was unavailable for comment. He
was being dried out at the Mother Goose Laudromat And Day Care Center over
on First Street. It seems he had been sitting in his car...

INSTRUCTIONS

LOAD the program in Integer BASIC and RUN it. The test consists of 63
multiple-choice questions typically found in the Califorinia Driver's Test.
At the conclusion of the 63 questions, or at any time you elect by answering
with choice number 4, go to scoring: you will be given your score. You will
then be asked if you want to repeat the questions you missed. If you elect
to do so, only those questions will be repeated which you previously
answered incorrectly.

PROGRAMMER'S CORNER

CALIFORNIA DRIVER'S TEST is in this volume even though the laws taught may
not be the same as in other states. The program was included because it is
an excellent example of good educational programming. The student is
rewarded if the right answer is given. But if the answer is wrong, the
student is given encouragement to keep trying, not condemned for being
stupid. And the computer acts as an educational, interacting partner, not
just mindlessly dictating questions without any means to check the answers.

Even a beginning programmer can easily make the program conform to another state's driving code. If you have to take a written test soon, consider re-writing this program to help you study. The program was, in fact, written by Mr. Kamins to prepare himself for the written test. This is a good way to learn--even better than RUNning the program.

The other reason for publishing the program is that it contains one of the classic lines of modern microprocessor programming: line 25Ø. Eloquent.

Program Name: MISSION: U-BOAT
Volume Number: 5
Software Bank Number: 00402
Submitted By: Eric Waller
Program Language: Integer BASIC
Minimum Memory Size: 12K Bytes

It is 1943 and you are Captain of a Battleship cruising Somewhere in the
Atlantic. Suddenly you are set upon by most of the submarines in the known
world. What do you do? Panic! And when you're through panicking, grab
your paddle, train your ear to the SONAR bleeper and start firing
depthcharges; you're on MISSION: U-BOAT!

INSTRUCTIONS

To play, LOAD the program in Integer BASIC and type RUN. You will be shown
the instructions; when you are through reading, press RETURN to start the
game.

This is a one-player game using APPLE's game control #0. You fire off depth
charges by pushing the game control's pushbutton, and avoid being sunk by
maneuvering your ship with the game control. Points are given for every
submarine sunk: the deeper the sub, the greater the number of points given.
But should you be sunk by a torpedo, you loose half your points. If you do
well, you will be given extra time at the end of the game; when it is all
over, you will receive a bonus. Bon Voyage!

PROGRAMMER'S CORNER

Eric Waller has created another well-constructed battle game in MISSION:
U-BOAT. To gain maximum speed of play, it is structured so that the most
often used subroutines occur on the lowest line numbers. While these
subroutines are not well REMmed (to allow maximum execution speed),
examining the label table (LIST 4000,4200) and the main control loop (LIST
5000, 5200), which are extremely well REMmed, will give you the facts you
need to analyze the entire program. To see how the animation looks without
using Page 2, remove lines 60 and 70.

Program Name: APPLE ORGAN
Volume Number: 5
Software Bank Number: ØØ111
Submitted By: Mark A. Cross
Program Language: Integer BASIC (and machine language)
Minimum Memory Size: 16K Bytes

Oh say, can you hear by the dawn's early light
Your APPLE computer, so shining and bright,
Play The Star Spangled Banner, and other stuff too?
If not, APPLE ORGAN's the program for you!

INSTRUCTIONS

Note: if you wish to SAVE this program on a cassette or another diskette,
you must not RUN it until _after_ it has been saved. If you LIST the
program, do not be alarmed to see that it is not your usual BASIC, as there
are sections in machine language.

To begin, LOAD the program in Integer BASIC and type RUN. You are shown a
menu of music-playing subroutines from which to choose.

Choice 1 gives directions on building song tables for the APPLE ORGAN, and
then plays the Star Spangled Banner (single tones) through APPLE's built-in
speaker while displaying, at your command, either the words or a U.S. flag.

Choice 2 composes and plays 4 different types of random computer music
through APPLE's built-in speaker.

Choice 3 displays instructions for building a simple (4 resistors and 1
capacitor) interface that plugs into APPLE's "GAME I/O" socket, allowing you
to play the APPLE ORGAN through your audio amplifier. The APPLE ORGAN plays
music stored in a song table in memory, computing the output waveform in
real time as the sum of 4 different sine waves. The APPLE ORGAN thus plays
4 simultaneous organ tones (4-note chords) -- a much richer sound than is
obtainable one tone at a time from APPLE's built-in speaker.

The following choices require the APPLE ORGAN interface described in Choice
3.

Choice 4 plays the Star Spangled Banner on the APPLE ORGAN in 4-note chords,
through your audio amplifier.

Choice 5 plays the computer-composed "APPLE Boogie" on the ORGAN, through
your audio amplifier.

Choice 6 lets you specify the starting address in memory of your own stored
song table, and the ORGAN plays the song found there through your audio
amplifier. Should there not be a song table, the APPLE ORGAN will play
whatever it finds there, leading to some marvelously original compositions!

PROGRAMMER'S CORNER

Choice 2 is an interesting exploration of various types of computer
generated music, and includes a reference to an excellent article on the
subject in the April 1978 issue of Scientific American.

In choice 3, reference is made to the September 1977 issue of Byte
Magazine, which has an article on D/A (digital to analog) converters.
APPLE is capable of really amazing sound with the D/A interface Mr. Cross
designed for the APPLE ORGAN. Spectacular results could be obtained from
even more sophisticated interfaces.

Program Name: ADD-LIBS
Volume Number: 5
Software Bank Number: ØØ197
Submitted By: Jo Kellner
Program Language: Integer BASIC
Minimum Memory Size: 16K Bytes

What's faster than a speeding lampshade, stronger than a spiked punch, capable of leaping tall tales in a single bound. It's ADD-LIBS Woman! Yes, this mild-mannered reporter for the daily APPLE will suddenly appear, dressed suspiciously like an ASCII keyboard, at your next gathering to cause chuckles, guffaws, and general good cheer. (Batteries not included -- Gleeps sold separately)

You, too, can write a story that would be rejected by Mad Magazine. Your group will complacently answer a few simple requests: "Give me a noun." "Give me a verb." "Give me an adjective."... Then rapturously watch APPLE go to work, spinning a yarn the likes of which you and your friends have never seen.

INSTRUCTIONS

LOAD the program in Integer BASIC, then RUN it. You will be presented with the title and, of course, the name of the Kellner who wrote it (Jo, in this case). The instructions follow, saying simply to answer APPLE's questions and watch the results. Ms. Kellner has described said results as "Hilarious", but the Klan Kellner is often given to such understatement. The fun is directly proportional to the number of participants: 1Ø people will get you elated, 2Ø will get you evicted!

If you press RETURN in response to a request for a word without entering any characters, the program will break out of the current story and ask if you wish another story or to exit the program.

When a story has been displayed, you will be asked if you want to write another story. Answer YES or Y to continue, NO or N to exit the program.

PROGRAMMER'S CORNER

This program, while externally sparkling with entertainment, is not internally tricky or flashy; it is properly structured, depending on subroutines residing below line 1ØØØ to do the repetitious work necessary for each story. But the real lesson to be learned here is the value of hard work: good programming has its price; there is a lot of entertainment to be had with this program because Jo Kellner was willing to spend the hours and days necessary to create it.

Program Name: THE GREAT AMERICAN PROBABILITY MACHINE
Volume Number: 5
Software Bank Number: 00006
Submitted By: Bruce Tognazzini
Program Language: Integer BASIC
Minimum Memory Size: 16K Bytes

Bruce Tognazzini, San Francisco animator, provides you an outstanding
feature film with humor, historical notes, mathematical insight, and the
most incredible cinematographic machine since Charlie Chaplin's "Modern
Times".

INSTRUCTIONS

LOAD the program in Integer BASIC, then RUN it. You will be instructed that
the text portion of the program (which explains the fascinating historical
background of the real Probability Machine) may be skipped by pressing the
ESC key. You will then be told how to speed up or slow down the text; after
which the story will unfold on the screen.

Following this narrative, the Great American Probability Machine itself will
suddenly be revealed. The steam engine, the blue object with the brightly
colored, rotating wheel is mounted towards the top of the 20-story, light
green engine tower on the right. (The original wheel, over 50 ft. in
diameter and made of solid brass, was moved to the Smithsonian during the
recent renovation of the tower.) In the middle of the lavender area on the
blue hopper tower, you will see the gearshift which controls the various red
conveyor belts.

When the mighty machine, with a great clanging of gears, roars into life,
these conveyor belts move behind the light-blue probability matrix inside
their violet cowling, up the hopper tower, to finally lift the blocks from
the brown hopper and drop them. Bouncing, crashing, through the maze of
hand-hewn timbers, they land in the medium-blue, 7-story chutes that stand
below to catch them. When the first of the chutes has filled to capacity,
the gearshift automatically turns off the hopper conveyor and switches to
the main conveyor, which carries the blocks resting upon it off toward the
hopper tower. It then electrically releases a second row of blocks onto the
conveyor, which carries them off, and so on, until the chutes are empty and
the hopper is refilled. The return loop for the main conveyor also snakes
through the lavender cowling. (The original Probability Machine was not so
brightly painted. The author apologizes for certain excesses.)

The machine will continue pitching blocks to go clattering down through the
maze for a period of 3 years, 4 months, and 12 days, or until you press the
ESC key, whichever occurs first. (The significance of this period is
simple: that was how long the original machine operated before accidentally
pitching the head scientist, Van Cliburn Farnsfarfle, who went clattering
down through the maze, immediately after which the project was abandoned.)

BIBLIOGRAPHY

Scientific American -- August, 1886, "The Secret Mathematical Engines
                                of the Civil War" by L. Lambert Post

Steam Power In Old New England -- May, 1936
                                by Mortimer Starzynski, Harper Press

The Smithsonian -- June, 1976, "The New Centennial Exhibit"

The New York Times -- July 28, 1978, "The Farnsfarfle Dynasty -- One
                                Hundred Years Of Government Funding"

PROGRAMMER'S CORNER:

There are really three programs contained within THE PROBABILITY MACHINE:

1) The pseudo-typewriter text program. This program was redone in
   block-structured form in THE INFINITE NUMBER OF MONKEYS on this
   same disk and the interested reader is well-advised to study that
   version rather than this.

2) The Probability Machine itself. The Machine is drawn on text page
   1 while the user is looking at text page 2. Remove line 4096 to
   see it drawn in real-time. You will note that text mode is on
   during the drawing; it must be to maintain text mode on page 2.
   After execution of lines 1000 to 1999 to draw the machine, color
   graphics mode is selected on line 4120. From 4130, where the
   user is switched back to page 1, to 4460 lies the main program
   section which drops the blocks and moves the conveyors.

   The actual descent of the blocks through the matrix occurs in lines
   2000 to 2999. The decision whether the block will fall left or
   right upon encountering each beam takes place on line 2040:

       2040 C= RND (1000)+1>500: IF NOT C THEN C=-1

   At last count, 14,386 more efficient ways to generate random 1s
   or -1s have been discovered, including simply using:

       C= RND (2): IF C = 0 THEN C =-1

   The author was painfully aware, however, that 16 thousand bytes
   of code and two months of his life all dangle from thes one rather
   thin line of code, so he decided to cover up its raw simplicity
   by making it as complex and obscure as cybernetically feasable.
   He has succeeded admirably.

   The test to determine when the matrix is full takes place at the
   end of this subroutine.

The inner subroutine that drops the blocks onto the main conveyor belt as it carries them away has been placed at the beginning of the program, lines 4Ø to 9Ø, to allow the most rapid execution.

Lines 2ØØ to 799 operate the conveyor belts. These lines are a nightmare to wander through. Sadly, there is not a REM statement as far as the eye can see; the author was attempting to drag every ounce of speed from the language. This area could not even be a series of subroutines because the conveyors would then jerk instead of flow. Please do note the constant use of the variable, "phase" throughout. ("Phase" is used to enable the same subroutine stream to move the blocks from an even square to an odd square and from an odd square to an even square.) Phase is just another variable name, holding a value in this program of Ø or 1, but it is an example of an excellent documenting tool open to you in programming: that of giving your variables descriptive names REM statements do an excellent job of explaining one or two lines of a program, but descriptive variable names explain themselves repeatedly -- "phase" is used over 3Ø times in this program.

3) An auto-list program. RUN it by typing RUN 2. It will ask you which lines you wish to list. Enter <starting line number>, <ending line number> and press RETURN. The program will display one "page" of listings and print a "@" prompt near the lower right hand corner of the screen. To continue your listing, press RETURN; to enter a new listing range, press ESC; to exit the listing program during a range request, enter a Ø for <ending line number>.

Again, a far better documented version can be found in THE INFINITE NUMBER OF MONKEYS program on this same volume.

The GREAT AMERICAN PROBABILITY MACHINE was finished less than 3 months after the author had first touched a real computer. While the program indeed accomplishes its task, the lack of internal documentation made the final stages of its creation extremely difficult. Whenever you are not fighting stringent space or speed constraints, REMember the REM.

Program Name: INTEGER BASIC RENUMBER AND APPEND, A Programming Aid
Volume Number: 5
Software Bank Number: ØØ4Ø6
Submitted By: WOZ
Program Language: Machine Language
Minimum Memory Size: 4K Bytes

It doesn't seem to matter how many lines you leave open when beginning a
program; even if you space lines 1ØØ line numbers apart, inevitably one
program section will fill in to the point that there are no numbers left at
all. And then the process of manually re-copying lines with new line
numbers begins. ("Gosh, Dad, how come Mommy's talking bad to APPLE?") An
equally joyous occasion arises when, in writing a new program, you wish to
use that marvelous sorting subroutine you worked out for another program.
All you have to do is SAVE the new program, LOAD and LIST the program
containing the sort subroutine, write out the sort routine on a piece of
paper, reLOAD the new program back into your APPLE, and type the sort
routine into the new program. And they told you that programming was fun.

Well, it is. And here is a program that helps make it so. RENUMBER AND
APPEND. With this little wonder, you can renumber portions of programs,
renumber entire programs, or glue one program to another (with no messy
clean-up). Note: only Integer BASIC programs can be renumbered or
appended, using this program.

INSTRUCTIONS

To load the program into your dealer's computer, type

BLOAD RENUMBER/APPEND, A$CØØ, VØ

To save the program onto diskette, type

BSAVE RENUMBER/APPEND, A$CØØ, L$1ØØ, VØ

To save the program onto cassette tape, enter the Monitor by pressing the
APPLE's RESET key, and type

CØØ.CFF W

Start the recorder in "record" mode, and then press the APPLE's RETURN key.
When the Monitor prompt (*) returns to the screen and the APPLE beeps, the
program has been saved.

TO USE THE PROGRAM:

Load the program from diskette by typing

BLOAD RENUMBER/APPEND, A$CØØ

To load the program from cassette, enter the Monitor by pressing the APPLE's RESET key. Then type

C∅∅.CFF R

Start the recorder in "play" mode, and then press the APPLE's RETURN key. (It may be necessary to remove the plug from the cássette recorder's MONITOR or EARPHONE jack, just long enough to hear the high-pitched steady tone at the beginning of the program. As soon as the tone starts, replace the plug and quickly press RETURN.) When the program has been sucessfully LOADed, stop the recorder and return to BASIC.

This will place the program at decimal location 3∅72 ($C∅∅, in hexadecimal). The program is fully relocatable and may be loaded anywhere you wish; this location was chosen because it is an area of memory not generally used by programs, variable tables, DOS's, or other nefarious creatures that stalk the RAMs.

If you do not have Disk II, you may wish to LOAD the program at decimal location 768 ($3∅∅, in hexadecimal) instead. To do this, type

3∅∅.3FF R

instead of C∅∅.CFF R   The CALL numbers for this location are noted in the following documentation.

The program may be LOADed while the BASIC program you wish to renumber or append is in the computer. Just remember not to type ctrl B when returning to BASIC from the Monitor (use ctrl C, instead). If you have a cassette-based APPLE, keep a copy of RENUMBER/APPEND at the beginning of a cassette tape for easy access to the program. The program is also available in permanent Read-Only Memory (ROM) in Programmer's Aid #1, available at your APPLE dealer.

SECTION 1: RENUMBER

RENUMBERING AN ENTIRE BASIC PROGRAM

After loading your program into the APPLE, type the

CLR

command. This clears the BASIC variable table, so that the Renumber feature's parameters will be the first variables in the table. The Renumber feature looks for its parameters by location in the variable table. For the parameters to appear in the table in their correct locations, they must be specified in the correct order and they must have names of the correct length.

Now, choose the number you wish assigned to the first line in your renumbered program. Suppose you want your renumbered program to start at line number 1∅∅∅. Type

START = 1∅∅∅

Any valid variable name will do, but it must have the correct number of characters. Next choose the amount by which you want succeeding line numbers to increase. For example, to renumber in increments of 1∅, type

STEP = 1∅

Finally, type this command:

CALL 3∅72   (if the program is located at $3∅∅, CALL 768)

As each line of the program is renumbered, its old line number is displayed with an "arrow" pointing to the new line number. A possible example might appear like this on the APPLE's screen:

7->1∅∅∅
213->1∅1∅
527->1∅2∅
698->1∅3∅
13∅∅∅->1∅4∅
13233->1∅5∅

RENUMBERING PORTIONS OF A PROGRAM

You do not have to renumber your entire program. You can renumber just the lines numbered from, say, 3∅∅ to 5∅∅ by assigning values to four variables. Again, you must first type the command

CLR

to clear the BASIC variable table.

The first two variables for partial renumbering are the same as those for renumbering the whole program. They specify that the program portion, after renumbering, will begin with line number 2∅∅, say, and that each line's number thereafter will be 2∅ greater than the previous line's:

START = 2∅∅
STEP = 2∅

The next two variables specify the program portion's range of line numbers before renumbering:

FROM = 3∅∅
TO = 5∅∅

The final command is also different. For renumbering a portion of a program, use the command:

CALL 3∅8∅   (if the program is located at $3∅∅, CALL 776)

64

If the program was previously numbered

```
100
120
300
310
402
500
2000
2022
```

then after the renumbering specified above, the APPLE will show this list of changes:

```
300->200
310->220
402->240
500->260
```

and the new program line numbers will be

```
100
120
200
220
240
260
2000
2022
```

You cannot renumber in such a way that the renumbered lines would replace, be inserted between or be intermixed with un-renumbered lines. Thus, you cannot change the <u>order</u> of the program lines. If you try, the message

*** RANGE ERR

is displayed after the list of proposed line changes, and the line numbers themselves are left unchanged. If you type the commands in the wrong order, nothing happens, usually (because the variable names then have the wrong lengths).

COMMENTS:

1. If you do not CLR before renumbering, unexpected line numbers may result. It may or may not be possible to renumber the program again and save your work.

2. If you omit the START or STEP values, the computer will choose them unpredictably. This may result in loss of the program.

3. If an arithmetic expression or variable is used in a GOTO or GOSUB, that GOTO or GOSUB will generally not be renumbered correctly. For example, GOTO TEST or GOSUB 10+20 will not be renumbered correctly.

65

4. Nonsense values for STEP, such as $\emptyset$ or a negative number, can render your program unusable. A negative START value can renumber your program with line numbers above 32767, for what it's worth. Such line numbers are difficult to deal with. For example, an attempt to LIST one of them will result in a >32767 error. Line numbers greater than 32767 can be corrected by renumbering the entire program to lower line numbers.

5. The display of line number changes can appear correct even though the line numbers themselves have not been changed correctly. After the *** RANGE ERR message, for instance, the line numbers are left with their original numbering. LIST your program and check it before using it.

6. The Renumber feature applies only to Integer BASIC programs.

7. Occasionally, what seems to be a "reasonable" renumbering does not work. Try the renumbering again, with a different START and STEP value.

SECTION 2: APPEND

APPENDING ONE BASIC PROGRAM TO ANOTHER

If you have one program or program portion stored in your APPLE's memory, and another saved on tape, it is possible to combine them into one program. This feature is especially useful when a subroutine has been developed for one program, and you wish to use it in another program without retyping the subroutine.

For the Append feature to function correctly, all the line numbers of the program in memory must be greater than all the line numbers of the program to be appended from tape. In this discussion, we will call the program saved on tape "Program1," and the program in APPLE's memory "Program2."

If Program2 is not in APPLE's memory already, use the usual command

LOAD

to put Program2 (with high line numbers) into the APPLE. Using the Renumber feature, if necessary, make sure that all the line numbers in Program2 are greater than the highest line number in Program1.

Now place the tape for Program1 in the tape recorder. Use the usual loading procedure, except that instead of the LOAD command use this command:

CALL 33$\emptyset$3   (if the program is located at 3\emptyset\emptyset$, CALL 999)

This will give the normal beeps, and when the second beep has sounded, the two programs will both be in memory.

If you get a *** MEM FULL ERR then use the command

CALL 332Ø   (if the program is located at $3ØØ, CALL 1Ø16)

to recover Program2.

COMMENTS:

1. The Append feature operates only with APPLE II Integer BASIC programs.

2. If the line numbers of the two progams are not as described, expect
unpredictable results.

Program Name: THE INFINITE NUMBER OF MONKEYS
Alias: INTEGER BASIC SUBROUTINE PACKAGE (see next page)
Volume Number: 5
Software Bank Number: ØØ178
Submitted By: Bruce Tognazzini
Program Language: Integer BASIC
Minimum Memory Size: 2ØK Bytes

What high-placed, red-faced government officials would rather this story
never got out? What federal agency did its best to suppress it and almost
got away with it? What Washington paper thought the story just too hot to
handle?

Now, for the first time ever, anywhere, APPLENEWS prints the TRUTH of the
most mendacious mess of monkey-business since the Cooper-Simian scandals of
1883.

APPLE's fearless reporter went where others with greater olfactory
sensibilities feared to tread to bring back the greatest story of the 2Øth
century. Find out why that New England paper company chopped down a 1ØØØ
year-old forest in Georgia. Learn the truth behind the recent Brazilian
banana shortage.

INSTRUCTIONS

LOAD the program in Integer BASIC and RUN it; you will be presented with the
menu. For a discussion of choices 2 and 3, see INTEGER BASIC SUBROUTINE
PACKAGE, described on the next page. Type "1" and press RETURN, and the
story will be told. Since the author can't imagine why anyone would wish to
leave this program during its execution, the only way out before the end of
the story (once it has begun) is to type a CTRL C.

Our intrepid journalist made three trips to the historic site during the
course of writing this program; three subjects were viewed, and three
results were recorded. Unfortunately, at some point the manuscripts became
jumbled, so do not be surprised to discover three different outcomes, chosen
almost, it seems, at random.

After the story has been presented, you will be told that pressing RETURN
will return you to the menu. Having returned, you may then exit the program
by selecting choice 4.

If you have only 16K bytes of memory, you may reduce the program size by
typing

DEL 7ØØ, 132Ø

before SAVEing it. This will delete the page-list program which is not used
by THE INFINITE NUMBER OF MONKEYS program itself.

Program Name: INTEGER BASIC SUBROUTINE PACKAGE (see Alias)
Alias: THE INFINITE NUMBER OF MONKEYS
Volume Number: 5
Software Bank Number: ØØ178
Submitted By: Bruce Tognazzini
Program Language: Integer BASIC
Minimum Memory Size: 2ØK Bytes

THE INFINITE NUMBER OF MONKEYS (see previous page) is an approximately 8K program which has been expanded to almost 2ØK by adding a large number of REM statements. These REM statements describe in detail the INTEGER BASIC SUBROUTINE PACKAGE, a group of short but powerful subroutines and functions (many just a single line long) which expand the power of Integer BASIC. The program, along with this documentation, was also designed to be a tutorial in advanced Integer BASIC programming. But neither the program nor the documentation assume anything beyond the user's having read the APPLE II BASIC Programming Manual. It is structured in clearly defined subroutine blocks which interact as little as possible. This allows the user to lift, for example, the VAL(V) function, whole, out of this program and append it to any other program. (Please see RENUMBER/APPEND in this volume.) The author keeps a diskette of such routines that can be immediately EXECed into place (see the APPLE DOS Manual).

The SUBROUTINE PACKAGE in THE INFINITE NUMBER OF MONKEYS program contains the following subroutines and functions:

1) Automatic LOMEM: (and Auto-CLR) Function
2) Integer BASIC CHR$(X) Function
3) Pseudo-typewriter Auto-formatting White Print Routine
4) Text Page 1 to Text Page 2 Memory Move Routine
5) Page LIST Program
6) Illegal Statement Writer
7) Integer BASIC VAL(V) Function

INSTRUCTIONS

LOAD the INFINITE NUMBER OF MONKEYS program in Integer BASIC and RUN it. You will be presented with a menu. For a discussion of choice 1, see the discussion of THE INFINITE NUMBER OF MONKEYS on the previous page.

If you have a 16K byte APPLE, you may reduce the size of the SUBROUTINE PACKAGE by typing

DEL 19ØØ, 28ØØ

This will eliminate the text portion of THE INFINITE NUMBER OF MONKEYS while retaining the entire subroutine package.

A NOTE ON RENUMBERING

Select choice 2, THE TABLE OF SUBROUTINES, from the main menu. Yes, it did happen. Return to the menu (by pressing the RETURN key) and select choice 2 again. When the author began this program, it was decided to make frequent, almost constant use of the RENUMBER/APPEND program, which the author has on-board in PROGRAMMER'S AID #1, but which is also available in a softer form on this disk. To allow fluid renumbering, everything in this program, including the line numbers on this menu, must be updated when the program is

renumbered. If you LIST line 184Ø, you will see the line that produced the menu; as the menu is a LISTing, rather than a print-out, the updating is done by the renumber program itself.

ON USING THE AUTO-LIST PROGRAM

If you are not already there, please return to the main menu (by pressing the RETURN key) and select choice 3, THE AUTO-LIST PROGRAM. You will again be shown the TABLE OF SUBROUTINES and will be given instructions on the use of the auto-lister. This lister will show one text screen of lines at a time and then pause either until you are ready for the next page (press the space bar) or until you wish to cancel the listing and return to the TABLE OF SUBROUTINES (press ESC). You give APPLE your list request by typing the word LIST followed by the number of the line with which you wish to start LISTing. (There is no need to specify an ending line as pressing ESC will end the listing at any time.) One of the delights of the program is that it allows such variants as LSIT and LIAR without giving you a SYNTAX ERR.

(Just to show how far some people will carry something, ESC to the TABLE OF SUBROUTINES and, after noting the line in the lister's instructions which says, TYPE "LIST 3Ø" AND PRESS "RETURN", LIST beginning at line 728. This is how one gets an updated line reference <u>inside</u> a print statement! All this will be much more dramatic if you subject the program to renumbering; we suggest you try it. Please ESC back to the head of the list program.

AN EXPLORATION

The following documentation will provide an overview of the various blocks, starting from the beginning of the program, and give more in-depth information where appropriate. Please type LIST and look at the program's comments as you read those supplied here.

AUTOMATIC LOMEM: (and Auto-CLR) Function

Line 5Ø

Attempting to enter LOMEM: within an Integer BASIC program will produce a
SYNTAX ERR; line 5Ø allows you to accomplish the same task legally. The
desired LOMEM: address in this case is 3Ø72; just replace 3Ø72 (in both
places where it appears) with whatever LOMEM: your program requires. This
must be done before any variables or arrays are either DIMentioned or
assigned values, because the LOMEM subroutine will clear the variable table
when it is run. The second and third statements in the line will act as a
CLR command when used alone.

PURPOSE: to set LOMEM: to a desired number from within a BASIC program.

TO USE: Place at beginning of program and execute before initializing any
variables.

EFFECT: equivalent to typing LOMEM: in immediate mode. Resets bottom of
variable table and CLeaRs variable table.

CALLS: No other lines are referenced by this routine.

INTEGER BASIC CHR$ FUNCTION

LINES 11Ø, 12Ø

11Ø CHS = CHR +128* (CHR<128)

12Ø LC1= PEEK (224): LC2= PEEK (
    225)-(LC1>243): POKE 79+LC1-
    256*(LC2>127)+(LC2-255*(LC2>
    127))*256,CHS:CHR$="A":RETURN

The author was able to make free use of quotation marks throughout THE
INFINITE NUMBER OF MONKEYS program because of the inclusion of this routine.
It is also the alleged generator of monkeytalk.

This subroutine gives you the same ability in Integer BASIC that the CHR$
function delivers in, for example, APPLESOFT BASIC.

In BASIC, the CHR$ function returns a character when given its numerical
position in the list of ASCII characters. (The list may be found in the
APPLESOFT manual.) Many characters cannot be generated on the APPLE II
since its keyboard is upper-case only. Other characters, such as CONTROL C,
cannot be stored in a program since they have special functions in the
system. Yet these characters are often necessary when controlling external
devices, writing programs that write programs, and in many other
applications.

It is important that the second line of the routine be typed <u>exactly</u> as
shown, with the exception that the line number may be changed. If an error
is made, your program may be irreversibly altered. SAVE the program before
RUNning, so should an error be discovered upon RUNning, you may reLOAD the
program and correct the line. The program once entered and checked is
completely safe and has the advantage over other Integer BASIC CHR$ schemes
of being completely independent of the variable table. If the line is used
in a long program on an APPLE with more than 32K bytes of memory, there is a
very remote possibility that one could get a *** >32767 ERR. If this should
happen, insert a REM statement with about 8Ø characters on a higher line
number to force the CHR$ function down in memory.

THEORY OF OPERATION: See ILLEGAL STATEMENT WRITER.

PURPOSE: to convert the ASCII code number placed in CHR into its equivalent
character in CHR$

SETUP: no initializing is necessary

TO USE:
INPUT: Is placed in CHR
GOSUB 11Ø
OUTPUT: Is found in CHR$

VARIABLES EFFECTED: CHS, LC1, LC2, CHR$

CALLS: no other lines are referenced by this routine.

EXAMPLE: contained on line 18Ø

TEXT PAGE 1 TO TEXT PAGE 2 MEMORY MOVE

LINES 33Ø,38Ø

You may move any block of memory to any other block by changing the
addresses listed on line 33Ø, or replacing the variable names with your
desired addresses. The author used these names to make the process as clear
as possible. ("DESTINACION" is purposely mispelled: "DESTINATION" contains
the reserved word, "AT".) Be sure you do not move a block of memory on top
of your program.

PURPOSE: Move a block of memory to another position in memory. Equivalent
to Monitor move routine.

SETUP: No initializing is necessary

TO USE:
INPUT: None
GOSUB 33Ø
OUTPUT: None

VARIABLES EFFECTED: DESTINACION, SOURCESTART, SOURCEFINISH

CALLS: No other lines are referenced by this routine.

PSEUDO-TYPEWRITER AUTO-FORMATTING WHITE PRINT ROUTINE

LINES 3ØØ, 69Ø

This is a very handy package that will take loosely entered PRINT statements and "type" them out to the screen, breaking lines between words, with options for typewriter-like sound and hesitation. With minor modification, as described below, it can be used for higher speed word-at-a-time output. The author developed it because he found he was spending an inordinate amount of time carefully formatting PRINT statements in his programs, only to find himself doing it all over every time he changed one word. With this system, editing is quite reasonable.

THEORY OF OPERATION

The programmer supplies the routine with a sentence or phrase contained in S$. Starting at the head of the main loop (Line 6ØØ), the first word is extracted from S$ and placed in W$. (Lines 61Ø, 63Ø). Line 65Ø adds the number of characters in the word to the current TAB position and, if the word will not fit on the line, GOSUBs the scrolling routine. Lines 66Ø and 67Ø print out the letters in the word, one-by-one, adding sound and delay if requested by main program.

DELeting line 66Ø and replacing it with: "66Ø POKE 5Ø,63: PRINT WRD$;: POKE 5Ø,255" will print one whole word at a time. (Good programming style dictates that the system should be left in the most normal condition possible when a user aborts a program. By turning off white text mode as soon as possible, the user is unlikely to be caught in reverse-mode after typing a ctrl C.

Note on Line 54Ø: when writing subroutine blocks that could be initiallized more than once, use a flag that is turned on when your variables are DIMensioned. This flag can then prevent their being DIMed again. This will prevent any REDIM'ED ARRAY ERRors.

PURPOSE: To print-out properly formatted text on a white background from unformatted PRINT statements.

SETUP: Initialize by a GOSUB 53Ø. This will DIM the variables and run a short routine immitating paper being scrolled into a typewriter.

TO TYPE WORDS:
INPUT: S$ should contain desired word(s), phrase, or sentence
GOSUB 6ØØ
OUTPUT: all output is to screen

TO MANUALLY SCROLL: (for paragraphing, etc.)
INPUT: none
GOSUB 57Ø
OUTPUT: all output is to screen

USER SELECTABLE FLAGS:
Set SOUND equal to 1 for typewriter sound
Set DELAY equal to a number between Ø (no delay) and 5Ø (long delay) for hesitation between characters

VARIABLES EFFECTED: S$, SOUND, DELAY, K, S, SS, WRD$, TIME, CURRENTPOS, SOURCESTART, SOURCEFINISH, DESTINACION

CALLS: no routines outside Lines 300, 690 are called, but AUTO-LOMEM: (line 50) must be executed before any variables are initiallized to allow use of Page 2. It is useful to retain Lines 1550 to 1660 to keep the directions. All other lines may be deleted to isolate the package for use in your own programs.

EXAMPLE: LIST and RUN Line 1640.

This package (with some of the REM statements stripped out) is very small and extremely easy to use, with only one variable, two optional flags, and three calls. It can save you hours of frustrating work; it took the author less than 20 minutes to enter the complete text for the (pre-written) story of The Infinite Number of Monkeys. Subsequent editing was done with a minimum of bother.

To edit any PRINT statements on the APPLE in either Integer BASIC or APPLESOFT II, first type

ESC @   (press and release the ESC key, then type @)

to clear the screen, then type

POKE 33, 33

Now LIST the line(s) you wish to edit. This reduced text window will eliminate the extra spaces in the listed lines and thus, the need for typing ESC A's to skip over them. When you are through editing, type

TEXT

to restore the normal text window.

LINES 7ØØ, 132Ø

Because THE INFINITE NUMBER of MONKEYS was written as a tutorial, the author
has included this listing routine to make the whole program as easy as
possible to study. Since it is written in BASIC, it is fairly slow. But it
does its job, and includes one extremely interesting line, "LIST X,Y" (Line
1Ø3Ø -- try typing it in yourself). This line, and a one-line routine for
entering such lines, will be discussed in detail in ILLEGAL STATEMENT
WRITER, the last section of this discussion.

THEORY OF OPERATION: When the user types a list request, such as, "   LIST
1Ø4Ø", the program carries out the following operations:

1) Line 78Ø. Leading spaces are removed, leaving, "LIST 1Ø4Ø".
2) Lines 82Ø, 83Ø. The first four characters are matched against the
command table. If a valid command is found, the command is removed from the
string, leaving , " 1Ø4Ø".
3)Next, the string is processed by the VAL(V) FUNCTION (see below), which
returns with the integer number 1Ø4Ø contained in V.
4)Finally, a loop X is started from 1Ø4Ø to 32767 and the lines are listed
out. See the REM statements contained in the LIST program for specifics.

Some of the newer programmers may feel the author went to a lot of
unnecessary work just so the user would have to type "LIST" instead of
simply entering the line number. More experienced programmers, of course,
have no doubt he went to a lot of unnecessary work. The author's rather
hastily constructed excuse for this was that he had always wanted to parse
somebody's syntax. He is being carefully watched.

INTEGER VAL(V) FUNCTION

Lines 1400, 1540

One of the marks of a professional programming job is that the program
doesn't "blow-up" when the user makes a predictable error. For example,
when The Infinite Number of Monkeys is first RUN, it is not inconceivable
that a naive user could type "END" instead of "4". Of course, the author
has been very careful to explain what kind of input the computer wishes, but
it is still good programming practice to avoid the avoidable. RUN The
Infinite Number of Monkeys, and, at the menu, type END instead of 4.

Two events took place: First, the computer simply did not accept your
input, giving clear indication that you did not enter it correctly. Second,
in a brief flash, it was announced, "EXTRA IGNORED". That was the VAL(V)
FUNCTION talking. When you input any number in this program, you are doing
so into a string. That string is then processed by the VAL(V) FUNCTION and,
if a valid number between -32767 and 32767 is found, that number is returned
in an integer variable for use by the program. This gives the programmer
complete control over what is coming in; the user cannot, short of
intentional sabotage, crash the program. It is a simple routine to use and
is easy installed within your program. The "EXTRA IGNORED" message alerts
users either that they have entered a number larger than 32767 and the extra
digits have been rejected, or that they have entered non-numeric information
following the number. This feature may be eliminated from the function if
not needed. In fact, it is not needed in this program, but was included to
allow you the option within your own programs.

Still at the main menu, try entering " 3ABC DE". The VAL function ignores
the leading spaces, and evaluates the initial 3 as your response. The extra
is ignored and execution of the command to go to the list routine takes
place. You may now LIST the VAL(V) function, if you wish, and see how it is
constructed. Line 1410 will only DIM V$ if it has not been previously
DIMentioned, line 1420 will cause a return if V$ is null (empty).
Therefore, to initialize this routine the programmer executes a "GOSUB
1400". The next time a "GOSUB 1400" is ordered, the subroutine will expect
V$ to contain the number to be processed.

PURPOSE: This routine converts the string V$ into an integer value V

SETUP: Initialize by a "GOSUB 1400" before using V$

TO USE:
INPUT: desired input is placed in V$
GOSUB 1400
OUTPUT: integer output is found in V upon return

VARIABLES AFFECTED: V$, V, VV, VVV, MINUSFLAG

CALLS: no other routines are referenced by the VAL(V) function

EXAMPLE: LIST and RUN line 1390

The Infinite Number of Monkeys itself, from line 1900 to line 2800, shall be left to the reader to explore. It is not particularly distinguished -- it was not meant to be. It is a sea of unformatted text and GOSUB statements. There are a fair number of delay loops that the author used in timing, or punctuating, the animated text; there is the section from line 2380 to 2670 that contains the "monkey business".

ILLEGAL STATEMENT WRITER

Line 1Ø2Ø (Called by 1Ø15, acts upon 1Ø3Ø)

```
1Ø2Ø LC1= PEEK (224):LC2= PEEK(
     225)-(LC1>243): POKE 81+POS+
     LC1-256*(LC2>127)+(LC2-255*
     (LC2>127))*256,CMD: RETURN
```

A great feature of APPLE Integer BASIC is its entry-time syntax checking. You need not wait until run-time to find out the depths of your folly; APPLE will beep it to the world just as soon as you press the RETURN key. While this is a generally commendable feature, it does inhibit exploration of some potentially interesting lines by steadfastly refusing to accept what seem to be perfectly reasonable commands. Surely, there should be some room for legitimate differences of opinion, APPLE!

   "BEEP! *** SYNTAX ERR."

Why are we not permitted to find out what happens when line 5Ø says, "5Ø NEW" and we RUN 5Ø? ʿWhy can't we DELete line 123 after it has already been used? Why are we forbidden to LIST X?

   "BEEP! *** SYNTAX ERR."

What we need is the ILLEGAL STATEMENT WRITER. With this handy little 1-line subroutine, you can write anything you want! (Of course, poor APPLE may be a litte confused over such lines as, "1ØØ A$=27/PRINT", but you _can_ write it.)

The ILLEGAL STATEMENT WRITER was created to allow BASIC programmers to POKE normally rejected commands, characters, and numbers into their programs. APPLE is quite capable of LISTing from a variable to a variable ("1Ø3Ø LIST X,Y") if one can just get the line into memory. APPLE is perfectly happy with line numbers higher than 32767 if one can just get them entered. (Most of you will have seen line 65535 as the last line of many programs.) One can even poke quotation marks (ASCII 162) inside a quote! This subroutine gives everyone the power that has been reserved for the machine-language jockeys until now.

THEORY OF OPERATION: The following paragraph is a highly technical explanation of how the subroutine works and need not be understood or even read to make full and complete use of the subroutine. The analysis is presented for advanced programmers who may find the previously undocumented pointers identified herein useful in designing new subroutines in this family. The author has written 5 different types of routines, including the CHR$ function in this program, all based on being able to pinpoint the actual memory location of a BASIC program line during run-time. It is his belief there are many more to be discovered.

When Integer BASIC encounters a statement such as "PRINT X", it reads it, parses (interprets) it, and goes off to execute it. When it departs, it stores its current program position in memory locations 224 and 225. Upon completing execution of the statement, it reads these locations to know

where to resume. This subroutine PEEKs these locations to pinpoint its own actual location in memory, and with that information, is able to POKE your chosen command (CMD) into a position (POS) in the next line. LC1 (LoCation1) is given the low byte of the address of the colon which immediately follows it, as this is where the program return pointer is when 224 is PEEKed. (For an explanation of high and low bytes, see INSTRUCTIONS FOR USE below.) LC2 is likewise given the high order byte of the address of its following colon. If LC1 was greater than 243, LC2 will have "clocked over" by the time the pointer reaches it, and thus a 1 is subtracted if LC1>243. We are left then with the actual location of the first colon stored in LC1 and LC2. 82 bytes beyond this colon is the command LIST in line 1030. If we make POS=1 and POKE 81 + POS + <the location in memory> , <number of desired token>, we can POKE any command token into place we want.
  (The balance of line 1020 that deals with adding and subtracting 255's and 256's turns any location numbers higher than 32767 into 2's complement form so users with more than 32K can use the routine.)

The only place this routine will not work properly is just above 32767; if you have more than 32K of memory and encounter a *** >32767 ERR, enter a REM statement with 100 spaces or so on a higher line number to force the ILLEGAL STATEMENT WRITER down in memory. The REM statement may be removed after you are through POKEing your line.

INSTRUCTIONS FOR USE
First, type:

DEL 0, 1014
DEL 1031, 32767

This will leave just three lines. The first line, 1015, controls the subroutine, telling it what to POKE where. The second line is the POKEr, the third, the POKEe. NOTE: LINE 1020 IS TO BE RETAINED, UNMODIFIED, IN EACH EXAMPLE BELOW. Retype line 1030 to read:

1030 PRINT X,Y

Then, RUN. It should read, "1030 LIST X,Y" again. APPLE Integer BASIC and APPLESOFT II are partially compiled languages. When you type a command such as GOTO into a program, APPLE substitutes the four characters G-O-T-O with a "token": a number between 0 and 127 which represents a command. In the case of GOTO, this token is the number 95. When you LIST the program and APPLE encounters a 95 it looks up this token in a table, finds out it means GOTO, and then prints it out that way. This compilation makes your program smaller and faster.

TOKENS

DEL 1015 and enter the following:

1030 PRINT
10 POS=1: FOR CMD = 0 TO 127: GOSUB 1020: PRINT CMD,: LIST 1030 :NEXT
CMD:END

Then RUN. You have just LISTed the complete token table. Most tokens can be used legally in differed mode (written into a program); below is the table of Integer BASIC tokens. (The HEX numbers are for the benefit of the aforementioned machine-language jockeys and are to be ignored by those of us of a more civilized ilk.)

INTEGER BASIC TOKEN TABLE

NUMBER		TOKEN	COMMENTS
DEC	HEX		
Ø	$Ø	HIMEM:	token irrelevent - used internally as begin-of-line.
1	$1		end-of-line token - each line ends with a 1
2	$2	_	used internally in delete line processing
3	$3	:	colon for statement separation
4	$4	LOAD	
5	$5	SAVE	
6	$6	CON	
7	$7	RUN	RUN n, where n is a line number
8	$8	RUN	RUN from first line of program
9	$9	DEL	
1Ø	$A	,	comma used with DEL (DEL Ø,1Ø)
11	$B	NEW	
12	$C	CLR	
13	$D	AUTO	
14	$E	,	comma used with AUTO (AUTO 1Ø,2Ø)
15	$F	MAN	
16	$1Ø	HIMEM:	the real thing
17	$11	LOMEM:	

The following are numeric operators:

18	$12	+	
19	$13	-	the associated parentheses are 56 and 114
2Ø	$14	*	example: A = 14 * (27 + 15)
21	$15	/	

The following are numeric variable logical operators:

22	$16	=	example: IF X = 13 THEN END
23	$17	#	
24	$18	>=	
25	$19	>	
26	$1A	<=	
27	$1B	<>	
28	$1C	<	
29	$1D	AND	
3Ø	$1E	OR	
31	$1F	MOD	

NUMBER		TOKEN	COMMENTS
DEC	HEX		
32	$2Ø	^	
33	$21	+	unused
34	$22	(	used in string DIMs: DIM A$(n)
35	$23	,	
36	$24	THEN	followed by a line number: IF X = 3 THEN 1Ø
37	$25	THEN	followed by a statement: IF X = 3 THEN A$ = "CAT"
38	$26	,	used with string inputs: INPUT "WHO", W$
39	$27	,	used with numeric inputs: INPUT "QUANTITY",Q
4Ø	$28	"	beginning quote
41	$29	"	ending quote
42	$2A	(	substring left parenthesis: PRINT A$(12,14)
			used with 114 as right parenthesis (see also 66)
43	$2B	!	unused
44	$2C	!	unused
45	$2D	(	variable array left parenthesis: X(12)
			used with 114 as right parenthesis
46	$2E	PEEK	uses 65 and 114 for parentheses
47	$2F	RND	" "
48	$3Ø	SGN	" "
49	$31	ABS	" "
5Ø	$32	PDL	" "
51	$33	RNDX	unused
52	$34	(	used in variable DIMS: DIM A(1Ø)
53	$35	+	unary signum: A = +5
54	$36	−	unary signum: B = −5
55	$37	NOT	numeric
56	$38	(	used with 114 in logic statements and numeric operations:
			IF C AND (A = 14 OR B = 12) THEN X = (27 + 3)/ 13
57	$39	=	string logical operator: IF A$ = "CAT" THEN...
58	$3A	#	string logical operator
59	$3B	LEN(	uses 114 as right parenthesis
6Ø	$3C	ASC(	" "
61	$3D	SCRN(	" "
62	$3E	,	comma used with scrn: PRINT SCRN(X, Y)
63	$3F	(	used with 114 after PEEK, SGN, ABS, and PDL

```
NUMBER      TOKEN      COMMENTS
DEC   HEX
 64   $4Ø      $         string
 65   $41      $         unused
 66   $42      (         special case string array right parenthesis.  used
                         when string array is the destination of the data.
                         in the example, A$(1) = B$(1) , the A$ left
                         parenthesis will be 66 and B$'s will be 42.
                         used with 114 as right parenthesis
 67   $43      ,         DIM <varname>, <string varname>
 68   $44      ,         DIM <varname>, <numeric varname>
 69   $45      ;         string PRINTs:    PRINT <varname>; <string varname>; "X"
 7Ø   $46      ;         numeric PRINTS:   PRINT <varname>; <numeric varname>; 7
 71   $47      ;         end of PRINT statement:    PRINT A;
 72   $48      ,         string PRINTS:    PRINT <varname>, <string varname>, "X"
 73   $49      ,         numeric PRINTS:   PRINT <varname>, <numeric varname>, 7
 74   $4A      ,         end of PRINT statement:    PRINT A$,
 75   $4B      TEXT
 76   $4C      GR
 77   $4D      CALL
 78   $4E      DIM       string var.   Parentheses 34 and 114, comma 67
 79   $4F      DIM       numeric var.  Parentheses 52 and 114, comma 68
 8Ø   $5Ø      TAB
 81   $51      END
 82   $52      INPUT     string with no prompt:       INPUT A$
 83   $53      INPUT     string or numeric with prompt:
                         INPUT "WHO", A$      uses comma 38
                         INPUT "NUMBER", A    uses comma 39
 84   $54      INPUT     numeric with no prompt:      INPUT A

      The following are for FOR/NEXT loops:
 85   $55      FOR
 86   $56      =
 87   $57      TO
 88   $58      STEP
 89   $59      NEXT
 9Ø   $5A      ,

 91   $5B      RETURN
 92   $5C      GOSUB
 93   $5D      REM
 94   $5E      LET
 95   $5F      GOTO
```

NUMBER		TOKEN	COMMENTS
DEC	HEX		
96	$6Ø	IF	
97	$61	PRINT	string variable or literal: PRINT A$: PRINT "HELLO"
98	$62	PRINT	numeric variable: PRINT A
99	$63	PRINT	dummy PRINT: PRINT : PRINT
1ØØ	$64	POKE	
1Ø1	$65	,	comma used with POKE
1Ø2	$66	COLOR=	
1Ø3	$67	PLOT	
1Ø4	$68	,	comma used with PLOT
1Ø5	$69	HLIN	
1Ø6	$6A	,	comma used with HLIN
1Ø7	$6B	AT	AT used with HLIN
1Ø8	$6C	VLIN	
1Ø9	$6D	,	comma used with VLIN
11Ø	$6E	AT	AT used with VLIN
111	$6F	VTAB	
112	$7Ø	=	string -- non-conditional: A$ = "HELLO"
113	$71	=	numeric -- non-conditional: A = 14
114	$72	)	the only right parenthesis token -- won most-popular-token award in Atlantic City
115	$73	)	unused
116	$74	LIST	LIST a range of numbers or specific number: LIST 1Ø : LIST 12Ø, 32767
117	$75	,	comma used with LIST
118	$76	LIST	LIST entire program
119	$77	POP	
12Ø	$78	NODSP	string variable
212	$79	NODSP	numeric variable
122	$7A	NOTRACE	
123	$7B	DSP	string variable
124	$7C	DSP	numeric variable
125	$7D	TRACE	
126	$7E	PR#	
127	$7F	IN#	

To use an illegal token inside a program, there must first be a legal line in which to POKE the new token. Because we could not enter, "LIST X,Y", we entered "PRINT X,Y" and then changed the two tokens. If you wish to have, "1Ø3Ø DEL 1Ø", then first enter "1Ø3Ø PRINT 1Ø" or "1Ø3Ø INPUT 1Ø". Then POKE in the new token.

VARIABLE NAMES AND SPACES

Variable names are made up of an alpha character which may be followed by a series of alpha or numeric characters. <u>Anything other than alphanumeric characters appearing outside of quotes and REM statements are tokens.</u>

Spaces between tokens, numbers, and variable names are deleted upon entry and re-supplied during LISTing. In counting the number of bytes in a line, all spaces outside of quotes and REM statements should be overlooked. When a REM is LISTed, one space is inserted after the word REM; therefore, the statement which lists as REM APPLE uses 6 bytes of memory, not 7.

CHARACTERS

The numbers from 128 to 256 are ASCII characters in Integer BASIC. A chart of these characters can be found in many computer manuals, including the APPLESOFT manual. If your chart lists characters with numbers from Ø to 127, just add 128 to compute their "negative-ASCII" counterparts. (You might find it useful to write the higher numbers into your APPLESOFT manual.) Enter the following lines:

1Ø3Ø PRINT "A"
1Ø POS =3: FOR CMD=128 TO 255: GOSUB 1Ø2Ø: PRINT CMD,: LIST 1Ø3Ø: NEXT CMD:
END

Then RUN it. To POKE the ASCII numbers between the quotation marks, it was first necessary to set POS. As PRINT is a command word, and therefore a single token, or byte, and its position is 1, then the space between the quotes is 3. By counting out from the beginning of the line, you may POKE your command, character, or number anywhere within the line.

NUMBERS

Numbers, like tokens, are converted upon entry. Unlike tokens, converted numbers always occupy 3 bytes. (The numbers we are considering are not numeric characters that make up part of a variable name, such as ALPHA3, but rather integers, as in X=32 or ALPHA (3).)

The first byte of a line number contains the number of bytes in the line; the first byte of a number within a line contains as a flag the ASCII value of the first digit in the number. This tells the language that the following two bytes are a number. (The flag can be the ASCII value of <u>any</u> digit -- 176 does nicely -- it is being used as a flag, not a value.) The actual number itself is made up of two bytes, the first being the low-order byte, the second, the high-order byte. When you enter a line number such as 1Ø3Ø into APPLE, the language first computes 1Ø3Ø MOD 256 and puts that number

in the first byte and then computes 1030 / 256 and puts that number in the second byte. With two bytes, each capable of storing numbers from 0 to 255, the square of 256, or 65536 numbers may be expressed. (This 65K range is from -32767 to +32767, or from 0 to 65535 depending on its interpretation.)

Therefore, whenever you wish to POKE a number into a position past integers in the line, POS must be increased by exactly 3 for each integer to be leap-frogged, be that integer 0 or 32767. Try the following new lines, retaining line 1020:

```
1030 X=14: PRINT "SAY @APPLE@" :END
10 POS = 13:CMD = 162:GOSUB 1020:LIST 1030:END
```

And then RUN it. You'll note, we've a quote within a quote. Your job is to put a quote at the end of the line, where the second @ is located. (The @ has been used arbitrarily; it could be any character.) Keep in mind that position 1 is where X is, that the reserved word (command) PRINT and the reserved word : (statement separator) are each one byte, and that the number 14 is three bytes. Any spaces outside of the quotes do not count. The only thing you must change is POS, the ASCII for a quotation mark, 162 stays the same. After you have both quotes, RUN 1030.

Finally, let's change line 1030 to 65535. The low and high bytes of 65535 are both 255. The line number bytes are the two immediately preceeding POSition 1 in our line, thus they are -1 (the low-order) and 0 (the high-order). To change our line, type this new line 10:

```
10 POS=-1: CMD=255: GOSUB 1020: POS=0:GOSUB 1020:LIST:END
```

Changing the line back to 1030 will be left to you; the information on computing the byte values may be found in the second paragraph of this section on numbers. Just do exactly what the computer normally does.

SELF-WRITING PROGRAMS

It is possible to write a program which can write itself, using this line, by dropping HIMEM: down, POKING in syntactically correct lines off the top end of the program, and POKING a new HIMEM: into place when done using the following line:

```
<linenumber> HMM= 82 + POS + LC1 + LC2*256 : POKE 76, HMM MOD 256: POKE 77, HMM / 256
```

HIMEM: is always 1 past the end of the program; thus we add 82, not 81. The above line assumes that you will not write above 32767 in memory; be sure to bring HIMEM: sufficiently below this figure before beginning. One use of this method would be to write a program that would convert machine language into POKE statements inside your BASIC program; another would be a DATA program that would write: <linenumber> <your input data> : RETURN. (This one would undoubtably lead to another round of HANGMAN games.) Your imagination can supply you with many other uses.

Try this simple sample primitive program, after setting HIMEM: to
(arbitrarily, but safely) 8192 and DELeting all but line 1Ø2Ø. (You need
not enter the REM statements.)

1Ø DIM Y(13):REM OUR LINE WILL BE 13 BYTES LONG
2Ø Y(1)=Ø:Y(2)=8Ø:Y(3)=97:Y(4)=4Ø:Y(5)=193:Y(6)=2Ø8:Y(7)=2Ø8:Y(8)=2Ø4:REM
TOKEN AND ASCII CODE FOR EACH BYTE IN LINE
3Ø Y(9)=197:Y(1Ø)=41:Y(11)=3:Y(12)=81:Y(13)=1:Y(Ø)=14:REM THE FIRST BYTE IN
THE LINE MUST ALWAYS CONTAIN THE LENGTH OF THE LINE
4Ø FOR POS = -2 TO 11:REM FIRST BYTE IN LINE IS AT POSITION -2
5Ø CMD = Y(POS +2):REM TO READ Y FROM Ø TO 13, WE MUST ADD 2 TO POS
6Ø GOSUB 1Ø2Ø:REM POKE THE BYTE IN PLACE
7Ø NEXT POS: POS = POS-1 :REM LOOP UNTIL DONE
8Ø REM WHEN EXITING THE LOOP, POS WILL HAVE BEEN INCREMENTED TO 12, ONE
GREATER THAN END-OF-LINE, SO SUBTRACT 1 TO MAKE IT ACCURATE
9Ø HMM=82 + POS + LC1 + LC2*256 : POKE 76,HMM MOD 256: POKE 77,HMM/256
1ØØ LIST :GOTO 2Ø48Ø

SUMMARY: It is hoped that the ILLEGAL STATEMENT WRITER and the above
discourse will lead you to a fuller understanding of Integer BASIC and
computer language processes in general. While many illegal statements are
fun to play with, "tricky" programming, such as having lines DELete
themselves during run-time should be avoided in serious programming whenever
possible. There are also many tasks that can be accomplished without using
the command word itself (See AUTO-LOMEM:), allowing editing after entry.
But when you need the WRITER for something special, it'll be there, and
you'll never have to take "*** SYNTAX ERR" as the final word again. (BEEP!)