Atari Logo Programming Examples

1) INTRODUCTION
2) VIDEO TURTLE
3) SETREAD and SETWRITE
4) LIST PROCESSING
   a) FRENCH QUIZ
   b) TAPETIME
   c) FLASHCARDS (with global variables)
   d) FLASHCARDS (with local variables)
5) ATARI LOGO RESOURCE GUIDE

ATARI INC.
CONSUMER PRODUCT SERVICE
PRODUCT SUPPORT GROUP
1312 Crossman Ave.
Sunnyvale, CA 94088

800-672-1404 inside CA
800-538-8543 outside CA

With many thanks to Brian Harvey at Atari Research and Development for his time, patience and encouragement.

DEMOPAC # 11
REV.1 JG/11/83

ATARI LOGO PROGRAMMING EXAMPLES

Introduction
JG/10/83

Programming in Logo is quite different from programming in a line oriented language like BASIC. Logo is a structured language. Because of this structure, it is easy for a person working in Logo to break down a problem into smaller problems and write procedures that solve each step of the smaller problem. The main procedure for a Logo program that draws a house might look like this:

```
TO HOUSE
WALLS
DOOR
WINDOW
ROOF
END
```

Each of the tasks required to solve the problem can be broken down into smaller tasks. A procedure can then be written and tested independently of the main program. This modular approach to problem solving is one of the characteristics that make Logo such an excellent "language for learning."

The key to programming in Logo is understanding Logo grammar. A Logo program can be thought of as a sequence of procedures (all Logo primitives are considered to be built-in procedures). Each procedure takes one or more inputs and produces one output. The procedure takes the input, processes it according to its definition and produces an output. This output can then be used as an input to another procedure. (See the Atari Logo Reference Guide for examples and further discussion.)

Programming in Logo consists of defining new procedures. Each newly defined procedure is treated by Logo exactly like a built in primitive. The ability to define new procedures is sometimes called extensibility. This extensibility is responsible for much of Logo's power and beauty.

Since there are many books written on Logo turtle graphics (see the Atari Logo Resource Guide), the enclosed programs demonstrate various aspects of Logo programming that are difficult to understand, not documented in the Atari Logo manuals, or are unique to Atari Logo. Most of the programs written for other versions of Logo will work in Atari Logo with little, if any, modification. Exceptions could occur with programs using list processing and/or local variables. Atari Logo differs from other Logos in its use of the LIST operation and in the ways it can handle local variables. The following discussions on list processing and local variables should help in converting programs written for other Logos and creating programs in Atari Logo.

LOCAL AND GLOBAL VARIABLES

One of Logo's more important features is that it supports local as well as global variables. Global variables are variables that remain in existence throughout the execution of a program. All variables in BASIC, for example, are global. Local variables, however, are local to the procedure that calls them, and only exist while the procedure is running. If we MAKE "NAME JOHN at toplevel, then JOHN will always be the thing in NAME. NAME is now a global variable. To use NAME as a local variable, NAME must be used as an input to a procedure as in the following procedure:

```
TO GREET :NAME
PR SE [HI THERE] :NAME
END
```

GREET will use a local variable NAME which takes on the value of the object that is input to GREET. Thus, if we type GREET BIGWIG then GREET will output the sentence "HI THERE BIGWIG". If we then type PR :NAME, Logo will print "JOHN". JOHN is still the thing in NAME because NAME was made a global variable with the above statement MAKE "NAME JOHN.

If NAME is a global variable and JOHN was the thing in NAME, why did GREET output "HI THERE BIGWIG" instead of "HI THERE JOHN?" The reason is that GREET takes a variable as an input. Whenever a variable is an input to a procedure, it becomes a local variable and creates its own private "library". While the procedure GREET is run, the input to GREET will become the thing in NAME. When the procedure is completed, NAME regains the status that it had before GREET was executed. In this case, JOHN again becomes the thing in NAME because that is what it was before GREET was called.

Local variables are an essential aspect of structured programming and they are necessary to take full advantage of the modular structure of Logo. In some versions of Logo, a local variable can be created as an input to a procedure (as in the above example) or by using the primitive "LOCAL" as follows:

```
TO GREET
LOCAL "NAME
PR [WHAT IS YOUR NAME?]
MAKE "NAME RL
PR SE [HI THERE] :NAME
END
```

The advantage of using the "LOCAL" primitive is that the user can use MAKE to assign values to local variables without requiring the variable to be an input to a procedure. Because Atari Logo does not contain the "LOCAL" primitive, using local variables interactively can require more steps as the following example illustrates:

```
TO GREETING
PR [WHAT IS YOUR NAME?]
PRINTNAME RL
END

TO PRINTNAME :NAME
PR SE [HI THERE] :NAME
END
```

In Atari Logo a local variable must be an input to a procedure. In the example above, GREETING reads a list from the keyboard and passes it as an input to the procedure PRINTNAME. While PRINTNAME is running :NAME has the value of the list entered from the keyboard. When PRINTNAME is finished it returns to GREETING and :NAME becomes whatever it was before GREET was executed.

To use 'MAKE' with local variables we could do the following:

```
TO GREETING
PR [WHAT IS YOUR NAME?]
PRINTNAME RL
END

TO PRINTNAME :NAME
MAKE "NAME SE :NAME "COMPUTERS
PR SE [HI THERE] :NAME
END
```

If we type ATARI in response to the prompt WHAT IS YOUR NAME? the above procedure PRINTNAME outputs the sentence "HI THERE ATARI COMPUTERS". Without the LOCAL primitive, all local variables must be passed as inputs to a procedure. When MAKE is used with a variable that is an input to a procedure as in the above example, the variable remains local. If MAKE is used at toplevel or with a variable that is not an input to a procedure, then a global variable is created. A major advantage of using local variables in procedure definations is that a Logo library or tool kit may be created. Since each variable is local to the procedure that uses it, there is no problem with conflicting variable names. Thus once a procedure is defined with local variables, it can become part of the programmers library and used in other programs when the need arises. For a more detailed study of the uses of local and global variables, refer to the FLASHCARDS examples in the list processing section of this document. For an in-depth study of logo grammar refer to Harold Ableson's book Apple Logo. Details of it and other useful resources are listed in the inclosed Atari Logo Resource Guide.

NOTE TO ADVANCED PROGRAMMERS

The manner in which Atari Logo interfaces with the Operating System is very different from BASIC or PILOT. BASIC, and to a lesser extent PILOT, allow a high degree of access to the Operating System (OS). Atari Logo overides many of the OS capabilties and renders them inaccessible to the user. Alternate text and graphics modes,

changing screen margins, etc., are some of the OS options that are not available to simple .DEPOSITS (POKES). Logo does support an assembly language .CALL command which may make it possible to overide some of the defaults of the OS.

VIDEO TURTLE
The Joystick and the Turtle
JG/9/83

VIDEO TURTLE is a program that teaches the turtle to "listen" to the joystick. The turtle will respond to the commands from the joystick to set its heading to any increment of 45 degrees and then take five steps forward. Pressing the joystick trigger will change the turtle and pen to the next of three preset colors and all following commands will then be carried out in the current turtle and pen color until the trigger is pressed again. (The preset pen and turtle colors in this version are red, yellow and blue.)

The pen and turtle colors are changed by using a WHEN demon in the SETUP procedure. The WHEN demon enables the Atari Logo collision detection and sets up a demon to constantly check for a collision or event. In this case the event is the pressing of the joystick trigger, one of the 20 types of collisions or events that WHEN demons can detect.

To make this program usable to younger turtle fans and to give more variety to the turtle's activities, a HELP menu is included when the program is started. The HELP menu can be removed any time a full screen graphics display is desired. While the basic draw and turn commands are given with the joystick, the HELP menu suggests other commands that may be given through keyboard entry. The DRAW procedure contains the KEYP predicate that checks for a keypress and executes the desired instruction via the PEN.UP.DOWN procedure.

VIDEO TURTLE
JG/9/83

```
TO START
INSTRUCTIONS
SETUP
HELP
DRAW 5
END

TO INSTRUCTIONS
SETBG 1
TS CT SETCURSOR [10 2]
PR [ VIDEO TURTLE ]
PR [] PR []
PR [PLUG IN A JOYSTICK IN PORT #1]
PR []
PR [TO CHANGE THE TURTLE'S PENCOLOR PRESS THE JOYSTICK BUTTON]
PR []
PR [THE TURTLE STARTS WITH THE YELLOW PEN DOWN]
PR [] PR []
PR [TO RECALL COMMANDS PRESS H FOR HELP]
SETCURSOR [4 20]
PR [* PRESS ANY KEY TO BEGIN *]
KEYPRESS RC
END

TO KEYPRESS :ANYKEY
END

TO SETUP
CT CS FS ST TELL 0
SETPN 0
SETPC 1 75
SETPC 2 35
SETBG 1
SETC 15
WHEN 3 [CHANGEPN CHANGETURT.CLR]
END

TO HELP
CT SS
PR [ U PICKS UP THE PEN]
PR [ D PUTS DOWN THE PEN]
PR [ X PUTS REVERSING PEN DOWN]
PR [ C CLEARS SCREEN]
PR [ E ERASE HELP]
END

TO DRAW :STEP
IF KEYP [PEN.UP.DOWN RC]
CHECKJOY JOY 0
DRAW :STEP
END
```

```
TO CHECKJOY :POS
IF :POS < 0 [STOP]
SETH 45 * :POS
FD :STEP
END

TO CHANGEPN
IF PN = 2 [SETPN 0 STOP]
SETPN ( PN + 1 )
END

TO CHANGETURT.CLR
IF PN = 0 [SETC 15]
IF PN = 1 [SETC 75]
IF PN = 2 [SETC 35]
END

TO PEN.UP.DOWN :UPDOWN
IF :UPDOWN = "E [CT FS]
IF :UPDOWN = "H [HELP]
IF :UPDOWN = "X [PX]
IF :UPDOWN = "C [SETUP]
IF :UPDOWN = "U [PU]
IF :UPDOWN = "D [PD]
END
```

USING SETREAD AND SETWRITE
More Than Just Dribbling
JG/6/83

The Atari Logo Reference Manual states that SETREAD and SETWRITE may be used to create a "dribble" file by recording the screen output to either a printer, disk or cassette. There is another use for SETWRITE that is not mentioned in the Atari manual. SETREAD and SETWRITE function like the "OPEN" command in ATARI BASIC. They open an Input Output Control Block (IOCB) for reading or writing to a device (casette, disk or printer).

The Atari Logo Reference Manual describes the SAVE command for saving the contents of the workspace to a specified device. The drawback to using this method when saving the workspace is that the procedures are saved in the order that they were written in, not in the logical order in which they are used. This can make for hard to read programs. The following program uses the procedure PRINTER to print readable listings to the printer:

```
TO PRINTER
SETWRITE "P:
PR[TITLE]
PR[]
PR[NAME AND DATE]
PR[]
PO [PROCEDURE NAMES]
SETWRITE []
END
```

PRINTER opens a channel to the printer with the SETWRITE "P: command. The next line prints any title for the heading of the listing. PR [] prints a blank line for formatting purposes. Then the name and date are printed. Another blank line and then the command PO [PROCEDURE NAMES] prints out the procedures names in the order listed inside the brackets. This allows control over the sequence of the procedures in the printout. If three procedures "ONE, TWO, and THREE" were created in the order THREE, TWO and ONE and they were printed to the printer using the SAVE "P: they would be printed in the sequence THREE TWO ONE. But if they were printed using the PRINTER procedure, PO [ONE TWO THREE], they would be listed in the sequence ONE TWO THREE. The SETWRITE [] command is used to close the file and return output back to the screen. It is important to use the SETWRITE and SETREAD commands within a procedure to avoid the "dribble" effect mentioned above.

SETWRITE can also be used when saving procedures to disk. If you have ten procedures in your workspace and you want to save three of them to a disk file SETWRITE can be quite useful. Instead of erasing all the procedures you don't want and then using the SAVE command to save the three you want the following procedure can be used:

```
TO SAVE.DISK
SETWRITE "D:DEMO
PO [ONE TWO THREE]
SETWRITE []
END
```

The file DEMO now contains the desired three procedures. Files created with the above procedure can be LOADed just like a file saved using the SAVE command. It is not necessary to use SETREAD to read files saved in the above manner.

LIST PROCESSING
Life Beyond Turtle Graphics
JG/10/83

The programs that follow will help you understand and use some of the beauty and power of the less known aspects of Atari Logo. There are a great number of books and articles on turtle graphics, (see the Atari Logo Resource Guide) but very few if any on Logo's non-graphics capabilities. It is the intention of this document to help fill that gap. Please experiment with changing the programs to learn how they work. After all, experimentation and discovery are what Logo is all about.

FRENCH QUIZ

FRENCH QUIZ is an example of using Logo's list processing to make a simple quiz. In BASIC this could be done with READ DATA loop. (See inclosed example.) The procedure QUIZ puts the data to be used in the quiz into a list. QUIZ then calls the procedure QUES and passes the list to QUES. The data list is stored in the local variable WORDLIST in QUES.

The line "IF EMPTYP :WORDLIST..." tests to see if the list WORD is empty. If :WORDLIST is not empty, the program continues. The next line "PR [WHAT IS FRENCH FOR]..." tells Logo to find the first item of the first list in :WORDLIST and print it as part of the sentence [WHAT IS FRENCH FOR]... The first item in :WORDLIST is [BOX BOITE] and the first of the first item is BOX. Logo now prints the sentence WHAT IS FRENCH FOR BOX?. The next line IF EQUALP (FIRST RL)...checks to see if the answer typed in (FIRST RL) is the same as the (LAST FIRST :WORDLIST).

The first time through the program the answer typed in would be compared to BOITE, which is the last of the first item in :WORDLIST. If the word typed in is not BOITE then EQUALP evaluates as "false". The program then goes to the next line and prints "TRY AGAIN". The recursive call QUES :WORDLIST sends WORDLIST back to the top of the procedure QUES where the process is repeated until BOITE is typed in.

When the word typed in is BOITE then EQUALP evaluates as "true" and executes the instructionlist [PR [YOU GOT IT!] QUES BF :WORDLIST]. First "YOU GOT IT!" is printed on the screen. Next the first item in WORDLIST [BOX BOITE] is removed as WORDLIST becomes BF WORDLIST. WORDLIST now contains the list [[PEN PLUME] [WINDOW] [FENETRE]]. Finally the new WORDLIST is sent back to the top of the procedure with the recursive call QUES BF :WORDLIST. This process is repeated until :WORDLIST is empty. When EMPTYP :WORDLIST evalutes to "true" (empty) then "END OF QUIZ" is printed to the screen and the procedure QUES stops. The program now returns to the calling procedure QUIZ and executes the next line END which ends the program.

The basic routine of creating a list, looping through the list, displaying selected items in the list, and testing the list for a match or an empty list, can be used in a wide

variety of Logo list processing applications.

TAPETIME

TAPETIME allows the user to enter in the names and times of songs on a record album. The program then prints a list of all the songs and their times along with a list of the total time of the album. TAPETIME follows the general pattern of FRENCH QUIZ but is a little more intricate in that the lists are created by receiving input from the keyboard. In FRENCH QUIZ the list was created by the programmer in the QUIZ procedure.

The procedure START is the main procedure for TAPETIME. The rest of the procedures, INSTRUCTIONS, SETUP, etc., are the subprocedures that are called by START. After each subprocedure is called and executed, the program returns to the main procedure (START) and then calls and executes the next subprocedure. Some Logo programmers prefer to name the main procedure something functional like START or BEGIN while others like to title the main procedure the same name as the program (in this case TAPETIME). In this example, I have chosen START to designate the main procedure.

The first subprocedure INSTRUCTIONS clears the screen and prints the instructions for using the program. The procedure SETUP initializes variables SONGLIST, TOTMINUTES and TOTSECONDS. SONGLIST is given the value of an empty list '[]' while TOTOMINUTES and TOTSECONDS are given the value 0. In ENTERSONG, if we attempt to MAKE "SONGLIST LPUT :SONG :SONGLIST we will get an error telling us that :SONGLIST has has no value. By giving the value of an empty list to "SONGLIST the error is avoided. Whenever a list is used it must have a value. If it has not received one earlier in the program, it must be assigned as an empty list before it can be used. Although they are not lists, the same basic rule applies for TOTMINUTES and TOTSECONDS. They must have a value in order to be used in a program.

ENTERSONG is the subprocedure that creates the user-generated list. After the song title, minutes and seconds are entered into their respective variabls by the user, the program puts them into a sentence and the sentence into a variable called SONG with the program line MAKE "SONG (SE :SONG :MINUTES [:] :SECONDS). Then The next statement MAKE "SONGLIST LPUT :SONG :SONGLIST, makes the song and time in SONG the last item in the list SONGLIST. The minutes are added to the total minutes and the seconds are added to the total seconds. The process repeats until RETURN is pressed signaling the end of the song entries.

PTEST then asks if the total list should be sent to the screen or the printer and stores the device name (P or S) in the variable DEVICE. PTEST then passes the device selection to ENDRECORDS. ENDRECORDS then prints the album title to the selected device (screen or printer) and then calls PRINTLIST and passes the list SONGLIST to PRINTLIST. PRINTLIST tests to see if LIST is empty. It then prints the first item, makes the list all but the first item, and then repeats the process until LIST is emtpy. The program then returns to ENDRECORDS where the next procedure COMPUTE.TIME is called.

COMPUTE.TIME receives as its input a value for the total seconds of all the songs (TSECS) and then displays the total time in minutes and seconds.

FLASHCARDS

FLASHCARDS is a program that functions as electronic flashcards. The program is presented twice; once with local and once with global variables. Both versions of FLASHCARDS are similar to TAPETIME in their general structure with the addition that FLASHCARDS also makes use of the material discussed in SETREAD and SETWRITE to create data files.

To create the data for FLASHCARDS the main procedure MAKE.QUIZ is used. The data is entered in ENTER.NAMES follows the general format discussed in TAPETIME. SAVE.QUIZLIST creates the data file. First the screen is cleared and the cursor is set to the middle of the screen. The message "SAVING TO DISK" is then printed to the screen. After a pause of one second (WAIT 60) the screen display is turned off with a .DEPOSIT 559 0. The screen display is turned off to prevent the data in QUIZLIST from being displayed on the screen while the file is being saved to the disk. (When using SETWRITE the data saved is normally displayed on the screen.) SETWRITE "D:QUIZLIST creates a file on the disk called QUIZLIST. The nextline, PR :QUIZLIST stores the list QUIZLIST on the diksette. SETWRITE [] closes the file. The cursor is then positioned in the center of the screen, the screen display is turned on, and the message "SAVE COMPLETED" is displayed.

TAKE.QUIZ is used to retrieve the data from the disk and to present it on the screen in a flashcard format. READ.QUIZLIST uses SETWRITE "D:QUIZLIST to open the data filefile QUIZLIST that was created with the SAVE.QUIZLIST procedure. The list QUIZLIST is then put into the variable QUIZLIST with the line MAKE "QUIZLIST RL. SETREAD [] closes the file. INIT puts the first item from QUIZ.LIST into a variable DISPLAY.LIST. The first item of QUIZLIST contains the data for the first round of the quiz (i.e., a president's name, homestate and what the president is famous for). DISPLAY.LIST then displays the data in the flashcard format. NEXTLIST prints the first item of the list and then performs the now familar process of removing the first item of the list and and storing the rest of the list back into the variable (in this case DISPLAY.LIST). After cycling through all the data for one round of the quiz, MOVELIST then drops the first item of QUIZLIST and stores the rest of the data back into DISPLAY.LIST. Then the next round of data is cycled through the DISPLAY.LIST procedure. When EMPTYP evaluates to "true" in NEXTLIST and MOVELIST, they return to DISPLAY.LIST which returns control to TAKE.QUIZ. TAKE.QUIZ then calls the last procedure END.OF.QUIZ which ends the program.

FLASHCARDS (with local variables) performs the same results as FLASHCARDS (with global variables). The difference is that instead of being stored into global variables with the MAKE command, the data becomes the input to a subprocedure. The subprocedure then performs the required operation and then returns to the calling procedure. Sometimes the extra complexity needed to use local variables in a program may not be worth the gain. The general rule is to use local variables when ever possible. Although FLASHCARDS (Local) may appear to be more difficult to write, the advantage is that with a little modification it could be uses as a general quiz program

whereas FLASHCARDS (Global) can only be used for one specific quiz (in this case a quiz on Presidents.) The local version is also more elegant and takes better advantage Logo's procedural structure. For those familar with programming in BASIC, the global version may be easier to follow, but it is hoped that the local version will serve as an example of how to make the most of programming in Atari Logo.

FRENCH QUIZ
JG/8/83

```
TO QUIZ
CT
QUES [[BOX BOITE] [PEN PLUME] [WINDOW FENETRE]]
END

TO QUES :WORDLIST
PR []
IF EMPTYP :WORDLIST [PR [END OF QUIZ] STOP]
PR ( SE [WHAT IS FRENCH FOR] FIRST FIRST :WORDLIST [?] )
IF EQUALP ( FIRST RL ) ( LAST FIRST :WORDLIST ) [PR [YOU GOT IT!]QUES BF :WORDLIST]
[PR [TRY AGAIN] QUES :WORDLIST]
END
```

FRENCH QUIZ (BASIC)
JG/8/83

```
10 REM ** ATARI BASIC FRENCH QUIZ **
20 REM ** INCLUDED IN THE ATARI LOGO PROGRAMMING EXAMPLES DEMOPAC **
100 DIM ENGLISH$(30),FRENCH$(30),ENG$(30)
110 PRINT "}"
120 READ ENGLISH$,FRENCH$
130 DATA BOX,BOITE,PEN,PLUME,WINDOW,FENETRE
140 PRINT "WHAT IS FRENCH FOR ";ENGLISH$;"?"
150 INPUT ENG$
160 TRAP 190
170 IF ENG$=FRENCH$ THEN PRINT "YOU GOT IT!":PRINT :GOTO 120
180 IF ENG$<>FRENCH$ THEN PRINT "TRY AGAIN":PRINT :GOTO 140
190 PRINT :PRINT "END OF QUIZ"
```

TAPETIME
JG/8/83

```
TO START
INSTRUCTIONS
SETUP
ENTERSONG
PTEST
ENDRECORDS :DEVICE
END

TO INSTRUCTIONS
CT PR [TYPE IN THE SONG TITLES AND TIMES AS]
PR [INDICATED.]
PR [TO END THE PROGRAM, PRESS RETURN WHENASKED FOR THE SONG TITLE.]
PR [] PR []
END

TO SETUP
MAKE "SONGLIST []
MAKE "TOTMINUTES 0
MAKE "TOTSECONDS 0
PR [TYPE IN THE TITLE OF THE ALBUM]
PR []
MAKE "TITLE RL
PR [] PR []
END

TO ENTERSONG
PR [TYPE IN SONG TITLE]
MAKE "SONG RL
IF EMPTYP :SONG [STOP]
PR []
PR [TYPE IN MINUTES]
MAKE "MINUTES FIRST RL
IF NOT NUMBERP :MINUTES [PR [NEED A NUMBER, RE - ENTER PLEASE] PR []
ENTERSONG STOP]
PR []
PR [TYPE IN SECONDS]
MAKE "SECONDS FIRST RL
IF NOT NUMBERP :SECONDS [PR [NEED A NUMBER, RE - ENTER PLEASE] PR []
ENTERSONG STOP]
PR []
MAKE "SONG ( SE :SONG :MINUTES [:] :SECONDS )
MAKE "SONGLIST LPUT :SONG :SONGLIST
PR :SONG
PR [_____]
MAKE "TOTMINUTES :TOTMINUTES + :MINUTES
MAKE "TOTSECONDS :TOTSECONDS + :SECONDS
PR [] ENTERSONG
END
```

```
TO PTEST
PR [TO SCREEN OR PRINTER? TYPE P OR S]
MAKE "DEVICE RC
END

TO ENDRECORDS :DEVICE
IF :DEVICE = "P [SETWRITE "P:]
CT PR ( SE ["] :TITLE ["] )
PR [CONTAINS THE FOLLOWING SELECTIONS:]
PR []
PRINTLIST :SONGLIST
COMPUTE.TIME ( :TOTMINUTES * 60 ) + :TOTSECONDS
SETWRITE []
END

TO PRINTLIST :LIST
IF EMPTYP :LIST [STOP]
PRINT FIRST :LIST
PRINTLIST BF :LIST
END

TO COMPUTE.TIME :TSECS
PR []
PR ( SE [TOTAL TIME IS] INT ( :TSECS / 60 ) [:] REMAINDER :TSECS 60
END
```

FLASHCARDS (With Global Variables)
JG/10/83

```
TO MAKE.QUIZ
MAKE "QUIZLIST []
ENTER.NAMES
SAVE.QUIZLIST
END

TO ENTER.NAMES
CT
PR [PRESIDENT'S NAME?]
MAKE "PRESNAME RL
PR []
IF EMPTYP :PRESNAME [STOP]
PR [HOME STATE?]
MAKE "HOMESTATE RL
PR []
PR [KNOWN FOR?]
MAKE "KNOWNFOR RL
MAKE "NAMELIST LPUT :KNOWNFOR LIST :PRESNAME :HOMESTATE
MAKE "QUIZLIST LPUT :NAMELIST :QUIZLIST
ENTER.NAMES
END

TO SAVE.QUIZLIST
CT SETCURSOR [10 10]
PR [SAVING TO DISK]
WAIT 60
.DEPOSIT 559 0
SETWRITE "D:QUIZLIST
PR :QUIZLIST
SETWRITE []
SETCURSOR [10 10]
.DEPOSIT 559 58
PR [SAVE COMPLETED]
END

TO TAKE.QUIZ
READ.QUIZLIST
INIT
DISPLAY.LIST
END.OF.QUIZ
END

TO READ.QUIZLIST
SETREAD "D:QUIZLIST
MAKE "QUIZLIST RL
SETREAD []
END
```

```
TO INIT
MAKE "DISPLAY.LIST FIRST :QUIZLIST
END

TO DISPLAY.LIST
IF EMPTYP :DISPLAY.LIST [STOP]
CT
TYPE [PRESIDENT'S NAME > ]
NEXTLIST PR []
PR []
PR [_____]
PR [HOME STATE?]
PR [_____]
PR []
CONTINUE
NEXTLIST PR []
PR []
CONTINUE
PR [_____]
PR [KNOWN FOR?]
PR [_____]
PR [] CONTINUE
NEXTLIST
CONTINUE
MOVELIST
DISPLAY.LIST
END

TO NEXTLIST
IF EMPTYP :DISPLAY.LIST [STOP]
PR FIRST :DISPLAY.LIST
MAKE "DISPLAY.LIST BF :DISPLAY.LIST
END

TO CONTINUE
MAKE "ANYKEY RC
END

TO MOVELIST
MAKE "QUIZLIST BF :QUIZLIST
IF EMPTYP :QUIZLIST [STOP]
MAKE "DISPLAY.LIST FIRST :QUIZLIST
END

TO END.OF.QUIZ
CT
SETCURSOR [8 10]
PR [* * * END OF QUIZ * * *]
END
```

```
TO DISPLAY.LIST :QUIZLIST
IF EMPTYP :QUIZLIST [STOP]
QUESTIONS FIRST :QUIZLIST
DISPLAY.LIST BF :QUIZLIST
END

TO QUESTIONS :FACTS
CT
TYPE [PRESIDENT > ]
PR FIRST :FACTS
ASKONE FIRST BF :FACTS [HOME STATE?]
ASKONE LAST :FACTS [KNOWNFOR?]
END

TO ASKONE :ANS :QUES
PR []
PR [_____]
PR :QUES
PR [_____]
PR []
CONTINUE
PR :ANS
CONTINUE
END

TO CONTINUE
KEYPRESS RC
END

TO KEYPRESS :ANYKEY
END

TO END.OF.QUIZ
CT
SETCURSOR [8 10]
PR [* * * END OF QUIZ * * *]
END
```

FLASHCARDS (With Local Variables)
JG/10/83

```
TO MAKE.QUIZ
MAKE "QUIZLIST []
ENTER.NAMES
SAVE.QUIZLIST
END

TO ENTER.NAMES
CT
MAKE "NAMELIST []
READ.ITEM [PRESIDENT'S NAME?]
IF EMPTYP FIRST :NAMELIST [STOP]
READ.ITEM [HOMETOWN?]
READ.ITEM [KNOWNFOR?]
MAKE "QUIZLIST LPUT :NAMELIST :QUIZLIST
ENTER.NAMES
END

TO READ.ITEM :PROMPT
PR []
PR :PROMPT
ADD.ITEM RL
END

TO ADD.ITEM :ITEM
MAKE "NAMELIST LPUT :ITEM :NAMELIST
END

TO SAVE.QUIZLIST
CT SETCURSOR [10 10]
PR [SAVING TO DISK]
WAIT 60
.DEPOSIT 559 0
SETWRITE "D:QUIZLIST
PR :QUIZLIST
SETWRITE []
.DEPOSIT 559 58
SETCURSOR [10 10]
PR [SAVE COMPLETED]
END

TO TAKE.QUIZ
READ.QUIZLIST
DISPLAY.LIST :QUIZLIST
END.OF.QUIZ
END

TO READ.QUIZLIST
SETREAD "D:QUIZLIST
MAKE "QUIZLIST RL
SETREAD []
END
```

Atari Logo Resource Guide

Atari Logo was developed by Logo Computer Systems Inc.(LCSI) of Montreal, Canada. It is a derivative of and highly compatible with an earlier version of Logo developed for the Apple computer called Apple Logo. The release of Atari Logo is historically significant because it allows for the first time a "full" implementation of Logo on an inexpensive home computer.

Atari Logo has been designed to take advantage of much of the hardware capabilities of the Atari system. This has resulted in some enhancements over previous versions of Logo, most notably the availability of four programmable "turtles" with collision detection and a player-missile "shape" editor. Other enhancements include a 128 color spectrum, easy access to sound and controllers, and the ability to call assembly language subroutines.

Although widely used in early education, Logo is a powerful and sophisticated language. It was designed to have "no threshold" and "no ceiling". It is actually a subset of LISP, a language known for its use in the area of artificial intelligence research. The Atari Logo version is a full featured Logo and includes advanced computer science constructs such as list processing, recursion and local variables.

The Logo Resource Sheet has been compiled to provide additional resources for users of Atari Logo. Most of the sources mentioned refer to versions of Logo that are similar to Atari Logo and those resources that refer specifically to Apple Logo will be the most compatible. The primary difference between Apple and Atari Logo is that Apple Logo has additional list processing commands while Atari Logo contains an enhanced set of turtle graphics commands.

Currently, only two books have been announced specifically relating to Atari Logo. They are Dan Watt's book Learning With Atari Logo and David Thornburg's Computer Art And Animation: A User's Guide to Atari Logo (both available in early '84). Other Atari Logo books are forthcoming and they will be included in future updates of this resource Guide.

Three books included here deserve special mention: Papert's Mindstorms, Abelson's Turtle Geometry and Abelson's Apple Logo. Mindstorms is the most comprehensive expression of the Logo philosophy of education and computing while Abelson's Apple Logo is the most useful manual to date for the serious student of the Logo language. (Do not confuse Apple Logo with Abelson's Logo for the Apple II. Apple Logo is the version that is most compatible with Atari Logo. Logo for the Apple II is for the MIT versions of Logo.) Abelson's Turtle Geometry is a profound excursion into the realms of turtle graphics and has been used as a college level text. Many of the resources included here are very useful in preparing a Logo curriculum and for introducing Logo to beginning students both in and out of a classroom setting.

BOOKS

Abelson, Harold. Apple Logo, New York: BYTE Books/McGraw-Hill, 1982. The best single reference work to date. Features a complete description of Apple Logo which is 90 % compatible with Atari Logo. The book emphasizes the more advanced features such as list processing, recursion and local variables.

Abelson, Harold and Andrea diSessa. Turtle Geometry, Cambridge,MA: MIT Press, 1981. A serious college level text on turtle graphics. Proves that turtle geometry is not just kid stuff.

Beardon, Donna. One, Two, Three, My Computer and Me: A Logo Funbook for Kids, Reston, VA: Reston Publishing Co., 1983.

Beardon, Donna, Kathleen Martin and Jim Muller. The Turtle's Sourcebook, Reston, VA: Reston Publishing Co. 1983. Formally distributed by the Young People's Logo Association, this book is filled with turtle graphics worksheets and and activities.

Bitter, Gary and Nancy Watson. Apple Logo Primer, Reston VA: Reston Publishing Co., 1983.

Burnette, J. Dale. Logo: An Introduction, Morristown, NJ: Creative Computing, 1983. A short collection of turtle geometry explorations.

Goldenberg, E. Paul. Special Technology for Special Children, Baltimore: University Park Press, 1979. Describes the use of Logo and computers with special-needs children.

Minnesota Educational Computing Consortium (MECC.) Apple Logo in the Classroom. MECC Distribution Center, 2520 Broadway Dr., St. Paul MN 55113. A Logo curriculum for children in grades 5 thru 9 - includes teachers manual and worksheets.

MIT Logo Group. Bibliography of Logo Memos.Mit Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA 02139. Capsule descriptions of over 60 publications describing more than ten years of Logo research at MIT.

Papert, Seymour. Mindstorms: Children, Computers and Powerful Ideas, New York: Basic Books, 1980. The philosophy of Logo by its chief proponent. A must for anyone who wants to understand the Logo educational philosophy.

Ross, Peter. Introducing Logo: For the Apple II, TI 99/A, and the Tandy Color Computer, Reading, MA: Addison-Wesley, 1983. Covers list processing and structured programming as well as turtle graphics. Includes several chapters of projects and activites to enhance problem solving capabilities.

Thornburg, David. Discovering Apple Logo, Reading, MA: Addison-Wesley, 1983. The study of how Logo can relate to art and patterns in nature. Covers such topics as fractals and the golden mean.

Computing Teacher, The; "The Logo Center" by Kathleen Martin and Tim Riordin.

Softalk "Logo, the Voice of the Turtle" by Jim Muller.

NEWSLETTERS

Follk, Friends of LISP/Logo & Kids, a non-profit organization formed by a group at San Francisco State University. 436 Arballo Dr., San Francisco CA 94132

Logophile, published by the College of Education, MacArthur Hall, Queen's University, Kingston, Ontario K71 3N6

National Logo Exchange, published by Posy Publications, P.O.Box 5341, Charlottesville, VA 22905

Polyspiral, published by the Boston Computer Society, Three Center Plaza, Boston MA 02108

The Logo and Educational Computing Journal, published by Interactive Educational Foundation, 1320 Stony Brook Road, Stony Brook, NY 11790

Turtle News, published by the Young People's Logo Association, 1209 Hillsdale Drive, Richardson, TX 75801.

PAPERS

The MIT Logo Group has published a series of memos and reports. They can be obtained by contacting the MIT Logo Group, 545 Technology Square, Cambridge MA 02139. Some of the titles are as follows:

Abelson and diSessa, "Student Science Training Program in Mathematics, Physics, and Computer Science." Logo Memo #29, MIT 1976.

Feurzig, Papert, Bloom, Grant, and Solomon, "Programming Languages as a Conceptual Framework for Teaching Mathematics." Report #1889, Bolt, Beranek and Newman, Cambridge, MA 1969.

Papert, "Teaching Children to be Mathematicians vs. Teaching About Mathematics." Logo Memo #4, MIT 1971.

Papert, "Uses of Technology to Enhance Education," Logo Memo #8, MIT 1973.

Papert, Abelson, diSessa, Watt, "Assessment and Documentation of a Children's Computer Laboratory." Logo Memo #48, MIT, 1977.

Papert, diSessa, Watt, Weir, "Final Report for the Brookline Logo Project, Parts I, II, and III." Logo Memos #53 and #54, MIT 1979.

Papert and Solomon, "Twenty Things to do with a Computer." Logo Memo #3, MIT 1978.

Papert and Weir, "Information Prosthetics for the Handicapped." Logo Memo #51.

Weir, "The Uses of Logo for the Diagnosis of Children's Abilities in Areas for Spatial Reasoning, and the use of Logo for Remediation." Internal working paper, MIT Logo Group, 1979.

Weir, "Evaluation of Cultivation of Spatial and Linguistic Abilities in Individuals with Cerebral Palsey." Logo Memo #51, MIT, March 1980.

Weir, and Emmanuel, "Using Logo to Catalyse Communication in an Autistic Child." Department of Artificial Intelligence Memo #15, University of Edinburgh, Scotland, 1976.

DATA STRUCTURES AND DATA FILES

Logo is very good at manipulating words, lists and numbers. It contains a variety of "list processing" commands for this purpose. List processing has been used extensively in artificial intelligence research and originated in the language called LISP of which Logo is a subset. Logo's list processing functions are similar to arrays in other languages. PILOT has no built in commands to manipulate data. It can be done, but not by beginners.

Neither Logo nor PILOT have random access capability and are therefore not suitable for many applications using data files.

SUMMARY

We have seen that the applications in which PILOT and Logo excel are a direct result of their design and purpose. Both can be used with excellent results in teaching computer literacy to beginning students. If interactive dialogues and computer aided instructional programs are desired along with ease of manipulating textual data, than Atari PILOT is ideal. If one chooses to work extensively with turtle graphics and develop a strong foundation for structured programming and the study of more advanced computer science concepts, then Logo is language to use. Of the two languages, PILOT is easier to learn and Logo is more powerful and sophisticated. Both, however, are excellent when used for the tasks for which they were designed.