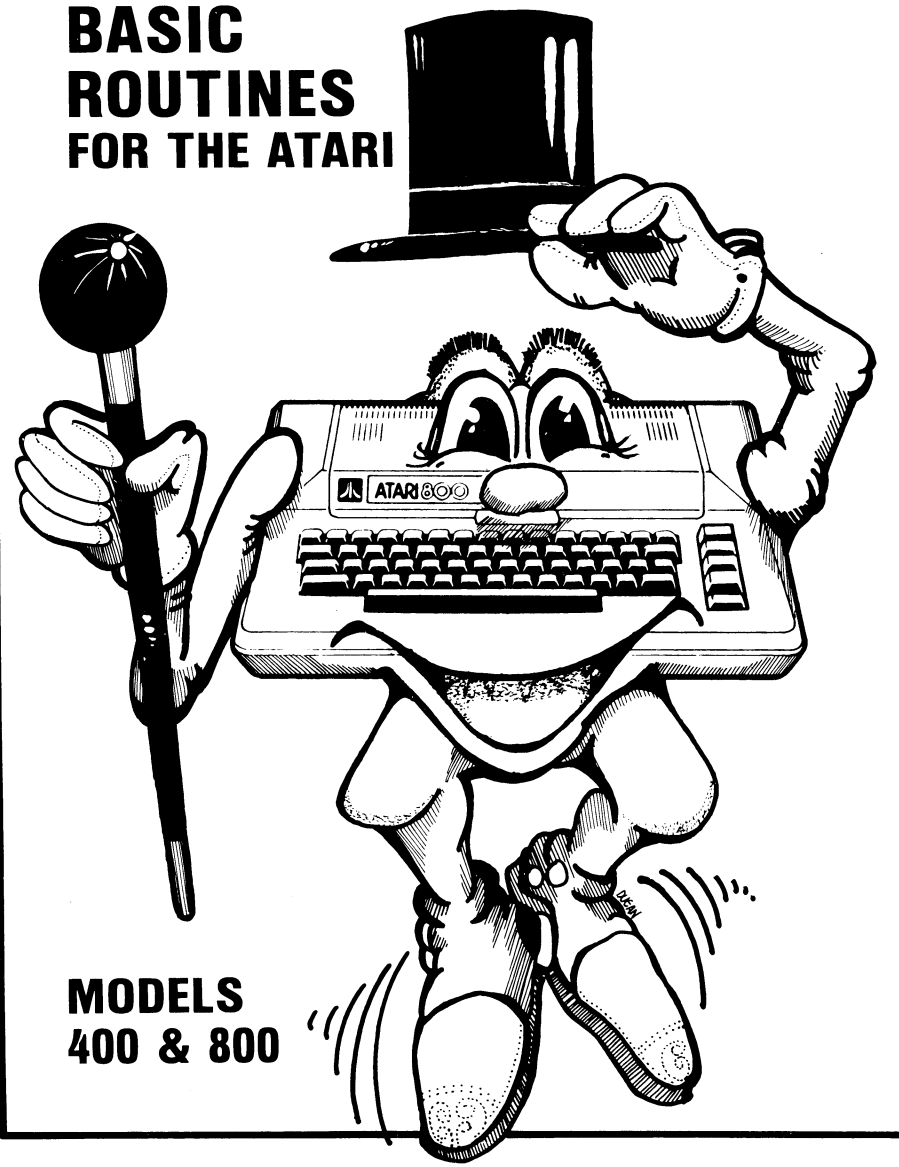
DOCUMENTATION

BASIC ROUTINES FOR THE ATARI



MODELS 400 & 800

ADVENTURE INTERNATIONAL

presents

BASIC ROUTINES

For The ATARI
by Jerry White

BOX 3435 • LONGWOOD, FL 32750 • (305) 862-6917

SOME MATERIAL USED WITH PERMISSION FROM:
A.N.A.L.O.G. 400/800 Magazine
COMPUTE! Magazine

SPECIAL THANKS TO

A.N.A.L.O.G. 400/800 Magazine
P.O. Box 23
Worcester, MA 01603

COMPUTE! Magazine
P.O. Box 5406
Greensboro, NC 27406

and

Those cooperative folks at ATARI INC.

INTRODUCTION

This book is for the Atari Personal Computer owner who wants to write programs using Atari BASIC. All of the literature supplied with the Atari computer is prerequisite reading, since this book will not explain the Atari BASIC commands. It will show you how to put Atari BASIC commands together, creating routines and programs.

When I bought an Atari 800 early in 1980, the BASIC Reference Manual had not yet been printed. I had only three sources of information: some preliminary documentation from Atari, a few tutorial programs available in magazines, and some demonstration programs that were floating around at the computer stores.

Now, the computer comes with a good BASIC Reference Manual and a beginners book on Atari BASIC. There are other manuals available from Atari on hardware, the Operating System (OS), and the Disk Operating System (DOS). Atari offers documentation for both the beginner and the advanced assembly language programmer. But what about the poor soul in the middle? How about the person who isn't ready for the heavy stuff but wants to write some good BASIC programs?

There are two sources of information available to this person. The first source is the many magazines that now carry Atari information and programs. I've written a few of these magazine tutorial programs myself, so you may recognize some of the program listings in this book.

This book is the other source. I learned a great deal from those early demo programs and magazine tutorials, so I wrote this compilation of program listings, with accompanying introductions. Looking at a program listing that contains descriptive variables and REM statements and then watching the program run is a great way to learn programming. Of course this is true only if you understand the BASIC commands.

Atari BASIC Routines supplies you with many short programs and routines, and a few medium-sized programs. They are easy to follow and can be used in writing your own programs. The key to understanding is experimentation. First use the programs and routines as they are — then modify them and see what happens. Dig in and use the Reference Manual to help you with your experimentation with, and understanding of, BASIC routines.

A few utility routines have been added for those who have disk systems. Although some of the programs require a disk system, most

will run on any ATARI 400 or 800 with 16K RAM. Those that require disk will require a minimum of 24K RAM.

I can't teach someone else how to be a BASIC programmer. You have to teach yourself. But if you want to write programs using Atari BASIC, this book plus reference manuals plus the will to learn equals all the tools you need!

TABLE OF CONTENTS

Chapter 1
Common Subroutines (ENGLISH)

ENGLISH is a collection of subroutines designed to demonstrate themselves. The program name, ENGLISH, describes the programming style used in the selection of variable names. Take a quick look at the program listing and you'll see what I mean. It is so easy to read that even a beginner can understand it!

There is only one routine that might be a bit confusing. As you select an option, it changes from normal to inverse video. The routine used to accomplish this begins at line 400 and ends at line 404. Here's how it works.

We set a TRAP to line 200 so that if anything goes wrong, the options will just reappear on the screen. GOSUB KEY sends the program to line 45 which is used to get a key typed by the user and return its value (– 48) as the numeric variable K. We subtract 48 so that K is equal to the actual number of the selected option. Returning to line 400, we set the variable PITCH equal to K*10, and go off to the subroutine at line 35. Here we create a pinging sound and then RETURN once again to line 400. Finally, we position the cursor at the beginning of the selected option on the screen.

At line 402, we begin a FOR NEXT loop using my initials as the X or horizontal coordinate. The cursor will start at the horizontal position 9, and move from left to right. The GET #4 will read what is on the screen since IOCB #4 was previously opened as device S (the screen). The variable 12 in the OPEN command specifies READ and/or WRITE. As each character is read from the screen, it is stored in the variable IT. We add 128 to IT creating the inverse video of the character we read.

At line 404, we reposition the cursor and PUT IT back on the screen. Using the FOR NEXT loop, the line is redrawn and highlighted by inverse video.

```
0 REM ATARI BASIC SUBROUTINES IN ENGLISH BY JERRY WHITE
1 REM ORIGINALLY PUBLISHED BY ZAPATA MICROSYSTEMS
2 REM
3 REM THIS IS A TUTORIAL PROGRAM DESIGNED TO DEMONSTRATE ITS
ELF
4 REM
5 REM BEGINNERS WILL FIND THE SUBROUTINES USEFUL IN DEVELOPI
NG PROGRAMS
6 REM
7 REM LIST THE PROGRAM TO DISK OR CASSETTE AND START A LIBRA
RY OF SUBROUTINES
8 REM
9 REM VARIABLE DEFINITION
10 ZERO=0:NON=1:TEN=10:TIME0=20:SECONDS=21:KOLOR=25:QUIET=30
:ANOTE=35
12 TWO=NON+NON:BLINK=40:KEY=45:BUTTON=50:BUZZ=55:WAIT=60:SND
```

```
=65
13 REM OPEN IOCB #4 AS THE SCREEN THEN SET TIMER TO ZERO
14 OPEN #4,12,0,"S:":GOSUB 20
15 REM DIMENSION STRINGS AND SKIP OVER SUBROUTINES
16 DIM SUB$(TEN),HEADING$(30):GOTO 200
18 REM LINE 20 RESETS TIMER AND LINE 21 GETS THE CURRENT TIM
E
20 FOR LOCATION=18 TO 20:POKE LOCATION,ZERO:NEXT LOCATION:RE
TURN
21 SEC=0:FOR LOCATION=18 TO 20:SEC=SEC*256+PEEK(LOCATION):NE
XT LOCATION:SEC=INT(SEC/60):RETURN
24 REM SET GRAPHICS MODE AND BACKGROUND COLOR
25 GRAPHICS ZERO:SETCOLOR TWO,C,TWO:RETURN
29 REM TURN OFF ALL SOUNDS
30 FOR OFF=ZERO TO 3:SOUND OFF,ZERO,ZERO,ZERO:NEXT OFF:RETUR
N
34 REM PING SOUND ROUTINE
35 FOR VOLUME=TEN TO ZERO STEP -WON:SOUND ZERO,PITCH,TEN,VOL
UME:NEXT VOLUME:RETURN
39 REM BLINK INVERSE VIDEO
40 COUNT=30:FOR BLINKS=WON TO 3:POKE 755,3:GOSUB WAIT:POKE 7
55,TWO:GOSUB WAIT:NEXT BLINKS:RETURN
44 REM GET FROM KEYBOARD AND RETURN NUMBER AS K
45 CLOSE #TWO:OPEN #TWO,4,ZERO,"K:":GET #TWO,K:CLOSE #TWO:K=
K-48
46 IF K>9 THEN ? :? CHR$(K):RETURN
47 ? :? "  OPTION ";K:RETURN
49 REM SENSE BUTTONS (B)
50 POKE 752,WON:? :? "PRESS OPTION FOR OPTIONS"
51 B=PEEK(53279):IF B=3 THEN POSITION 8,20:? "
       ":RETURN
52 IF B=5 THEN POSITION 8,20:? "SELECT KEY WAS PRESSED":GOTO
 51
53 IF B=6 THEN POSITION 8,20:? " START KEY WAS PRESSED":GOTO
 51
54 GOTO 51
55 REM MAKE CONSOLE BUZZER SOUND
56 FOR COUNT=WON TO 50:POKE 53279,WON:NEXT COUNT:RETURN
59 REM HESITATION ROUTINE
60 FOR STALL=WON TO COUNT:NEXT STALL:RETURN
64 REM TURN ON A SOUND
65 SOUND VOICE,PITCH,DIST,VOLUME:RETURN
99 REM IF YOU GOT HERE, YOU DIDN'T ENTER THE PROGRAM CORRECT
LY
100 ? :? "CHECK YOUR PROGRAM.  YOU SHOULD NEVER":? :? "GET T
O LINE 100.":STOP
199 REM RESET ANY TRAPS, PICK A RANDOM COLOR, GOSUB, THEN TU
RN OFF THE CURSOR
200 TRAP 40000:C=RND(ZERO)*16:GOSUB KOLOR:POKE 752,WON
219 REM SUBROUTINE MENU (LINE 220...TYPE ALL WITHIN QUOTES U
SING INVERSE VIDEO
220 HEADING$="TYPE SUBROUTINE NUMBER"
240 POKE 82,9:POSITION 9,2:? HEADING$
250 ? :? "1 SET CLOCK TO ZERO"
260 ? :? "2 PRINT SECONDS"
270 ? :? "3 SHUT OFF ALL SOUND"
280 ? :? "4 ONE SHORT CLEAR SOUND"
290 ? :? "5 BLINK INVERSE VIDEO"
300 ? :? "6 SOUND THE BUZZER"
310 ? :? "7 TURN ON A SOUND"
320 ? :? "8 EXIT THIS PROGRAM"
399 REM READ FROM SCREEN AND CHANGE TO INVERSE VIDEO
400 TRAP 200:GOSUB KEY:PITCH=K*10:GOSUB ANOTE:POSITION 9,K*T
WO+TWO
402 FOR JW=9 TO 32:POSITION JW,K*TWO+TWO:GET #4,IT:IT=IT+128


404 POSITION JW,K*TWO+TWO:PUT #4,IT:NEXT JW
410 COUNT=100:GOSUB WAIT
420 GOSUB K*1000:GOSUB WAIT:GOTO 200
1000 GOSUB TIME0:GOSUB KOLOR:POSITION 9,5:? "THE CLOCK HAS B
EEN":? :? "SET AT ZERO.":GOSUB BUTTON:RETURN
2000 GOSUB SECONDS:GOSUB KOLOR:POSITION 9,5:? SEC;" SECONDS
ELAPSED.":GOSUB BUTTON:RETURN
3000 GOSUB QUIET:GOSUB KOLOR:POSITION 9,5:? "ALL SOUNDS ARE
OFF.":GOSUB BUTTON:RETURN
```

```
4000 GOSUB KOLOR:POSITION 9,5:? "TYPE PITCH (0-255)";:TRAP 4
000:INPUT PITCH:TRAP 40000:GOSUB ANOTE:RETURN
5000 GOSUB BLINK:RETURN
6000 GOSUB BUZZ:RETURN
7000 GOSUB KOLOR:? :? :? "TYPE VOICE (0-3)";:TRAP 7000:INPUT
 VOICE:TRAP 40000
7010 IF VOICE<ZERO OR VOICE>3 THEN 7000
7100 ? :? "TYPE PITCH (0-255)";:TRAP 7100:INPUT PITCH:TRAP 4
0000
7110 IF PITCH<ZERO OR PITCH>255 THEN 7100
7200 ? :? "TYPE DISTORTION AS":? "0 2 4 6 8 10 12 OR 14";:TR
AP 7200:INPUT DIST:TRAP 40000
7210 IF DIST<ZERO OR DIST>14 THEN 7200
7220 IF DIST/TWO<>INT(DIST/TWO) THEN 7200
7300 ? :? "TYPE VOLUME (1-15)";:TRAP 7300:INPUT VOLUME:TRAP
40000
7310 IF VOLUME<ZERO OR VOLUME>15 THEN 7300
7400 GOSUB SND:RETURN
7999 REM RESET LEFT MARGIN TO TWO, RESET GRAPHICS MODE TO CL
EAR THE SCREEN
8000 POKE 82,TWO:GRAPHICS ZERO:? :? "     Your wish is my co
mmand!";
8010 RUN "D:MENU"
8099 REM SET THE BACKGROUND COLOR TO DARK BLUE, CLOSE SCREEN
 IOCB
8100 SETCOLOR 2,7,ZERO:CLOSE #4:END
```

Chapter 2
Using the Paddles (PADDLE)

The PADDLE program demonstrates PADDLE value interpretation and also provides a quick and easy way to hear many of the sounds your computer can produce.

Plug the paddles into controller jack #1. The left paddle is PADDLE(0) and the right paddle is PADDLE(1). Use PADDLE(0) to change PITCH and PADDLE(1) to change VOLUME. The red trigger button on PADDLE(0) is used to change DISTORTION. Press any key to list the program onto the screen.

This little program uses a few pokes you may not have seen before. Notice that in line 140, there are pokes to locations 710 and 712. The number poked into both locations is 145. In GRAPHICS 2, these locations contain the background colors of the text and graphics windows. The number 145 produces a dark turquoise color. The same color could have been achieved using SETCOLOR 2,9,1:SETCOLOR 4,9,1. Using the POKE commands to set colors is one way to save a few bytes of memory. The number 145 was selected by multiplying the desired color (9) by 16, then adding the desired luminescence (1).

You may have also noticed POKE 77,0 in line 310. Did you ever walk away from your computer and return a few minutes later to find the screen changing colors? This is called the *attract mode.* It is a safety feature which is built in so that a constant image on the screen does not become burnt into your TV screen. If no key is pressed for more than seven minutes, the color changes will begin automatically. Since these color changes might be undesirable in games where the keyboard is not used, put POKE 77,0 in your program in a line that will be used whenever a trigger is pressed. This will allow the attract mode to function if the computer is not being used, but avoid it when it is not necessary.

```
0 REM PADDLE  (c) 1981 by Jerry White 11/6/81
1 REM DEMONSTRATION USING PADDLES TO MANIPULATE A SOUND COMM
AND
2 REM
140 GRAPHICS 2:POKE 752,1:POKE 710,145:POKE 712,145:POKE 201
,10:? CHR$(28)
150 ? ,"PADDLE(0)=PITCH"
160 ? ,"PTRIG(0)=DISTORTION"
170 ? ,"PADDLE(1)=VOLUME"
180 ? ,"PRESS ANY KEY TO END";
190 POKE 764,255
200 VOLUME=2
210 DISTORTION=10
220 DELAY=320
230 PITCH=PADDLE(0)
```

```
240 IF PTRIG(0)=0 THEN DISTORTION=DISTORTION+2:GOSUB DELAY
250 IF DISTORTION=16 THEN DISTORTION=0
260 VOLUME=INT(PADDLE(1)/10)
270 IF VOLUME>15 THEN VOLUME=15
280 IF PEEK(764)<>255 THEN GRAPHICS 0:POKE 764,255:? :? "LOA
DING MENU";:RUN "D:MENU"
290 POSITION 2,4:? #6;"sound 0,";PITCH;",";DISTORTION;",";VO
LUME;"     "
300 SOUND 0,PITCH,DISTORTION,VOLUME
310 SETCOLOR 0,DISTORTION,10:SETCOLOR 1,VOLUME,10:POKE 77,0:
GOTO 230
320 FOR WAIT=1 TO 100:NEXT WAIT:RETURN
```

Chapter 3
Using the Joystick (JOYSTICK)

JOYSTICK is a simple demonstration program which shows how to move an object by interpreting a joystick. This program alternates between draw and erase modes when the red trigger button is pressed.

To begin, plug a joystick into controller jack #0. Move the stick in any direction or press the trigger to begin.

Now move the graphics cursor around the screen. Press the trigger to toggle between drawing and erasing positions on the screen. Notice the value of STICK displayed in the text window.

To exit the program and examine BASIC code, press any key. To pause the display, press CTRL and 1. To continue, press CTRL and 1 again. NOTE: You can always abort a scrolling program display by pressing BREAK.

```
10 REM JOYSTICK by Jerry White 11/6/81
30 REM
40 GRAPHICS 17:? #6:? #6:? #6;"   JOYSTICK DEMO"
50 ? #6:? #6;"   BY JERRY WHITE"
60 ? #6:? #6:? #6:? #6;" PLUG JOYSTICK INTO"
70 ? #6:? #6;"  JACK NUMBER ONE."
80 ? #6:? #6:? #6:? #6;"HOLD TRIGGER TO DRAW"
90 ? #6;"  RELEASE TO ERASE"
100 IF STICK(0)<>15 THEN 130
110 IF STRIG(0)<>1 THEN 130
120 GOTO 100
130 X=19:Y=10:REM STARTING POSITION
140 GRAPHICS 3:POKE 764,255:POKE 752,1:SETCOLOR 2,0,0:S=STIC
K(0):GOTO 290
150 S=STICK(0):IF PEEK(764)<>255 THEN GRAPHICS 0:POKE 764,25
5:? :? "LOADING MENU";:RUN "D:MENU"
160 IF S=15 THEN 150
170 IF S=5 THEN X=X+1:Y=Y+1
180 IF S=6 THEN Y=Y-1:X=X+1
190 IF S=7 THEN X=X+1
200 IF S=9 THEN X=X-1:Y=Y+1
210 IF S=10 THEN X=X-1:Y=Y-1
220 IF S=11 THEN X=X-1
230 IF S=13 THEN Y=Y+1
240 IF S=14 THEN Y=Y-1
250 IF X<0 THEN X=0
260 IF Y<0 THEN Y=0
270 IF X>39 THEN X=39
280 IF Y>19 THEN Y=19
290 ? ,"   STICK(0)=";S
300 POKE 53279,0:REM CLICK SPEAKER
310 IF STRIG(0)=1 THEN 330
320 PX=X:PY=Y:COLOR 1:PLOT X,Y:GOTO 150
330 REM TRIGGER PRESSED
340 COLOR 0:PLOT PX,PY
350 COLOR 1:PLOT X,Y
360 PX=X:PY=Y:GOTO 150
```

Chapter 4
Using a Timer (TIMER)

The TIMER program demonstrates the use of the real time clock. The routine from line 11 thru line 13 calculates elapsed time based on the values in memory locations 18, 19, and 20. These values are converted into hours, minutes, seconds, and tenths of a second.

The following routine creates a string called T$ used to store the time in a format which will be displayed on the screen.

For demonstration purposes, this program will prompt the user for a number of hours and minutes and store these values in the variables TH (Timer Hour) and TM (Timer Minute). Set the computer's clock to zero by poking this value into locations 18, 19, and 20. T$ will be updated and displayed until the time specified by the user is equal to the elapsed time. When this is true, the program loops and rings the console bell until a key is pressed.

Using your Atari computer is an expensive way to create an alarm clock! However, this program does show you how to interpret the memory locations required to keep track of time. The program could be made more useful if a message were displayed when the alarm bell rings to remind someone to do a specific task at a given time. The routines in this demonstration could also be used to see how long it takes a player to solve a puzzle, or they could be modified to perform other timing tasks.

```
5 GOSUB 30
11 P18=PEEK(18)*256*256:P19=P18+PEEK(19)*256:P20=P19+PEEK(20
)
12 HR=INT(P20/216000):MIN=INT(P20/3600):SEC=INT(P20/60)-MIN*
60-HR*60
13 TSEC=INT(P20/6)-SEC*10-MIN*600-HR*36000
14 T$="  :  :  .":IF HR<10 THEN T$(1,1)="0":T$(2,2)=STR$(HR)
:GOTO 16
15 T$(2,2)=STR$(HR)
16 IF MIN<10 THEN T$(4,4)="0":T$(5,5)=STR$(MIN):GOTO 18
17 T$(4,5)=STR$(MIN)
18 IF SEC<10 THEN T$(7,7)="0":T$(8,8)=STR$(SEC):GOTO 20
19 T$(7,8)=STR$(SEC)
20 T$(10,10)=STR$(TSEC)
21 POSITION 5,8:? #6;T$:IF TH=HR AND MIN=TM THEN 50
22 GOTO 11
30 GRAPHICS 0:POKE 764,255:POKE 710,0:? :? :? ,"SET TIMER HO
URS";
32 TRAP 30:INPUT TH:TRAP 40000:TH=INT(TH):IF TH<0 THEN 30
40 ? :? ,"SET TIMER MINUTES";
42 TRAP 40:INPUT TM:TRAP 40000:TM=INT(TM):IF TM>59 OR TM<0 T
HEN 40
44 POKE 18,0:POKE 19,0:POKE 20,0
46 GRAPHICS 18:DIM T$(11):POSITION 7,2:? #6;"[illegible]"
48 POSITION 7,4:? #6;"hours";TH:POSITION 7,5:? #6;"minss";TM
:RETURN
```

```
50 POKE 764,255
60 IF PEEK(764)<>255 THEN GRAPHICS 0:? :? "LOADING MENU";:RU
N "D:MENU"
70 ? CHR$(253),"PRESS ANY KEY FOR MENU":GOTO 60
78 REM
80 REM TIMER TUTORIAL BY JERRY WHITE
82 REM
84 REM USE INVERSE VIDEO FOR THE FOLLOWING...
85 REM
86 REM LINE 14   THE THREE COLONS IN T$
88 REM LINE 46   TIMER:
90 REM LINE 48   hour=
92 REM LINE 48   mins=
```

Chapter 5
Interpreting the Keyboard (KEYDEMO)

Your ATARI keyboard provides a considerable amount of power for screen editing. Many ATARI 400/800 owners don't realize how lucky they are to have these features. Others simply aren't aware of all the power at their fingertips.

The KEYDEMO program needs little explanation since the program code contains many descriptive remark statements. This program was designed to point out the difference between a character's ASCII value and its internal code value. The program also shows the purpose of some important memory locations like 16, 82, 83, 201, 752, 764, and 53774. Study the program listing to see how PEEK and POKE commands are used with these locations.

It is important to note that the keyboard may be used as a device similar to the EDITOR (E:), CASSETTE (C:), DISK (D:), SCREEN (S:), or PRINTER (P:). The keyboard is a read-only device and may be opened as shown in line 360 of the KEYDEMO program. This provides the ability to GET a key as user input, bypassing the need to press RETURN when only a single-key response is required.

Here's a little helpful hint: Always POKE 764,255 before getting keyboard input. This is true no matter how the input is to be read by your program. POKE 764,255 tells the computer that the last key pressed was null, or that no key has been pressed. The user may have pressed a key long before your program is ready to read input from the keyboard. This simple POKE avoids lots of potential problems.

For example, let's assume you have a program that asks for the user's name. Before this routine is executed, the user has pressed RETURN. When your program gets to the input routine, the computer has a 12 in location 764 which means the last key pressed was RETURN. If you don't change the 12 to 255, your input routine may be bypassed. Remember to use this POKE and you will avoid this type of problem.

Speaking of POKE routines, here's one you can have some fun with.

```
100 REM KEYBOARD DEMO (KEYDEMO)
110 REM by Jerry White 10/13/81
130 REM
140 POKE 201,8:REM PRINT TAB WIDTH
150 POKE 82,2:REM LEFT MARGIN
160 POKE 83,39:REM RIGHT MARGIN
170 GOTO 270:REM BYPASS SUBROUTINE
180 REM
190 REM CLEAR SCREEN & TURN OFF CURSOR
200 GRAPHICS 0:POKE 752,1
210 REM
```

```
220 REM DISABLE BREAK KEY
230 TEST=PEEK(16)
240 IF TEST>128 THEN TEST=TEST-128:POKE 16,TEST:POKE 53774,T
EST
250 RETURN :REM FROM SUBROUTINE
260 REM
270 GOSUB 190
280 ? :? "   THIS PROGRAM READS THE KEYBOARD"
290 ? "  AND DISPLAYS THE KEY PRESSED PLUS"
300 ? ,"IT'S ATASCII VALUE."
310 ? :? :? ,"PRESS START FOR MENU":? :?
320 REM
340 REM OPEN KEYBOARD AS DEVICE #1
350 REM TO READ ONLY
360 OPEN #1,4,0,"K:"
370 REM
380 REM DISABLE CONTROL CHARACTERS
390 POKE 766,1
400 REM
410 REM POKE LAST KEY PRESSED TO NULL
420 POKE 764,255
430 REM
440 REM WAIT FOR KEY OR BUTTON PRESS
450 IF PEEK(764)=255 AND PEEK(53279)=7 THEN 450
460 BUTTON=PEEK(53279)
470 IF BUTTON=7 THEN 500
490 IF BUTTON=6 THEN 800
500 GET #1,KEY
510 IF KEY=27 THEN ? ,"ESC",KEY
520 IF KEY=28 THEN ? ,"UP",KEY
530 IF KEY=29 THEN ? ,"DOWN",KEY
540 IF KEY=30 THEN ? ,"LEFT",KEY
550 IF KEY=31 THEN ? ,"RIGHT",KEY
560 IF KEY=32 THEN ? ,"SPACE",KEY
570 IF KEY=155 THEN ? ,"RETURN",KEY;
580 IF KEY=125 THEN ? ,"CLEAR",KEY
590 IF KEY=157 THEN ? ,"INSERT SPACE ";KEY
600 IF KEY=255 THEN ? ,"INSERT LINE ";KEY
610 IF KEY=156 THEN ? ,"DELETE CHAR ";KEY
620 IF KEY=254 THEN ? ,"DELETE LINE ";KEY
630 IF KEY=127 THEN ? ,"CLR TAB",KEY
640 IF KEY=158 THEN ? ,"SET TAB",KEY
650 IF KEY=159 THEN ? ,"TAB",KEY
660 ? ,CHR$(KEY),KEY
670 KEYS=KEYS+1:IF KEYS<>5 THEN 390
680 FOR WAIT=1 TO 50:NEXT WAIT
690 REM
700 REM POSITION CURSOR
710 REM ENABLE CONTROL CHARACTERS
720 KEYS=0:POSITION 2,9:POKE 766,0
730 REM
740 REM DELETE OLD DISPLAY
750 FOR DELETE=1 TO 10:? CHR$(156);:NEXT DELETE
760 REM
770 REM GO TO BEGINNING OF MAIN LOOP
780 GOTO 390
790 REM
800 REM ENABLE BREAK KEY
810 POKE 16,64:POKE 53774,247
820 REM
830 REM ENABLE CONTROL CHARACTERS
840 POKE 766,0
850 REM
860 REM TURN ON CURSOR AND END
870 POKE 752,0
880 REM
890 REM RESET PRINT TAB WIDTH
900 POKE 201,10
910 REM
920 REM CLOSE KEYBOARD DEVICE & EXIT
930 CLOSE #1:GRAPHICS 0:? :? "LOADING MENU";:RUN "D:MENU"
```

Chapter 6
Setting the Stops (TABDEMO)

One feature of the ATARI keyboard which is rarely used is TAB SET. When items must be displayed in columns on the screen, this feature is handy.

TABDEMO provides an example that neatly formats six columns of random numbers on the screen by setting TAB STOPS.

The random numbers selected in this case are less than one hundred. The program places a blank before any number less than ten to provide right-justified amounts. (See the RJUSTIFY program to see how larger numbers such as dollar amounts may be right justified.)

The routine in line 20 clears the default tab settings. In line 40, the TABS are set. The program then goes on to place 112 random numbers onto the screen using our TAB STOPS to create a display that is neatly formatted and easy to read.

In many cases, using tabs can replace complex string manipulations. Tabs also come in handy when used to position the cursor for keyboard input. Since the TABS command can be imbedded within a string using ESC, it can also be a memory saver when used in place of blanks or spaces to position information displayed on the screen.

```
0 REM TABS DEMO BY JERRY WHITE
10 GRAPHICS 0:DIM BLANK$(5):BLANK$="     "
12 POKE 752,1:POKE 82,2:POKE 83,38:POKE 201,10:?
20 FOR X=1 TO 6:? CHR$(127);CHR$(158);:NEXT X
30 POKE 82,4:?
40 FOR TAB=1 TO 6:? BLANK$;CHR$(159);:NEXT TAB
70 ? CHR$(125):? ,," TAB  DEMO":?
80 FOR NUMBER=1 TO 112:RAN=INT(RND(0)*100)
90 IF RAN<10 THEN ? " ";
100 ? RAN;CHR$(127);:NEXT NUMBER
110 ? :POKE 82,2:? "       PRESS START FOR MENU";
120 IF PEEK(53279)<>6 THEN 120
130 GRAPHICS 0:? :? "LOADING MENU";:RUN "D:MENU"
```

Chapter 7
Right Justified Amounts (RJUSTIFY)

Many versions of BASIC have a command called PRINT USING which is used to format data for display or printing. Columns of figures, such as dollars and cents, are much easier to read if they are right-justified. PRINT USING is a handy command to have in situations like this. Unfortunately, ATARI BASIC does not have this command.

RJUSTIFY demonstrates a method of string manipulation which simulates PRINT USING. Used as a subroutine, dollar and cent amounts may be right-justified within a string and then displayed, printed, or stored in a data file.

For this demonstration, run the program, and enter the following amounts.

2.51 10 1.1

Instead of entering a fourth amount, press RETURN. Notice that the amounts line up along the right edge of the screen. 10's displayed as 10.00, and 1.1 is displayed as 1.10. This routine adds zeros to the right of the decimal point as needed to display dollars and cents in a standard format. When the user enters data, he or she need not bother typing .00 after an even dollar amount. (Of course if the decimal point and zeros are entered, no harm is done.)

This program requires very little explanation other than some variable definitions. RJA$ is the Right Justified Amount. TEMP$ temporarily stores the amount as a work string. AMOUNT$ is the amount as it is entered by the user, and B$ is a string of ten blanks. DS$ is used when printing the total, and adds a dollar sign to the left of the right-justified total.

```
100 REM RJUSTIFY BY JERRY WHITE
110 DIM RJA$(10),TEMP$(10),B$(10),AMOUNT$(10),DS$(11)
120 B$="          ":RJA$=B$:TEMP$=B$
130 GRAPHICS 0:SETCOLOR 2,0,0:POKE 201,11:POKE 82,1:POKE 83,
39
140 ? :? "ENTER AMOUNTS OF LESS THAN ONE MILLION"
150 ? :? "PRESS RETURN FOR TOTAL RIGHT JUSTIFIED":? :GOTO 20
0
160 RJA$="":WORK=INT(100*AMOUNT)/100:IF WORK=0 THEN RJA$="
    0.00":RETURN
170 IF WORK<0 THEN WORK=WORK-0.01
180 TEMP$=STR$(WORK+5.0E-03)
190 RJA$=B$(1,11-LEN(TEMP$)):RJA$(LEN(RJA$)+1)=TEMP$(1,LEN(T
EMP$)-1):RETURN
200 FOR BUZZ=0 TO 10:POKE 53279,0:NEXT BUZZ
210 ? "ENTER AMOUNT";:INPUT AMOUNT$:IF LEN(AMOUNT$)=0 THEN 2
60
220 TRAP 200:AMOUNT=VAL(AMOUNT$):TRAP 40000
```

```
230 IF AMOUNT>999999.99 OR AMOUNT<-99999.99 THEN 200
240 REM NEXT LINE FOLLOWING OPEN QUOTES=ESC-CTRL-UP ARROW TH
EN 6 SPACES
250 GOSUB 160:? ,,"↑       ";RJA$:TOTAL=TOTAL+AMOUNT:GOTO 200


260 AMOUNT=TOTAL:GOSUB 160
270 FOR DS=1 TO 10:IF RJA$(DS,DS)=" " THEN 290
280 RJA$(DS-1,DS-1)="$":DS$=RJA$:POP :GOTO 310
290 NEXT DS
300 DS$="$":DS$(LEN(DS$)+1)=RJA$
310 ? "TOTAL",,"          ";DS$
320 POKE 201,10:POKE 82,2:? :? "       PRESS OPTION TO RERUN"
:? "       PRESS START  FOR MENU";
330 IF PEEK(53279)=3 THEN RUN
340 IF PEEK(53279)<>6 THEN 330
350 GRAPHICS 0:? :? "LOADING MENU";:RUN "D:MENU"
```

Chapter 8
Dice Game-GR.O Graphics (DICE)

DICE demonstrates the use of text mode graphics characters (GRAPHICS 0), and cursor control using the arrow keys. This simple one-player dice and game is known as CRAPS, and uses text mode graphics to display a pair of dice on the screen.

Program lines 1 thru 6 contain the strings required for the display of a die. Inverse video was used to create white dice with dark circles to indicate each possible value from 1 thru 6. To display a die with a value of 1, GOSUB 1. To display a die which has a value of 2, GOSUB 2, and so on.

The program begins by creating six dice strings. DATA in line 11 creates WON$, DATA in line 12 creates TWO$, and DATA in lines 13 through 16 creates the strings for the other four dice. Lines 17 through 22 read this DATA, and create the display strings.

To enter the characters used in our dice string WON$, use the following instructions. Note these abbreviations:

AT = ATARI KEY
SP = SPACE BAR
DA = DOWN ARROW
LA = LEFT ARROW

WON$ = "AT SP SP SP AT (ESC CTRL + DA) (ESC CTRL + LA) (ESC CTRL + LA) (ESC CTRL + LA) AT SP CTRL + T SP AT (ESC CTRL + DA) (ESC CTRL + LA) (ESC CTRL + LA) (ESC CTRL + LA) AT SP SP SP AT SP"

When playing Craps, the player starts with $1000 and may bet from $1 to $100 on each turn. On the very first roll, the player wins on a 7 or 11, and loses on a 2, 3, or 12. If any other number occurs on the first roll, this becomes what is called the POINT. The player continues to roll until the POINT or a 7 is rolled. If the point is rolled before a 7 pops up, the player wins. If a 7 is rolled before the point, the player loses.

On each roll of the dice, the program goes to the subroutine beginning at line 100. Both dice change ten times to simulate rolling dice. In line 110, two random numbers are selected and stored in the variables D1 and D2. Line 120 sets a TRAP to line 130, positions the cursor, goes to the appropriate die display subroutine, and then executes an assembler subroutine.

Upon return from the assembler subroutine, a BASIC error will occur, hence the trap to the following line number. The assembler subroutine is actually part of the Operating System and creates a keyboard click. This is used to simulate the clicking of dice. Line 130 does the same thing as line 120 for the second die. Line 140 contains the NEXT of the FOR NEXT loop, resets any traps, and then returns.

The rest of the program plays the game as described in the rules. You may wish to experiment by adding an option for more than one player. If you understand the casino options of this game, you might try adding secondary betting. Have fun!

```
0 GRAPHICS 0:POKE 752,1:GOTO 10:REM DICE DEMO BY JERRY WHITE

1 ? WON$;:RETURN :REM 1
2 ? TWO$;:RETURN :REM 2
3 ? THREE$;:RETURN :REM 3
4 ? FOUR$;:RETURN :REM 4
5 ? FIVE$;:RETURN :REM 5
6 ? SIX$;:RETURN :REM 6
10 DIM WON$(17),TWO$(17),THREE$(17),FOUR$(17),FIVE$(17),SIX$
(17)
11 DATA 160,160,160,29,30,30,30,160,148,160,29,30,30,30,160,
160,160
12 DATA 148,160,160,29,30,30,30,160,160,160,29,30,30,30,160,
160,148
13 DATA 148,160,160,29,30,30,30,160,148,160,29,30,30,30,160,
160,148
14 DATA 148,160,148,29,30,30,30,160,160,160,29,30,30,30,148,
160,148
15 DATA 148,160,148,29,30,30,30,160,148,160,29,30,30,30,148,
160,148
16 DATA 148,148,148,29,30,30,30,160,160,160,29,30,30,30,148,
148,148
17 FOR DICE=1 TO 17:READ IT:WON$(DICE,DICE)=CHR$(IT):NEXT DI
CE:GOSUB 1:? :?
18 FOR DICE=1 TO 17:READ IT:TWO$(DICE,DICE)=CHR$(IT):NEXT DI
CE:GOSUB 2:? :?
19 FOR DICE=1 TO 17:READ IT:THREE$(DICE,DICE)=CHR$(IT):NEXT
DICE:GOSUB 3:? :?
20 FOR DICE=1 TO 17:READ IT:FOUR$(DICE,DICE)=CHR$(IT):NEXT D
ICE:GOSUB 4:? :?
21 FOR DICE=1 TO 17:READ IT:FIVE$(DICE,DICE)=CHR$(IT):NEXT D
ICE:GOSUB 5:? :?
22 FOR DICE=1 TO 17:READ IT:SIX$(DICE,DICE)=CHR$(IT):NEXT DI
CE:GOSUB 6
23 FOR DICE=1 TO 200:NEXT DICE
25 GRAPHICS 0:SETCOLOR 2,11,0:POKE 82,2:POKE 83,39:POKE 201,
10:POKE 752,1
30 ? :? ,"TEXT MODE GRAPHICS":? :STAKE=1000
40 ? :? ,"     CRAP GAME":GOTO 200
100 FOR TIME=1 TO 10
110 D1=INT(RND(0)*6)+1:D2=INT(RND(0)*6)+1
120 TRAP 130:POSITION 16,6:GOSUB D1:CLICK=USR(64728)
130 TRAP 140:POSITION 22,6:GOSUB D2:CLICK=USR(64728)
140 NEXT TIME:TRAP 40000:RETURN
200 SETCOLOR 4,11,0:ROLL=0:POSITION 13,12:? "YOU HAVE $";STA
KE;"  "
210 POSITION 12,14:FOR ERASE=1 TO 6:? CHR$(156);:NEXT ERASE
211 POSITION 13,14:POKE 752,0:? "PLACE YOUR BET";:GOSUB 2000
:POKE 764,255
212 TRAP 1000:INPUT BET:TRAP 40000:POKE 752,1:? " ":IF BET=0
 THEN 3000
220 BET=INT(BET):IF BET<1 OR BET>100 THEN 1000
230 ROLL=ROLL+1:IF ROLL>1 THEN 300
```

```
240 GOSUB 100:TOTAL=D1+D2:IF TOTAL=2 OR TOTAL=3 OR TOTAL=12
THEN 500
250 IF TOTAL=7 OR TOTAL=11 THEN 600
260 PPOINT=TOTAL:POSITION 13,14:? "YOUR POINT IS ";PPOINT;"
    "
270 GOSUB 900:GOTO 230
300 GOSUB 100:TOTAL=D1+D2:IF TOTAL=7 THEN 500
310 IF TOTAL<>PPOINT THEN GOSUB 900:GOTO 230
320 GOTO 600
500 SETCOLOR 4,4,4:REM CRAPS YOU LOSE
510 POSITION 13,14:? "YOU LOSE $";BET;"          ":STAKE=STAKE
-BET
520 FOR PITCH=150 TO 250:SOUND 0,PITCH,10,4:SOUND 1,PITCH+2,
10,4
522 SOUND 2,PITCH+4,10,4:SOUND 3,PITCH+5,10,4:NEXT PITCH
524 FOR OFF=0 TO 3:SOUND OFF,0,0,0:NEXT OFF
525 IF STAKE<=0 THEN GOSUB 900:GRAPHICS 0:? :? "EASY COME, E
ASY GO":? :END
530 GOSUB 900:GOTO 200
600 SETCOLOR 4,1,12:REM 7/11 OR POINT....YOU WIN
610 POSITION 13,14:? "YOU WIN $";BET;"          ":STAKE=STAKE
+BET
620 FOR PITCH=250 TO 10 STEP -10:SOUND 0,PITCH,10,10:NEXT PI
TCH
622 FOR VOLUME=15 TO 0 STEP -0.5:SOUND 0,0,2,VOLUME:SOUND 1,
1,2,VOLUME
624 SOUND 2,2,2,VOLUME:SOUND 3,3,2,VOLUME:NEXT VOLUME
630 GOSUB 900:GOTO 200
900 FOR WAIT=1 TO 200:NEXT WAIT:RETURN
1000 POKE 752,1:POSITION 13,14:? "BET FROM 1 TO 100      ":GO
SUB 2000:GOSUB 900:GOTO 210
2000 FOR BUZZ=1 TO 20:POKE 53279,0:NEXT BUZZ:RETURN
3000 GRAPHICS 0:? :? "LOADING MENU";:RUN "D:MENU"
```

Chapter 9
Mixed Graphics Modes (MODE123)

MODE123 demonstrates a screen using three different graphics modes at the same time. Combinations of these graphic modes are possible. Although MODE123 contains meaningful variable names and REM statements, you will also need the following information as you create your own DISPLAY LISTS.

DEFINITIONS:

ANTIC is a microcomputer in the Atari 400/800 which is dedicated to the video display. Antic has a user-programmable instruction set, a program called **display list**, and **display memory.**

Display list specifies display options, display modes used to interpret screen data, and where screen data may be found.

JUMP/WAIT is an instruction to ANTIC which is followed by two bytes that point to the beginning of display list.

Mode lines are groups of **scan lines.** The screen consists of 192 horizontal scan lines. A mode line is a group of scan lines based on the BASIC or ANTIC display mode being used.

O.S. modes are mode numbers that are recognized by the Operating System. O.S. modes are numbered thru 15, so there are more O.S. modes than BASIC Graphics or Display Modes.

Pixels are units of vertical distance on the TV screen. There are 192 pixels in each vertical line of display. Remember that magic number—192!

There is some valuable information on modes and screen formats on the inside back cover of your Atari Basic Reference Manual. That chart, combined with the information supplied here, will provide all you need to create your own display lists.

All screen displays must consist of 192 scan lines. In other words, the screen is 192 scan lines high. To calculate the number of scan lines in a row of a Basic Graphics Display Mode, use the Table of Modes and Screen Formats to find the number of rows in a full screen. Take, for example, good old Graphics Mode 0. Look up the number of vertical rows in a FULL SCREEN. That number is 24. Divide 192 by that number. 192/24 = 8. That means that there are 24 rows, each being 8 scan lines high, in the Graphics Mode 0 display.

Study the MODE123 demonstration program carefully. When creating a display list, you must begin with a Basic GRAPHICS command. Use the mode that requires the most memory in your new display list. In MODE123, this is GR.1 which uses more memory (RAM) than either MODE 2 or 3.

The routine at line 200 locates the DISPLAYLIST. The POKE at line 210 is used only if the mode at the top of the screen is NOT the one which uses the most memory. Since we have MODE 2 at the top, we must do this POKE to one location before the DISPLAYLIST. The following little chart was used to find the number to be POKEd, depending on the Basic Graphics Mode:

GRAPHICS MODE	1	2	3	4	5	6	7	8
	66	70	71	72	73	74	75	77	79

To find the following pokes into our new DISPLAYLIST, we use the same chart but subtract 64.

First we must determine how many lines of each mode we will have on the screen, making sure the total number of scan lines comes to the magic number of 192. This will take some thought and calculation on your part. (Nobody said this stuff was easy!) Then we use POKEs into memory creating our own ANTIC program.

Look again at the MODE123 program. Line 220 and 230 POKE the number 7 into DISPLAYLIST + 1 and + 2. That's MODE 2. The number 7 was determined by subtracting 64 from 71.

We don't have to change the next two locations since they already contain the number 70 for Graphics Mode 1.

We use the loop at line 240 to change 16 locations to Graphics Mode 3 by poking the number 8 (72-64).

Our final task is to loop our ANTIC program back to the beginning of the DISPLAYLIST. Line 250 POKEs in the number 65 which tells ANTIC to JUMP and WAIT. Lines 260 and 270 tell ANTIC where to JUMP which creates a loop. ANTIC executes this loop 60 times per second which creates the screen display.

The rest of the program is self explanatory, with the execption of those POKEs to location 87. Before printing to the screen, POKE 87 with the Graphics Mode number of the line you are about to display. This location is where the O.S. gets it's information on the display mode. Faking out the O.S. can create interesting results. Experiment and you'll see what I mean.

Still confused??? After giving a reasonable try at creating your own display list, if you still don't understand what's going on, don't feel bad. I had a hard time understanding it, too. You might try the easy way.

There's an easy way??? Now he tells me! Yes but there's also a catch. (That figures. . .)

As of August 1982 the company Educational Software was selling a great little package which contains a series of DISPLAY LIST editor programs. You don't have to know what you're doing to use them.

Good luck!

```
100 REM MODE123 (c) 1981 Jerry White
110 REM DISPLAY LIST MODIFICATION
120 REM
130 GRAPHICS 1:P=0:DING=580
140 REM
150 REM MODIFY DISPLAY LIST TO
160 REM DEMONSTRATE GRAPHICS MODES
170 REM 1, 2, AND 3 ON THE SCREEN
180 REM AT THE SAME TIME
190 REM
200 DISPLAYLIST=PEEK(560)+PEEK(561)*256+4
210 POKE DISPLAYLIST-1,71
220 POKE DISPLAYLIST+2,7
230 POKE DISPLAYLIST+3,7
240 FOR CHANGE=6 TO 21:POKE DISPLAYLIST+CHANGE,8:NEXT CHANGE

250 POKE DISPLAYLIST+22,65
260 POKE DISPLAYLIST+23,PEEK(560)
270 POKE DISPLAYLIST+24,PEEK(561)
280 SETCOLOR 0,7,10
290 SETCOLOR 1,11,10
300 SETCOLOR 2,1,10
310 SETCOLOR 3,4,10
320 SETCOLOR 4,0,0
330 POKE 87,2
340 POSITION 5,0:PRINT #6;"THREE LINES":GOSUB DING
350 POSITION 5,1:PRINT #6;"OF GRAPHICS":GOSUB DING
360 POSITION 6,2:PRINT #6;"MODE  TWO":GOSUB DING
370 FOR WAIT=1 TO 1000:NEXT WAIT
380 POKE 87,1
390 POSITION 5,3:PRINT #6;"two lines of":GOSUB DING
400 POSITION 2,4:PRINT #6;"graphics mode one":GOSUB DING
410 FOR WAIT=1 TO 1000:NEXT WAIT
420 DATA 17,13,18,12,19,11,20,11,21,12,22,13,21,14,20,15,21,
16,22,17,21,18
430 DATA 20,19,19,19,18,18,17,17
440 POKE 87,3:FOR THREE=1 TO 15
450 COLOR 3:READ X,Y:PLOT X,Y
460 FOR VOLUME=15 TO 0 STEP -1:SOUND 0,X*3,10,VOLUME:NEXT VO
LUME
470 NEXT THREE
480 FOR LOOP=1 TO 10
490 FOR X=0 TO 39:COLOR INT(RND(0)*5):PLOT X,21:NEXT X
500 NEXT LOOP
510 FOR BACK=255 TO 0 STEP -1
520 POKE 712,BACK:FOR WAIT=1 TO 10:NEXT WAIT
530 NEXT BACK
540 P=2:GOSUB DING:P=5:GOSUB DING
550 GRAPHICS 0:? :? "LOADING MENU";:RUN "D:MENU"
560 REM
570 REM DING SUBROUTINE
580 FOR VOLUME=15 TO 0 STEP -0.2:SOUND 0,P,2,VOLUME:NEXT VOL
UME:RETURN
```

DISPLAY LIST MODE CHART

MODE O.S.	NUMBER BASIC	PIXEL LINES PER MODE LINE	NUMBER OF COLORS	MEMORY BYTES PER MODE LINE	TEXT OR GRAPHIC MODE
0	—	—	—	—	* BLANK
1	—	—	—	—	JUMP
2	0	8	2	40	TEXT
3	NEW	10	2	40	TEXT
4	NEW	8	4	40	TEXT
5	NEW	16	4	40	TEXT
6	1	8	5	20	TEXT
7	2	16	5	20	TEXT
8	3	8	4	10	GRAPHIC
9	4	4	2	10	GRAPHIC
10	5	4	4	20	GRAPHIC
11	6	2	2	20	GRAPHIC
12	NEW	1	2	20	GRAPHIC
13	7	2	4	40	GRAPHIC
14	NEW	1	4	40	GRAPHIC
15	8	1	2	40	GRAPHIC

* Blank lines in Display List:

2	=	16
3	=	32
4	=	48
5	=	68
6	=	80
7	=	96
8	=	112

JUMP & WAIT = 65

This chart is being supplied as additional information. Do not confuse this chart with the Table of Modes and Screen Formats in your Atari Basic Reference Manual.

Chapter 10
Text in Graphics Mode 8 (GR8TEXT)

GR8TEXT demonstrates the placement of normal size text in the high resolution graphics window of GRAPHICS 8. The program code itself is a bit cryptic because it was designed to be as fast as possible without resorting to an assembler subroutine.

Briefly, a string called TEXT$ is interpreted and displayed at a rate of about 6 characters per second. For purposes of demonstration, the text window is used to prompt the user to enter the desired message.

Screen coordinates X and Y are used to position the graphics window display. W1 and W2 hold the address of screen memory based on the X + Y coordinates.

The string goes through a conversion from ASCII to display codes and is then actually drawn in the graphics window by the routine in line 50.

This routine allows you to add text to a high resolution graphics display. It can be used with a graph, as demonstrated (in a lower resolution graphics mode) by the GRAPH program in the next chapter.

```
0 GOTO 100:REM GR8TEXT BY JERRY WHITE
40 W1=PEEK(88)+PEEK(89)*256:W2=W1+Y*40+X:IF LEN(TEXT$)=0 THE
N 200
42 FOR ME=1 TO LEN(TEXT$):X=ASC(TEXT$(ME,ME)):IF X>127 THEN
X=X-128
44 IF X>31 AND X<96 THEN X=X-32:GOTO 48
46 IF X<32 THEN X=X+64
48 TEXT$(ME,ME)=CHR$(X):NEXT ME
50 FOR Z=1 TO LEN(TEXT$):X=ASC(TEXT$(Z,Z)):W3=57344+X*8:FOR
ME=0 TO 7:POKE W2+ME*40,PEEK(W3+ME):NEXT ME:W2=W2+1:NEXT Z
60 IF Y>100 THEN FOR Y=1 TO 300:NEXT Y:RUN
70 ? "{CLEAR}ENTER TEXT";:INPUT TEXT$:X=10:Y=Y+20:? "{CLEAR}DISPLAY RATE
= 6 CHARACTERS PER SECOND";:GOTO 40
100 DIM TEXT$(30):X=5:Y=20:GRAPHICS 8:TEXT$=" *** Graphics M
ode 8 Text ***":SETCOLOR 2,0,0:COLOR 1
110 PLOT 13,18:DRAWTO 300,18:DRAWTO 300,30:DRAWTO 13,30:DRAW
TO 13,18:GOTO 40
200 GRAPHICS 0:? :? "LOADING MENU";:RUN "D:MENU"
```

Chapter 11
Monthly Bar Graph (GRAPH)

Atari graphics can be used not only in game programs, but also to display data. For example, monthly data can be compared using a graphic display. Once numeric data is entered for each month, the GRAPH program will display this data in a colorful bar graph.

In line 120, we start by dimensioning three arrays. A$ is used as a work string, and the two numeric arrays will each store twelve items. Line 510 begins our input subroutine. One numeric amount is entered for each month. (Do not enter amounts greater than 9,999 or less than 1.)

When we return to line 130, the screen is set to graphics mode 6 which includes a text window at the bottom. The cursor is positioned using coordinate variables PX and PY. The heading is stored in the string A$. The subroutine that begins at line 170 will display the string A$ at the top of screen. Returning from the subroutine, a rectangle is drawn using COLOR 1. At line 200, the highest amount (HAMT) is determined.

The highest bar will include 65 positions. J1 is used to store this length (J1 = HAMT/65). Line 230 then determines the top position of each bar. The graph's keys are displayed on the left side of the screen. At line 360, the screen margins are set as wide as possible and abbreviations for each month are displayed in the text window. Line 400 begins drawing the outline for each bar, line 460 pokes color 3 into location 765, and an X10 command is used to fill in the bars.

As a finishing touch, a loop at line 470 sounds a tone as each bar is completed. The higher the tone, the higher the bar.

To return to the disk MENU program, simply press any key.

```
100 REM GRAPH (c) 1980 by Jerry White
110 REM ORIGINALLY PUBLISHED IN COMPUTE! MAGAZINE
111 REM P.O. BOX 5406, GREENSBORO, NC, 27406
112 REM
120 DIM A$(20),AMT(12),JW(12):GOSUB 510
130 GRAPHICS 6:SETCOLOR 2,4,4:SETCOLOR 4,4,4:SETCOLOR 0,1,10

140 PX=4:PY=0:A$="MONTHLY GRAPH":GOSUB 170
150 COLOR 1:PLOT 18,9:DRAWTO 158,9:DRAWTO 158,75:DRAWTO 18,7
5:DRAWTO 18,9
160 GOTO 200
170 DL=PEEK(560)+PEEK(561)*256:D1=PEEK(DL+4)+PEEK(DL+5)*256
180 FOR U=1 TO LEN(A$):D2=57344+((ASC(A$(U,U))-32)*8):D3=D1+
PY*20+PX+U-1
190 FOR JZ=0 TO 7:POKE D3+JZ*20,PEEK(D2+JZ):NEXT JZ:NEXT U:R
ETURN
200 FOR MON=1 TO 12:IF AMT(MON)>HAMT THEN HAMT=AMT(MON)
210 NEXT MON
220 J1=HAMT/65
```

```
230 FOR MON=1 TO 12:TAMT=75-(AMT(MON)/J1):JW(MON)=INT(TAMT):
NEXT MON
240 IF HAMT>=10000 THEN K=INT(HAMT/1000):GOTO 280
250 IF HAMT>=1000 THEN K=INT(HAMT/100):GOTO 280
260 IF HAMT>=100 THEN K=INT(HAMT/10):GOTO 280
270 K=INT(HAMT)
280 KD=K/5:K2=INT(K-KD):K3=INT(K-(KD*2))
290 K4=INT(K-(KD*3)):K5=INT(K-(KD*4))
300 A$=STR$(K):PX=2-LEN(A$):PY=10:GOSUB 170
310 IF K<5 OR K>99 THEN 360
320 A$=STR$(K2):PX=2-LEN(A$):PY=24:GOSUB 170
330 A$=STR$(K3):PX=2-LEN(A$):PY=38:GOSUB 170
340 A$=STR$(K4):PX=2-LEN(A$):PY=52:GOSUB 170
350 A$=STR$(K5):PX=2-LEN(A$):PY=66:GOSUB 170
360 POKE 82,0:POKE 83,40:POKE 752,1
370 ? :? "  K  J  F  M  A  M  J  J  A  S  O  N  D"
380 ? "  E  A  E  A  P  A  U  U  U  E  C  O  E"
390 ? "  Y  N  B  R  R  Y  N  L  G  P  T  V  C"
400 FOR MON=1 TO 12:JY=MON-1
410 PLOT 18+(JY*12),JW(MON)
420 DRAWTO 25+(JY*12),JW(MON)
430 DRAWTO 25+(JY*12),75
440 DRAWTO 18+(JY*12),75
450 POSITION 18+(JY*12),JW(MON)
460 POKE 765,3:XIO 18,#6,0,0,"S:"
470 FOR VOL=10 TO 0 STEP -1:SOUND 0,JW(MON),10,VOL:NEXT VOL:
NEXT MON
480 POKE 82,2:POKE 83,39:POKE 764,255
490 IF PEEK(764)<>255 OR PEEK(53279)<>7 THEN GRAPHICS 0:POKE
 764,255:? :? "LOADING MENU";:RUN "D:MENU"
500 GOTO 490
510 GRAPHICS 0:SETCOLOR 2,0,0:SETCOLOR 1,0,10:SETCOLOR 4,0,0


520 ? :? ," MONTHLY GRAPH"
530 ? :? "   TYPE AMOUNTS FOR EACH MONTH:":?
540 ? "   DO NOT USE NEGATIVE AMOUNTS":?
550 TRAP 550:? ,"JAN=";:INPUT JAN:AMT(1)=JAN:TRAP 40000
560 TRAP 560:? ,"FEB=";:INPUT FEB:AMT(2)=FEB:TRAP 40000
570 TRAP 570:? ,"MAR=";:INPUT MAR:AMT(3)=MAR:TRAP 40000
580 TRAP 580:? ,"APR=";:INPUT APR:AMT(4)=APR:TRAP 40000
590 TRAP 590:? ,"MAY=";:INPUT MAY:AMT(5)=MAY:TRAP 40000
600 TRAP 600:? ,"JUN=";:INPUT JUN:AMT(6)=JUN:TRAP 40000
610 TRAP 610:? ,"JUL=";:INPUT JUL:AMT(7)=JUL:TRAP 40000
620 TRAP 620:? ,"AUG=";:INPUT AUG:AMT(8)=AUG:TRAP 40000
630 TRAP 630:? ,"SEP=";:INPUT SEP:AMT(9)=SEP:TRAP 40000
640 TRAP 640:? ,"OCT=";:INPUT OCT:AMT(10)=OCT:TRAP 40000
650 TRAP 650:? ,"NOV=";:INPUT NOV:AMT(11)=NOV:TRAP 40000
660 TRAP 660:? ,"DEC=";:INPUT DEC:AMT(12)=DEC:TRAE:TRAP 40H
"
"
"
"
670 RETURN
```

Chapter 12
Sorting a String (SORTDEMO)

Sorting data so it is in alphabetical sequence makes data easier to find, and is often essential.

SORTDEMO is an adaption of the Shell-Metzner sort. It will sort both alphabetic and numeric data. Since the ASCII value of any number is less than the ASCII value of the letter A, numbers will sort ahead of letters.

This sort is set up to handle a maximum of 100 records, but can be modified to suit your needs. Each record can contain no more than 30 characters.

If you need more than 30 charcters, these simple modifications can be made. Any reference to 30 will be changed for the number of characters you need. For instance, if you want a 35 character record, change every 30 to a 35. Any references to 29 should be changed to 34. In line 130, A$(3000) would be changed to A$(3500) in both instances. The number within the parenthesis is the length of a record times 100. If you need more than 100 records, this number will be the length of the record times the maximum number of records.

There is no need to enter all records at once; you do not have to type in all 30 characters. Type in a few names and numbers, hitting RETURN after each one. Each data item will be given a record number. When you have finished entering the data, press **RETURN** again.

The data you entered will be sorted and displayed on the screen. By holding down **CTRL** and the number **1**, you can pause. To continue screen scrolling, press **CTRL** and **1** again.

After the data has been displayed, press **OPTION** to enter more data. Press **RETURN** when all data is entered to sort again.

```
100 REM ATARI BASIC STRING SORT
110 REM TUTORIAL BY JERRY WHITE
112 REM ORIGINALLY PUBLISHED IN COMPUTE MAGAZINE
113 REM P.O. BOX 5406, GREENSBORO, NC, 27403
120 REM *** SETUP ***
130 DIM A$(3000),B$(30),C$(30):A$(1)=" ":A$(3000)=" ":A$(2)=
A$:B$="":C$=""
140 GRAPHICS 0:SETCOLOR 2,0,0:POKE 82,2:GOTO 320
150 REM *** SORT A$ ***
160 T=INT(T/3)+1:FOR L1=1 TO REC-T:FOR L2=L1 TO 1 STEP -T
170 IF A$(L2*30-29,L2*30)<=A$((L2+T)*30-29,(L2+T)*30) THEN 2
10
180 C$=A$(L2*30-29,L2*30):A$(L2*30-29,L2*30)=A$((L2+T)*30-29
,(L2+T)*30)
```

```
190 A$((L2+T)*30-29,(L2+T)*30)=C$
200 NEXT L2
210 SOUND 0,REC+10-L1,10,2:NEXT L1
220 IF T>1 THEN 160
230 REM *** DISPLAY SORTED DATA ***
240 SOUND 0,0,0,0:? CHR$(125):? ,"*** SORTED DATA ***"
250 FOR ME=1 TO REC:? A$(ME*30-29,ME*30):NEXT ME
260 REM *** CONTINUE OPTION ***
270 ? :? "        PRESS OPTION TO ADD DATA":? "        PRESS S
TART FOR MENU";
280 IF PEEK(53279)=3 THEN 330
290 IF PEEK(53279)=6 THEN GRAPHICS 0:? :? "LOADING MENU";:RU
N "D:MENU"
300 GOTO 280
310 REM *** DATA ENTRY ***
320 ? :? "ENTER UP TO 100 RECORDS TO BE SORTED:"
330 REC=REC+1:? :? "ENTER RECORD ";REC:B$="":INPUT B$:LB=LEN
(B$)
340 IF LB=0 THEN REC=REC-1:T=REC:? ,"*** S O R T I N G ***":
GOTO 160
350 A$(REC*30-29,REC*30-29+LB)=B$
360 GOTO 330
```

Chapter 13
Musical End or Exit Routine (MUSICEND)

Many Atari commercial programs begin and end with music. This program plays an ending tune of seven notes. If you do not have a background in music, this program will also help you to understand the terms *whole note, half note, quarter note,* and *eighth note.* As indicated by their names, the duration of a half note is half that of a whole note. The duration of a quarter note is half that of a half note, and the duration of an eighth note is half that of a quarter note.

The tempo of a song can be based on the duration of the whole note. BASIC can then be used to calculate the duration of other notes using simple division.

Run the program. You will be asked to enter the duration of a whole note. For this demonstration, enter the number 20. Seven notes will be played. Press OPTION to run it again. This time, enter the number 40 as the whole note duration. Notice that all notes are held twice as long which makes the song play at half the speed.

MUSICEND uses one of the five countdown timers to kill time while a note is being played or held. The variable HOLD is used to set the timer at memory location 540. In line 180, we POKE the value of HOLD into this location. We loop at line 190 until the value in location 540 is equal to zero. When we fall through to line 200, we RETURN from this subroutine.

Whenever a value greater than zero is poked into location 540, the Atari begins counting down to zero at the rate of 60 per second until this value becomes zero. A unit of time 1/60th of a second long is known as a *jiffy.* Any integer from zero to 255 may be poked into location 540. The routine in lines 180 through 200 may be used for any delay up to 255 jiffies or four and one quarter seconds.

Since the speed of a FOR/NEXT loop varies, using a countdown timer is a much more accurate method of creating a time delay.

The SOUND capabilities of the Atari are truly amazing. The BASSNOTE and SOUNDEMO programs in this package further demonstrate the use of sound. If you would like additional information on sound and music, look into Jerry White's Music Lessons from Swifty Software, TRICKYTUTORAL #6 (SOUNDS) from Educational Software, and the SOUND chapter in the APX publication De Re ATARI.

```
0 REM MUSICEND BY JERRY WHITE
100 GRAPHICS 0:SETCOLOR 2,15,0:POKE 82,1:POKE 83,39
110 ? :? :? "      MUSICAL END OF JOB ROUTINE":? :? ," by Jer
ry White"
120 ? :? :? "ENTER THE DURATION OF A WHOLE NOTE";
130 TRAP 120:INPUT WN:TRAP 40000
140 IF WN>255 THEN WN=255
150 POKE 752,1:? :? ," WHOLE NOTE=";WN:HN=INT(WN/2):? :? ,"
HALF NOTE=";HN
160 QN=INT(WN/4):? :? ," QUARTER NOTE=";QN
170 EN=INT(WN/8):? :? ," EIGHTH NOTE=";EN:GOTO 210
180 POKE 540,HOLD
190 IF PEEK(540)>0 THEN 190
200 RETURN
210 SOUND 0,48,12,8:HOLD=HN:GOSUB 180
220 SOUND 0,0,0,0:HOLD=QN:GOSUB 180
230 SOUND 0,63,12,8:HOLD=QN:GOSUB 180
240 SOUND 0,0,0,0:HOLD=EN:GOSUB 180
250 SOUND 0,63,12,8:HOLD=QN:GOSUB 180
260 SOUND 0,0,0,0:HOLD=EN:GOSUB 180
270 SOUND 0,57,12,8:HOLD=HN:GOSUB 180
280 SOUND 0,0,0,0:HOLD=QN:GOSUB 180
290 SOUND 0,63,12,8:HOLD=HN:GOSUB 180
300 SOUND 0,0,0,0:HOLD=WN:GOSUB 180
310 SOUND 1,51,12,8:SOUND 0,47,10,2:SOUND 2,37,10,2:SOUND 3,
31,10,2
320 ? :? :? ," THAT'S";:HOLD=HN:GOSUB 180
330 GOSUB 360:? " ALL":HOLD=QN:GOSUB 180
340 SOUND 1,48,12,8:SOUND 0,45,10,2:SOUND 2,35,10,2:SOUND 3,
29,10,2
350 HOLD=WN:GOSUB 180:GOSUB 360:POKE 82,2:POKE 752,0:? :? :G
OTO 370
360 FOR OFF=0 TO 3:SOUND OFF,0,0,0:NEXT OFF:RETURN
370 ? :? "      PRESS OPTION TO RERUN":? "      PRESS START
 FOR MENU";
380 IF PEEK(53279)=3 THEN RUN
390 IF PEEK(53279)<>6 THEN 380
400 GRAPHICS 0:? :? "LOADING MENU";:RUN "D:MENU"
```

Chapter 14
Deep Bass Sounds (BASSNOTE)

BASSNOTE is self-documenting. It demonstrates the use of distortion level 12 in the SOUND command, and plays part of the bass line of the theme from the Barney Miller TV show.

Why use distortion level 12? Enter this little BASIC routine and you'll hear a much deeper tone in level 12 than is possible in level 10.

```
10 GRAPHICS 0:?:?"This is the lowest pitch possible in level 10."
20 FOR DELAY = 1 TO 500:NEXT DELAY:?:LIST 30
30 FOR VOLUME = 15 TO 0 STEP-0.2:SOUND 0,255,10,VOLUME:NEXT
VOLUME
40 ?:?"NOW HEAR THIS. . .":LIST 50:FOR DELAY = 1 TO 500:NEXT
DELAY
50 FOR VOLUME = 15 TO 0 STEP-0.2:SOUND 0,102,12,VOLUME:NEXT
VOLUME
```

BASSNOTE will display the notes and pitch numbers for two octaves of low bass notes. This information is not in your Reference Manual, so save it for future reference. As the demonstration song is played, the actual SOUND commands being used will be displayed on the screen.

```
0 REM BASSNOTE TUTORIAL BY JERRY WHITE
1 REM ORIGINALLY PUBLISHED IN A.N.A.L.O.G. 400/800 MAGAZINE
2 REM
20 GOSUB 600:GOSUB 250:GOTO 105
40 SOUND 0,0,0,0:READ PITCH:D=12:V=14:SETCOLOR 2,PITCH,0:SOU
ND 0,PITCH,D,V
42 POSITION 10,20:? "  SOUND 0,";PITCH;",";D;",";V;"     ":RE
TURN
50 FOR HOLD=1 TO 200:NEXT HOLD:SOUND 0,0,0,0:PITCH=0:D=0:V=0
:GOSUB 42:RETURN
70 FOR HOLD=1 TO 50:NEXT HOLD:RETURN
80 FOR HOLD=1 TO 25:NEXT HOLD:SOUND 0,0,0,0:RETURN
100 DATA 102,90,85,82,75,72,67,67,60,67,60,67,75,67,51,60,75
,90
105 FOR TIME=1 TO 2:GOSUB 40:GOSUB 50:GOSUB 50
110 GOSUB 40:GOSUB 70:GOSUB 40:GOSUB 70
120 GOSUB 40:GOSUB 50:GOSUB 50
130 GOSUB 40:GOSUB 70:GOSUB 40:GOSUB 70
140 GOSUB 40:GOSUB 50:GOSUB 50
150 FOR QUARTERNOTE=1 TO 8:GOSUB 40:GOSUB 70:NEXT QUARTERNOT
E
190 GOSUB 40:GOSUB 80:GOSUB 40:GOSUB 80
200 GOSUB 40:GOSUB 80:RESTORE :NEXT TIME
220 RESTORE :GOSUB 40:GOSUB 50:POKE 752,0:? "       PRESS OPT
ION TO RERUN":? "          PRESS START  FOR MENU";
230 IF PEEK(53279)=3 THEN RUN
235 IF PEEK(53279)<>6 THEN 230
240 ? CHR$(125):? :? "LOADING MENU";:RUN "D:MENU"
250 ? :? ," PITCH = NOTE":GOSUB 700
300 ? :? "25=E","27=D#","28=D ","30=C#"
310 ? "31=C ","33=B ","36=A#","37=A "
320 ? "40=G#","42=G ","45=F#","48=F "
```

```
330 ? "51=E ","55=D#","57=D ","60=C#"
340 ? "63=C ","67=B ","72=A#","75=A "
350 ? "82=G#","85=G ","90=F#","97=F "
360 ? "102=E":GOSUB 700
380 ? :? "  THE ATARI BASIC SOUND COMMAND:"
400 ? :? "SOUND VOICE,PITCH,DISTORTION,VOLUME":GOSUB 700:RET
URN
600 GRAPHICS 0:POKE 752,1:GOSUB 700:? "       THE THEME FROM
BARNEY MILLER"
620 ? :? "BASSNOTES USING SOUND DISTORTION 12":GOSUB 700:RET
URN
700 FOR CTRLR=2 TO 36:? CHR$(18);:NEXT CTRLR:RETURN
790 REM ********************************
800 REM * D=DISTORTION V=VOLUME      *
810 REM * GOSUB 50 FOR WHOLE NOTE    *
820 REM * GOSUB 70 FOR QUARTER NOTE  *
830 REM * GOSUB 80 FOR EIGHTH NOTE   *
840 REM * GOSUB 700 TO DRAW A LINE   *
850 REM ********************************
```

Chapter 15
Sound Effects (SOUNDEMO)

SOUNDEMO contains its own documentation, which is displayed as the program runs. After running the program, study the program listing. You may use any of the sound effect routines of SOUNDEMO in your own programs.

The sound related programs in this package barely scratch the surface of this subject. Experimentation is the best way to learn. Substitute your own values in some of the many FOR/NEXT loops of this program. Most of these routines were discovered using the trial and error method.

Experiment using pokes to the SOUND registers found in memory locations 53760 through 53768. Locations 53760 and 53761 control voice zero. 53762 and 53763 control voice one, 53764 and 53765 control voice two, and 53766 and 53767 control voice three. Poking various values into location 53768 can create strange effects in any and all voices.

Although a programmer can achieve a much wider range of effects with machine language, there is certainly a wide range of sounds available to the BASIC programmer.

```
0 REM SOUNDEMO (c) 1981 by Jerry White
10 GRAPHICS 0:POKE 710,0:POKE 82,2:POKE 752,1:GOSUB 40:GOTO
100
19 REM TURN OFF SOUNDS
20 FOR OFF=0 TO 3:SOUND OFF,0,0,0:NEXT OFF:RETURN
29 REM TIME DELAY T=60ths of a second
30 POKE 540,T
31 IF PEEK(540)<>0 THEN 31
32 GOSUB 20:RETURN
39 REM NEW SCREEN HEADING
40 ? CHR$(125):? :? "     SOUND ROUTINES BY JERRY WHITE":? :?
 :RETURN
50 POSITION 6,21:? "  PRESS START TO CONTINUE":POKE 53279,0
52 IF PEEK(53279)<>6 THEN 52
53 RETURN
100 ? "    One simple clear sound can be made":? "to sound di
fferently just by altering"
110 ? "it's duration and volume.    Each of":? "the following
 routines play the same"
120 ? "note (middle C).  Notice the how the":? "sound is alt
ered, just by changing"
130 ? "the volume and duration.":? :? "    The sound command
used to play"
140 ? "middle C is SOUND 0,121,10,8.  The 4":? "variables ar
e:":? :? "SOUND VOICE,PITCH,DISTORTION,VOLUME."
150 GOSUB 50:GOSUB 40:LIST 160
160 SOUND 0,121,10,8:T=1:GOSUB 30
170 T=120:GOSUB 30:? :LIST 180
180 FOR TIME=1 TO 10:SOUND 0,121,10,8:T=1:GOSUB 30:NEXT TIME


190 T=120:GOSUB 30:? :LIST 200
200 SOUND 0,121,10,8:T=20:GOSUB 30
210 T=120:GOSUB 30:? :LIST 220
220 FOR TIME=1 TO 10:SOUND 0,121,10,8:T=20:GOSUB 30:NEXT TIM
```

```
E
230 GOSUB 50:GOSUB 40:LIST 240
240 FOR TIME=1 TO 3:FOR V=1 TO 15:SOUND 0,121,10,V:NEXT V:FO
R V=15 TO 0 STEP -1:SOUND 0,121,10,V:NEXT V:NEXT TIME
250 T=120:GOSUB 30:? :LIST 260
260 FOR TIME=1 TO 3:FOR V=15 TO 0 STEP -0.2:SOUND 0,121,10,V
:NEXT V:NEXT TIME
270 T=120:GOSUB 50:GOSUB 40:? "   The following routines als
o use":? "the clear distortion level 10.  The"
280 ? "difference will be changing the pitch":? "with FOR NE
XT loops.":GOSUB 50:GOSUB 40:LIST 290
290 FOR P=0 TO 255:SOUND 0,P,10,8:NEXT P:FOR P=255 TO 0 STEP
 -1:SOUND 0,P,10,8:NEXT P:GOSUB 20
300 T=120:GOSUB 30:LIST 310
310 FOR P=0 TO 255 STEP 5:SOUND 0,P,10,8:NEXT P:FOR P=255 TO
 0 STEP -5:SOUND 0,P,10,8:NEXT P:GOSUB 20
320 GOSUB 50:GOSUB 40:? "   Now we will use multiple voices"
:? "which will force us to use more than"
330 ? "one line in many routines.  We will":? "still use onl
y the clear sound of"
340 ? "distortion level 10 and change the":? "pitch, the vol
ume, or both with the"
350 ? "FOR NEXT loops.":GOSUB 50:GOSUB 40:LIST 360
360 FOR P=1 TO 255:SOUND 0,P,10,8:SOUND 1,255-P,10,8:NEXT P:
GOSUB 20
370 T=120:GOSUB 30:? :LIST 380
380 FOR P=0 TO 255 STEP 5:SOUND 0,P,10,8:SOUND 1,255-P,10,8:
NEXT P:GOSUB 20
390 GOSUB 50:GOSUB 40:LIST 400,410
400 FOR V=15 TO 0 STEP -0.5:FOR P=3 TO 10:SOUND 0,P,10,V:NEX
T P
410 FOR P=13 TO 25 STEP 12:SOUND 1,P,10,V:NEXT P:NEXT V
420 T=120:GOSUB 30:LIST 430,460
430 FOR V=0 TO 15:FOR P=200 TO 206 STEP 2:SOUND 0,P,10,V:NEX
T P
440 FOR P=208 TO 214 STEP 2:SOUND 1,P,10,V:NEXT P
450 FOR P=216 TO 222 STEP 2:SOUND 2,P,10,V:NEXT P
460 FOR P=224 TO 232 STEP 2:SOUND 3,P,10,V:NEXT P:NEXT V:GOS
UB 20
470 GOSUB 50:GOSUB 40:? "   The key to using Atari's sounds"
:? "with Basic is the FOR NEXT loop and"
480 ? "experimentation.  These last few":? "routines will cr
eate a variety of"
490 ? "sound effects by combining different":? "distortion l
evels and voices."
500 ? :? "   I hope you've enjoyed this little":? "demonstra
tion and that you will soon"
510 ? "be creating your own sound effects.":GOSUB 50:GOSUB 4
0:LIST 520
520 FOR TIME=1 TO 5:FOR V=15 TO 0 STEP -2:SOUND 0,V,4,V:SOUN
D 1,13,4,V:SOUND 2,V,0,V:NEXT V:NEXT TIME:GOSUB 20
530 T=120:GOSUB 30:? :LIST 540
540 FOR V=15 TO 0 STEP -0.5:FOR P=6 TO 0 STEP -1:SOUND 0,P,2
,V:NEXT P:NEXT V
550 GOSUB 50:GOSUB 40:LIST 560,570
560 FOR TIME=1 TO 30:SOUND 0,25,10,2:SOUND 1,0,2,2:FOR T2=1
TO 2:SOUND 0,25,10,8:SOUND 1,0,2,8:NEXT T2
570 SOUND 0,0,0,0:SOUND 1,0,0,0:NEXT TIME:FOR V=8 TO 0 STEP
-0.1:SOUND 0,25,10,V:SOUND 1,0,2,V:NEXT V
580 T=120:GOSUB 30:? :LIST 590
590 FOR V=15 TO 0 STEP -0.2:SOUND 0,25,4,V:SOUND 1,240,2,V:S
OUND 2,255-V*14,0,V:NEXT V
600 GOSUB 50:GOSUB 40:LIST 610,620
610 FOR V=15 TO 0 STEP -1:FOR P=10 TO 60 STEP 5:SOUND 0,P,10
,V:NEXT P
620 FOR P=70 TO 140 STEP 10:SOUND 1,P,10,V:NEXT P:FOR P=210
TO 10 STEP -50:SOUND 2,P,10,V:NEXT P:NEXT V
630 T=120:GOSUB 30:? :LIST 640
640 FOR V=15 TO 0 STEP -0.05:SOUND 0,51,12,V:SOUND 1,102,12,
V:SOUND 2,51,12,V:SOUND 3,102,12,V:NEXT V
650 GOSUB 40:? :? :? "      PRESS OPTION TO RERUN":? :? "
    PRESS START  FOR MENU"
660 IF PEEK(53279)=3 THEN RUN
670 IF PEEK(53279)<>6 THEN 660
680 ? :? "      LOADING MENU.....":RUN "D:MENU"
```

Chapter 16
Binary to Decimal Conversions (BINCONV)

Situations sometimes demand that a programmer convert data from binary to decimal numbers, or from decimal to binary. Creating Players using Player Missile Graphics is such a situation. If you decide to create your own Players using the PMDEMO program, this BINCONV program will be helpful.

Use graph paper to design a player. Section off a strip eight boxes wide. Use a pencil to shade the boxes required to created the desired shape. Think of the shaded boxes as ones and the blank boxes as zeros.

When you are satisified with your player, run this program. Select Binary to Decimal and enter the first binary number to be converted. This should be the top line of your player: eight zeros. This program expects eight digits for each binary number you enter. As each number is entered, it will replace the line of eight inverse video spaces or cursors you see on the screen. Since this program checks only for zeros, any position not entered will be assumed to be a binary 1. BINCONV also checks the first position you type to see if it is the letter E or the letter R. E is used to end the program and R is used to restart it.

As you convert each line of your binary player into a decimal number, write down the decimal numbers. These numbers will be entered into the PMDEMO program, which will convert the decimal numbers into STRING$ format.

The BINCONV program is easy to follow. B$ holds the binary number while D$ is the decimal number string. Numeric variables D, D2, and GC are used to store the decimal number, that value divided by 2, and the character entered through the GET Character routine (Line 34).

```
0 REM BINARY TO DECIMAL AND DECIMAL TO BINARY CONVERSION PRO
GRAM
1 REM (c) 1981 by Jerry White
2 REM
3 REM ORIGINALLY PUBLISHED IN COMPUTE! MAGAZINE
4 REM P.O. BIX 5406, GREENSBORO, NC, 27403
5 REM
20 DIM D$(3),B$(8):B$="00000000"
30 GRAPHICS 0:SETCOLOR 2,0,0:POKE 752,1:POSITION 5,5
32 ? "TYPE B TO CONVERT FROM BINARY":? :? "   TYPE D TO CONV
ERT FROM DECIMAL"
34 OPEN #1,4,0,"K:":GET #1,GC:CLOSE #1
36 IF GC=66 THEN 50
38 IF GC=68 THEN 500
40 GOTO 34
50 ? CHR$(125):SETCOLOR 2,0,0:DV=0:POKE 752,1:? :? "BINARY T
O DECIMAL CONVERSION PROGRAM:":GOTO 355
96 REM
```

```
97 REM LINE 100 HAS 8 INVERSE VIDEO SPACES FOLLOWED BY
98 REM TWO NORMAL VIDEO SPACES FOLLOWED BY TWO UP ARROWS
99 REM WITHIN THE QUOTES
100 ? :? ,"        ↑↑"
101 REM
110 ? :? ,;:INPUT B$:IF B$(1,1)="E" THEN GRAPHICS 0:? :? "LO
ADING MENU";:RUN "D:MENU"
112 IF B$(1,1)="R" THEN RUN
120 DV=0:TRAP 360
200 IF B$(1,1)="0" THEN 220
210 DV=DV+128
220 IF B$(2,2)="0" THEN 240
230 DV=DV+64
240 IF B$(3,3)="0" THEN 260
250 DV=DV+32
260 IF B$(4,4)="0" THEN 280
270 DV=DV+16
280 IF B$(5,5)="0" THEN 300
290 DV=DV+8
300 IF B$(6,6)="0" THEN 320
310 DV=DV+4
320 IF B$(7,7)="0" THEN 340
330 DV=DV+2
340 IF B$(8,8)="0" THEN 355
350 DV=DV+1
355 POSITION 2,3:? :? ,"BINARY ";B$;"="
356 POSITION 26,6:? "    ":POSITION 2,5
360 ? :? ,"DECIMAL VALUE=";DV
370 ? :? ,"TYPE E TO END":? ,"TYPE R TO RERUN":? ,"OR TYPE A
 BINARY NUMBER."
400 TRAP 40000:GOTO 100
500 ? CHR$(125):SETCOLOR 2,0,0:POKE 752,1:? :? "DECIMAL TO B
INARY CONVERSION PROGRAM:":GOTO 800
508 REM
509 REM LINE 510 HAS 5 SPACES THEN TWO UP ARROWS WITHIN THE
QUOTES
510 ? :? ,"     ↑↑";
511 REM
520 ? :? :? ,;:INPUT D$:IF D$(1,1)="E" THEN GRAPHICS 0:? :?
"LOADING MENU";:RUN "D:MENU"
530 IF D$(1,1)="R" THEN RUN
540 TRAP 500:IF VAL(D$)>255 THEN 500
545 DN=VAL(D$):B$=""
550 FOR DIGIT=1 TO 8
560 D2=VAL(D$)/2:D=INT(D2)
570 IF D2=INT(D2) THEN B$(9-DIGIT,9-DIGIT)="0":GOTO 590
580 B$(9-DIGIT,9-DIGIT)="1"
590 D$=STR$(D)
600 NEXT DIGIT
800 POSITION 2,3:? :? ,"DECIMAL ";DN;"=     "
810 ? :? ,"BINARY VALUE=";B$
820 ? :? ,"TYPE E TO END":? ,"TYPE R TO RERUN":? ,"OR TYPE A
 DECIMAL NUMBER."
830 TRAP 40000:GOTO 510
```

Chapter 17
Player Missile Strings (PMDEMO)

To Enter This Program Manually

If you purchased the disk or cassette version of this package, you obviously will not have to manually enter the programs. If you must enter the programs manually, you will need the following information.

PMDEMO stores data for players and assembly subroutines in strings. These strings consist of ASCII characters which cannot be printed on paper. In order to create these strings, a routine has been provided in program lines 9500 through 9960 to read DATA statements and create the required strings. This routine actually generates BASIC code which will become part of the PMDEMO program.

This routine is used to create program lines 10, 65 through 90, 5070, and 5080. Before creating these lines, enter the rest of this program and save it on cassette. Be sure to read program listing lines 10000 through 10160 and note the line numbers and text which must be entered using inverse video.

Before using the routine for creating strings, read the REM statements from line 9500 through line 9680. You will need the documentation pages that contain the PMDEMO DATA FOR CREATING STRINGS.

The following instructions will walk you through the first DATA to STRING conversion:

Note the first two lines on the first page of PMDEMO DATA:

```
DATA FOR C$ IN LINE 10
9900 DATA 0,40,2,136,112,0
```

Enter line 9900 exactly as it appears above and press RETURN. The routine beginning at line 9700 makes the data-to-string conversion, but first you must answer two questions. Type GOTO 9700 and press RETURN.

The first prompt asks for the line number of the string (not 9900). The line number for C$ is 10, so type 10 and press RETURN.

The second prompt asks for the string name including the dollar sign, so type C$ and press RETURN.

Line 10 should now be part of the PMDEMO program. Just to make sure, type LIST 10 and press RETURN. If line 10 does not appear on the

screen, check the program in memory from line 9700 through line 9960 against the program listing, correct any errors, and try again.

That's one down and 28 to go. The second string will be line 65 and the string name will be DAT$. There are 26 DAT$ strings. Once all DAT$ strings are entered, two more strings called ERASE$ and PM$ are created. You will have to enter three lines of data for PM$ (9900, 9901, and 9902).

```
PMDEMO DATA FOR CREATING STRINGS:

DATA FOR C$ IN LINE 10
9900 DATA 0,40,2,136,112,0

DATA FOR DAT$ IN LINE 65
9900 DATA 0,100,16,0,0,24,24,60,60,102,102,102,102,126,126,102,102,0,0,0

DATA FOR DAT$ IN LINE 66
9900 DATA 0,100,16,0,0,124,124,102,102,124,124,102,102,102,102,124,124,0,0,0

DATA FOR DAT$ IN LINE 67
9900 DATA 0,100,16,0,0,60,60,102,102,96,96,96,96,102,102,60,60,0,0,0

DATA FOR DAT$ IN LINE 68
9900 DATA 0,100,16,0,0,120,120,108,108,102,102,102,102,108,108,120,120,0,0,0

DATA FOR DAT$ IN LINE 69
9900 DATA 0,100,16,0,0,126,126,96,96,124,124,96,96,96,96,126,126,0,0,0

DATA FOR DAT$ IN LINE 70
9900 DATA 0,100,16,0,0,126,126,96,96,124,124,96,96,96,96,96,96,0,0,0

DATA FOR DAT$ IN LINE 71
9900 DATA 0,100,16,0,0,62,62,96,96,96,96,110,110,102,102,62,62,0,0,0

DATA FOR DAT$ IN LINE 72
9900 DATA 0,100,16,0,0,102,102,102,102,126,126,102,102,102,102,102,102,0,0,0

DATA FOR DAT$ IN LINE 73
9900 DATA 0,100,16,0,0,126,126,24,24,24,24,24,24,24,24,126,126,0,0,0

DATA FOR DAT$ IN LINE 74
9900 DATA 0,100,16,0,0,6,6,6,6,6,6,6,6,102,102,60,60,0,0,0

DATA FOR DAT$ IN LINE 75
9900 DATA 0,100,16,0,0,102,102,108,108,120,120,120,120,108,108,102,102,0,0,0

DATA FOR DAT$ IN LINE 76
9900 DATA 0,100,16,0,0,96,96,96,96,96,96,96,96,96,96,126,126,0,0,0

DATA FOR DAT$ IN LINE 77
9900 DATA 0,100,16,0,0,99,99,119,119,127,127,107,107,99,99,99,99,0,0,0

DATA FOR DAT$ IN LINE 78
9900 DATA 0,100,16,0,0,102,102,118,118,126,126,126,126,110,110,102,102,0,0,0

DATA FOR DAT$ IN LINE 79
9900 DATA 0,100,16,0,0,60,60,102,102,102,102,102,102,102,102,60,60,0,0,0


PMDEMO DATA FOR CREATING STRINGS:

DATA FOR DAT$ IN LINE 80
9900 DATA 0,100,16,0,0,124,124,102,102,102,102,124,124,96,96,96,96,0,0,0
```

```
DATA FOR DAT$ IN LINE 81
9900 DATA 0,100,16,0,0,60,60,102,102,102,102,102,102,108,108,54,54,0,0,0

DATA FOR DAT$ IN LINE 82
9900 DATA 0,100,16,0,0,124,124,102,102,102,102,124,124,108,108,102,102,0,0,0

DATA FOR DAT$ IN LINE 83
9900 DATA 0,100,16,0,0,60,60,96,96,60,60,6,6,6,6,60,60,0,0,0

DATA FOR DAT$ IN LINE 84
9900 DATA 0,100,16,0,0,126,126,24,24,24,24,24,24,24,24,24,24,0,0,0

DATA FOR DAT$ IN LINE 85
9900 DATA 0,100,16,0,0,102,102,102,102,102,102,102,102,102,102,126,126,0,0,0

DATA FOR DAT$ IN LINE 86
9900 DATA 0,100,16,0,0,102,102,102,102,102,102,102,102,60,60,24,24,0,0,0

DATA FOR DAT$ IN LINE 87
9900 DATA 0,100,16,0,0,99,99,99,99,107,107,127,127,127,127,99,99,0,0,0

DATA FOR DAT$ IN LINE 88
9900 DATA 0,100,16,0,0,102,102,102,102,60,60,60,60,102,102,102,102,0,0,0

DATA FOR DAT$ IN LINE 89
9900 DATA 0,100,16,0,0,102,102,102,102,60,60,24,24,24,24,24,24,0,0,0

DATA FOR DAT$ IN LINE 90
9900 DATA 0,100,16,0,0,126,126,12,12,24,24,48,48,96,96,126,126,0,0,0

DATA FOR ERASE$ IN LINE 5070
9900 DATA 104,104,133,205,104,133,204,162,4,160,0,169,0,145,204,200,208,251,230,
205,202,208,246,96

DATA FOR PM$ IN LINE 5080
9900 DATA 104,104,133,204,104,133,203,104,104,133,205,104,133,206,104,162,0,160,
00,177,203,153,0,6,200,196,205,208,246,24
9901 DATA 189,0,6,101,206,133,208,232,189,0,6,133,207,232,189,0,6,133,209,232,16
0,0,189,0,6,145,207,232,200,198
9902 DATA 209,165,209,201,0,208,241,189,0,6,201,0,240,6,232,160,0,24,144,206,96
```

To Use PMDEMO

This program demonstrates two ways to use Player/Missile graphics other than forming rockets and bullets. It also provides a utility routine which accepts decimal values for player data and places the data into a string called DATS$. Careful study of this program plus a little experimentation will provide the tools required to use this unique feature of your computer.

PMDEMO uses players to create smooth movement of Graphics Mode 2-size text characters. The data for all capital letters from A thru Z are provided in string subroutines from line 65 thru line 90. To make Player into a letter, GOSUB to the ASCII value of the desired letter. For example, to use the A, GOSUB ASC("A") or GOSUB 65. For letter B, GOSUB ASC("B") or GOSUB 66, and so on.

Once the player letter is where it will stay on the screen, replace it with the same color text letter, then erase the player.

Two assembler routines are provided for drawing and erasing players. This could be done using Atari BASIC, but it's easier with a

little help from machine language. The erase routine may be found as line 19, and the draw routine in line 20 of the PMDEMO program. Change the name of the string to reflect the string name you use. The utility portion of this program will generate the string DATS$. The string used for the large capital letters is called DAT$.

The utility routine uses a player as a cursor. If you're tired of that big white block on the screen, create a player for this purpose. Your new cursor can be any color or shape you desire. I used a simple underline that curves upwards at each end.

I used this utility program to create the cursor and stored it in the string C$. To create the string, I had to provide decimal data to the program. To determine the decimal data, I had to convert binary values into their decimal equivalents. Confused?

Players are stored in the computer as bit patterns. There are 8 bits in a byte and there may be as many as 255 bytes in a player. To design the cursor, I used only two bytes. The first byte is a decimal 136 and the second is a 112.

The values of the 8 bits from left to right are 128, 64, 32, 16, 8, 4, 2, and 1. The first byte of the cursor player is a combination of two bits, 128 and 8. The second byte is a combination of 3 bits, 64, 32, and 16. 128 + 8 = 136 and 64 + 32 + 16 = 112. In other words, the two bytes would look like this:

Byte 1 = X O O O X O O O Byte 2 = O X X X O O O O

Use the BINCONV program supplied in this package to help you make your binary to decimal conversions.

Once you know the decimal values of your player, PMDEMO will create a string for you in the form of a BASIC line on the screen. Make changes where needed in line number or string name, position the cursor on the line press RETURN. That line will then be part of the PMDEMO program. List this line to disk or cassette for later use in your own programs.

When you create your player, it is important to make at least the first two bytes and the last two bytes zeros. (This was not done in creating the cursor because I did not need smooth movement vertically.) When you move a player vertically, zeros will erase the portion of the previous player which is otherwise left on the screen.

Vertical movement is achieved by changing the second position of

the player data, then using the assembler call to put it on the screen in its new position. The first byte of the string is the player number and the third byte is the number of bytes in the player string. The actual player data begins in the fourth position of the string.

Horizontal movement of a player is much easier. Once the player is on the screen, a simple POKE will change its horizontal position. The horizontal position of player 0 is poked into location 53248. POKE locations 53249, 53250, and 53251 for horizontal movement of players 1, 2, and 3.

Player color is also selected by using the POKE command. Color data for player 0 is poked into location 704. Color data for players 1, 2, and 3 is poked into location 705, 706, and 707. The color is determined by multiplying the color number by 16, then adding the luminance. For example, a POKE 704,0 would make player 0 black. POKE 705,15 would make player 1 white. Color 4 is red. Dark red or maroon would be a value or the color number 4 multiplied by 16 (64). Adding ten to this number would provide a light pink color.

Again, study of the program itself and a little experimentation is the best teacher.

```
0 REM PLAYER MISSILE DEMO/UTILITY (c) 1981 by Jerry White
1 REM NOTE REMARKS BEGINNING AT LINE 9500 FOR PROGRAM ENTRY
INFORMATION
2 REM IF YOU DID NOT PURCHASE THE DISKETTE VERSION OF THIS P
ACKAGE.
3 REM ************************************
9 DIM C$(6):REM PLAYER CURSOR
10 C$="♥( ◤p♥"
11 POKE 53277,0:POKE 53248,0:GOTO 9000
15 REM *** SUBROUTINES ***
19 A=USR(ADR(ERASE$),PMBASE)
20 A=USR(ADR(PM$),ADR(C$),LEN(C$),PMBASE):RETURN
25 A=USR(ADR(ERASE$),PMBASE):FOR ME=10 TO 0 STEP -0.5:SOUND
0,0,2,ME:NEXT ME:RETURN
30 A=USR(ADR(PM$),ADR(DAT$),LEN(DAT$),PMBASE):RETURN
50 C$(2,2)=CHR$(8*PEEK(84)+40):POKE 53248,PEEK(85)*4+48:GOSU
B 19
58 GET #1,GC:POSITION PEEK(85),PEEK(84):? CHR$(GC);
60 C$(2,2)=CHR$(8*PEEK(84)+40):POKE 53248,PEEK(85)*4+48:GOSU
B 19:RETURN
62 REM ************************************
63 REM STRINGS FOR ALPHABET OF PLAYERS
64 REM ************************************
65 DAT$="♥d♣♥♥▬▬<<ffff◖◖ff♥♥♥":RETURN
66 DAT$="♥d♣♥♥▌▌ff▌▌ffff▌▌♥♥♥":RETURN
67 DAT$="♥d♣♥♥<<ff♥♥♥♥ff<<♥♥♥":RETURN
68 DAT$="♥d♣♥♥××llffff11××♥♥♥":RETURN
69 DAT$="♥d♣♥♥◖◖♥♥▌▌♥♥♥♥◖◖♥♥♥":RETURN
70 DAT$="♥d♣♥♥◖◖♥♥▌▌♥♥♥♥♥♥♥♥":RETURN
71 DAT$="♥d♣♥♥>>♥♥♥♥nnff>>♥♥♥":RETURN
72 DAT$="♥d♣♥♥ffff◖◖ffffff♥♥♥":RETURN
73 DAT$="♥d♣♥♥◖◖▬▬▬▬▬▬◖◖♥♥♥":RETURN
74 DAT$="♥d♣♥♥////////ff<<♥♥♥":RETURN
75 DAT$="♥d♣♥♥ff11××××11ff♥♥♥":RETURN
76 DAT$="♥d♣♥♥♥♥♥♥♥♥♥♥◖◖♥♥♥":RETURN
77 DAT$="♥d♣♥♥ccww▶▶kkcccc♥♥♥":RETURN
```

```
78 DAT$="♥d♣♥♥ffvv◀◀◀◀nnff♥♥♥":RETURN
79 DAT$="♥d♣♥♥<<ffffffff<<♥♥♥":RETURN
80 DAT$="♥d♣♥♥||ffff||♦♦♦♦♥♥♥":RETURN
81 DAT$="♥d♣♥♥<<ffffff1166♥♥♥":RETURN
82 DAT$="♥d♣♥♥||ffff||11ff♥♥♥":RETURN
83 DAT$="♥d♣♥♥<<♠♠<<////<<♥♥♥":RETURN
84 DAT$="♥d♣♥♥◀▬▬▬▬▬▬▬▬♥♥♥":RETURN
85 DAT$="♥d♣♥♥fffffffff◀◀♥♥♥":RETURN
86 DAT$="♥d♣♥♥fffffff<<▬▬♥♥♥":RETURN
87 DAT$="♥d♣♥♥cccckk▶▶▶▶oo♥♥♥":RETURN
88 DAT$="♥d♣♥♥ffff<<<<ffff♥♥♥":RETURN
89 DAT$="♥d♣♥♥ffff<<▬▬▬▬♥♥♥":RETURN
90 DAT$="♥d♣♥♥◀◀""▬▬00♠♠◀◀♥♥♥":RETURN
100 POSITION 0,0:FOR ME=1 TO 10:? #6;"                    ":NE
XT ME:POKE 708,31
110 POSITION 0,2:? #6;"   player missile":? #6:? #6;"
emo by":GOSUB 25
120 GOSUB ASC("J"):DAT$(2,2)=CHR$(128):GOSUB 30:FOR X=40 TO
80:POKE 53248,X:NEXT X:POSITION 4,6:? #6;"J"
130 GOSUB 25:GOSUB ASC("E"):DAT$(2,2)=CHR$(128):POKE 53248,2
20:GOSUB 30:FOR X=220 TO 88 STEP -1:POKE 53248,X:NEXT X
140 POSITION 5,6:? #6;"E":GOSUB 25:GOSUB ASC("R"):DAT$(2,2)=
CHR$(128):POKE 53248,220:GOSUB 30
150 FOR X=220 TO 96 STEP -1:POKE 53248,X:NEXT X:POSITION 6,6
:? #6;"R":GOSUB 25:GOSUB 30
160 FOR X=96 TO 104:POKE 53248,X:NEXT X:POSITION 7,6:? #6;"R
":GOSUB 25:GOSUB ASC("Y"):DAT$(2,2)=CHR$(128)
170 POKE 53248,220:GOSUB 30:FOR X=220 TO 112 STEP -1:POKE 53
248,X:NEXT X:POSITION 8,6:? #6;"Y":GOSUB 25
180 POKE 710,15:POSITION 10,6:? #6;"■":GOSUB 25:POSITION 11,
6:? #6;"■":GOSUB 25
190 POSITION 12,6:? #6;"■":GOSUB 25:POSITION 13,6:? #6;"■":G
OSUB 25:POKE 704,15
200 GOSUB ASC("E"):POKE 53248,88:DAT$(2,2)=CHR$(128):GOSUB 3
0:FOR ME=128 TO 144 STEP 2:DAT$(2,2)=CHR$(ME):GOSUB 30
210 NEXT ME:FOR X=88 TO 160:POKE 53248,X:NEXT X:FOR ME=144 T
O 128 STEP -2:DAT$(2,2)=CHR$(ME):GOSUB 30:NEXT ME
220 POSITION 14,6:? #6;"■":GOSUB 25
290 FOR ME=1 TO 200:NEXT ME:FOR ME=250 TO 10 STEP -10:SOUND
0,ME,10,8:NEXT ME:SOUND 0,0,0,0:POKE 53248,0
300 POSITION 0,0:FOR ME=1 TO 10:? #6;"                    ":NE
XT ME:POKE 704,31
310 POSITION 7,2:? #6;"PLAYER":POSITION 7,4:? #6;"MISSILE":G
OSUB 25
320 GOSUB ASC("T"):DAT$(2,2)=CHR$(127):GOSUB 30:FOR ME=40 TO
 104:POKE 53248,ME:NEXT ME
325 POSITION 7,6:? #6;"T":DAT$="":GOSUB 25
330 GOSUB ASC("E"):POKE 53248,152:DAT$(2,2)=CHR$(100):FOR ME
=100 TO 128 STEP 2:DAT$(2,2)=CHR$(ME):GOSUB 30:NEXT ME
333 FOR ME=152 TO 112 STEP -1:POKE 53248,ME:NEXT ME
335 POSITION 8,6:? #6;"E":GOSUB 25
340 GOSUB ASC("X"):DAT$(2,2)=CHR$(128):GOSUB 30:FOR ME=220 T
O 120 STEP -1:POKE 53248,ME:NEXT ME
345 POSITION 9,6:? #6;"X":GOSUB 25
350 GOSUB ASC("T"):POKE 53248,104:FOR ME=127 TO 144 STEP 2:D
AT$(2,2)=CHR$(ME):GOSUB 30:NEXT ME
355 FOR ME=104 TO 128:POKE 53248,ME:NEXT ME
360 FOR ME=144 TO 128 STEP -2:DAT$(2,2)=CHR$(ME):GOSUB 30:NE
XT ME
365 POSITION 10,6:? #6;"T":GOSUB 25
380 FOR X=1 TO 3:SETCOLOR 0,X,2:FOR M=10 TO 250 STEP 10:SOUN
D 0,M,10,X*3:SOUND 1,M,0,X*3:POKE 712,M:NEXT M:NEXT X
390 SOUND 0,0,0,0:SOUND 1,0,0,0:OP=2:POKE 53277,0:POKE 712,0
:FOR X=1 TO 255:POKE 708,X:NEXT X
400 GOSUB 25:GRAPHICS 0:POKE 710,240:POKE 709,252:GOTO 5010
3999 REM
4000 REM *** SETUP P/M GRAPHICS ***
4001 REM
5000 POKE 710,0:DIM DAT$(255),PM$(81),ERASE$(24)
5010 PMADD=PEEK(106)-16:REM PM AREA
5020 POKE 559,62:REM SINGLE LINE RES.
5030 POKE 54279,PMADD
5040 POKE 704,31:REM PLAYER 0 COLOR
5060 PMBASE=(PMADD+4)*256:REM PM0 BASE
```

```
5070 ERASE$="hh[illegible]"
5080 PM$="hh[illegible]"
5090 POKE 53277,3:REM ENABLE PM
5100 IF OP=1 THEN 100
5140 REM
5150 REM *** CREATE P/M STRINGS ***
5160 REM
5200 ? CHR$(125):POKE 709,12:CLOSE #1:OPEN #1,4,0,"K:":POKE
752,1:?
5205 DIM DATS$(255),YN$(1):K=0:? "          CREATE PLAYER STRI
NG:":?
5220 ? :? "ENTER PLAYER MISSILE NUMBER (0 TO 3)";
5225 GOSUB 50:GC=GC-48:IF GC<0 OR GC>3 THEN 5220
5227 K=K+1:DATS$(K,K)=CHR$(GC)
5230 GOSUB 7000:? :? :? "ENTER VERTICAL POSITION (40-230)";
5232 GOSUB 50:GC=GC-48:IF GC<1 OR GC>9 THEN 5230
5233 V1=GC:GOSUB 50:GC=GC-48:IF GC<0 OR GC>9 THEN 5230
5234 V2=GC:GOSUB 50:IF GC=155 THEN VP=V1*10+V2:GOTO 5239
5235 GC=GC-48:IF GC<0 OR GC>9 THEN 5230
5237 VP=V1*100+V2*10+GC:IF VP<40 OR VP>230 THEN 5230
5239 K=K+1:DATS$(K,K)=CHR$(VP)
5240 GOSUB 7000:? :? :? "HOW MANY BYTES (1-255)";
5241 GOSUB 50:GC=GC-48:IF GC<1 OR GC>9 THEN 5240
5242 B1=GC:GOSUB 50:IF GC=155 THEN NB=B1:GOTO 5250
5243 GC=GC-48:B2=GC:GOSUB 50:IF GC=155 THEN NB=B1*10+B2:GOTO
 5250
5244 GC=GC-48:IF GC<0 OR GC>9 THEN 5240
5247 NB=B1*100+B2*10+GC:IF NB<1 OR NB>255 THEN 5240
5250 K=K+1:DATS$(K,K)=CHR$(NB)
5260 ? CHR$(125):? :? ,"  ENTER ";NB;" BYTES:":FOR I=1 TO NB


5270 GOSUB 7000:? :? ,"   BYTE ";I;"= ";
5271 GOSUB 50:GC=GC-48:IF GC<0 OR GC>9 THEN 5270
5272 B1=GC:GOSUB 50:IF GC=155 THEN BY=B1:GOTO 5279
5273 GC=GC-48:IF GC<0 OR GC>9 THEN 5270
5274 B2=GC:GOSUB 50:IF GC=155 THEN BY=B1*10+B2:GOTO 5279
5275 GC=GC-48:IF GC<0 OR GC>9 THEN 5270
5277 BY=B1*100+B2*10+GC:IF NB<1 OR BY>255 THEN 5270
5279 K=K+1:DATS$(K,K)=CHR$(BY):NEXT I:K=K+1
5280 ? CHR$(125):? :? :? "ANOTHER DRWG IN THE SAME DISPLAY (
Y/N)";
5282 GOSUB 50:IF GC=89 THEN DAT$(K,K)=CHR$(128):? CHR$(125):
GOTO 5220
5284 IF GC<>78 THEN 5280
5300 DATS$(K,K)=CHR$(0)
5370 POKE 53248,0:POKE 766,0:? CHR$(125):? :? "[illegible]
 STRING IN YOUR PROGRAM":?
5380 ? :? "(1) FIX  LINE # AND STRING NAME"
5400 ? "(2) MOVE THE CURSOR OVER THE LINE":? "     AND PRESS
[illegible]":? :?
5410 ? "[illegible] IF DATS$ CONTAINS A (";CHR$(34);"),":? "
CHANGE IT TO A BLANK":? "       AND ADD THIS NEW LINE:"
5420 ? "       NNN DATS$(X,X)=CHR$(34)":? :? "WHERE X IS THE
POSITION OF THE BLANK":?
5510 L=INT(K/90):POKE 766,1
5520 IF L=0 THEN ? "996 DATS$=";CHR$(34);DATS$;CHR$(34):GOTO
 6000
5530 IF L=1 THEN ? "996 DATS$(1,90)=";CHR$(34);DATS$(1,90);C
HR$(34):GOTO 5560
5540 ? "996 DATS$(1,90)=";CHR$(34);DATS$(1,90);CHR$(34):? "9
97 DATS$(91,180)=";CHR$(34);DATS$(91,180);CHR$(34)
5550 ? "998 DATS$(181,";K;")=";CHR$(34);DATS$(181,K);CHR$(34
):GOTO 6000
5560 ? "997 DATS$(91,";K;")=";CHR$(34);DATS$(91,K);CHR$(34)
6000 GOSUB 25:POKE 752,0:POKE 53277,0:POKE 766,0:CLOSE #1:EN
D
7000 GOSUB 25:POKE 764,255:RETURN
9000 GRAPHICS 0:POKE 710,0:POKE 752,1:POKE 201,8:? :? :? ,"
[illegible] DEMO"
9020 ? :? "   This program demonstrates  the use":? :? "of P
layer Missile Graphics and can be"
9040 ? :? "used  to create strings  from decimal":? :? "play
er data."
```

```
9060 ? :? "   Special thanks to Fernando Herrera":? :? "for
his contribution of the assembler"
9080 ? :? "routines  used to put players  on the":? :? "sore
en and erase."
9100 ? :? "          Press [START] to begin.":POKE 709,12
9120 IF PEEK(53279)<>6 THEN POKE 755,1:POKE 755,2:GOTO 9120
9140 POKE 755,2:OP=1:GRAPHICS 18:GOTO 5000
9500 REM **********************
9510 REM CONVERT PROGRAM DATA
9515 REM * TO STRING FORMAT *
9520 REM **********************
9530 REM ENTER EACH LINE OF DATA AS SHOWN IN THE DOCUMENTATI
ON LABELED "PMDEMO DATA FOR CREATING STRINGS.
9540 REM
9550 REM IN EACH CASE, THE DATA LINE NUMBER WILL BE 9900 (NO
TE: DATA FOR PM$ WILL USE LINES 9900, 9901, AND 9902).
9560 REM
9570 REM AFTER CREATING DATA FOR EACH STRING, TYPE GOTO 9700
 AND PRESS RETURN.
9580 REM
9590 REM TYPE THE LINE NUMBER FOR THE STRING (NOT 9900).  EX
AMPLE: THE FIRST LINE NUMBER IS 10.
9600 REM
9610 REM TYPE THE STRING NAME INCLUDING THE $.  EXAMPLE: THE
 FIRST STRING NAME IS C$
9620 REM
9640 REM REPEAT THIS PROCEDURE FOR EACH STRING TO BE ENTERED
 (TOTAL=29).
9650 REM
9660 REM EXAMPLE FOR C$ IN LINE 10
9670 REM
9680 REM ********************************
9700 REM STRING DIMENSIONS FOR PROGRAM           DATA TO STRI
NG CONVERSIONS
9701 REM ********************************
9710 TRAP 9720:DIM NAME$(6),LINE$(4),C$(6),DAT$(100),ERASE$(
24),PM$(81)
9720 ? :? "ENTER LINE NUMBER";:INPUT LINE$
9730 TRAP 9720:LINE=VAL(LINE$):IF LINE<10 OR LINE>5080 THEN
9720
9740 ? :? "ENTER STRING NAME";:INPUT NAME$
9750 LN=LEN(NAME$):TRAP 9740:IF NAME$(LN,LN)<>"$" THEN 9740
9800 RESTORE 9900:POKE 766,1:? :? LINE$;" ";NAME$;"=";CHR$(3
4);
9820 TRAP 9950:READ NUMBER:? CHR$(NUMBER);:GOTO 9820
9900 DATA 0,40,2,136,112,0
9950 ? CHR$(34);:IF LINE>64 AND LINE<91 THEN ? ":RETURN";
9955 POKE 766,0:? CHR$(28);:? CHR$(28);:POKE 764,12
9960 CLR :END
10000 REM
10010 REM SEE DOCUMENTATION FOR INFORMATION ON CONVERTING DA
TA TO STRINGS
10020 REM
10030 REM THE FOLLOWING IS A LIST OF LINE NUMBERS AND PROGRA
M TEXT WHERE
10040 REM TEXT MUST BE ENTERED IN INVERSE VIDEO.
10050 REM
10060 REM LINE#   INVERSE VIDEO TEXT
10070 REM =====   ==================

10080 REM 110     demo by
10090 REM 180     W  H
10100 REM 190     I  T
10110 REM 220     E
10120 REM 5370    TO ENTER THE STRING IN YOUR PROGRAM:
10130 REM 5400    RETURN
10140 REM 5410    NOTE:
10150 REM 9000    PLAYER MISSILE DEMO
10160 REM 9100    START
```

Chapter 18
Disk-Based Inventory (INVENT)

This program creates a 100-record inventory file and permits the user to update it by using a method called random access. Random access allows immediate access to any given record in a file without reading each record in the file each time you request a record.

The initial program display consists of three options. First we must create a data file. Type the number 1. This will send the program to line 100. Here we create a 100-record data file. Each record will contain a record number, an item count, and an item description field. The fields are separated by commas. As the file is created, it will be displayed on the screen. After record 100 is written, the file is closed and you will be returned to the three original options.

Now type the number 2. In order to use random access updating, we must know exactly where each record begins on the disk. Before updating, the program will create an index using arrays in memory. Once this is done, we can instantly find any record using the POINT instruction. First, we must NOTE the current disk location and store sectors and bytes in our arrays. We only have to do this once; then we can inspect or change as many records as needed.

The index is created using the routine starting at line 300 and ending at 420. The beginning of the random access routine is at line 500.

At line 310, we check a flag to see if an index has already been created. If so, we do not have to repeat this procedure. To create the index, we read the data file. Before reading each record, we NOTE the sector and byte position and store it in our SEC and BYT arrays.

Once the array is completed, we are ready to display or update any record in the data file using the POINT instruction to locate the record. Let's start by displaying record 50. Type D and press RETURN. Then type 50 and press RETURN. At line 760, we POINT to the sector and byte of record 50. At line 780, we INPUT that record, clear the screen, and display record 50 on the screen.

Remember the number of items in this record! Press any key and you will be returned to the option routine at line 5000. This time, type the number 2, then U and press RETURN. Type 50 and press RETURN.

Record 50 will again be displayed, but now we have 3 new options.

Let's take them in order. Type 1 and then press RETURN. To update the quantity, we can add to it by typing the number of items to add, or subtract from it by typing a negative number. Remember that our quantity field is only 3 positions, so don't increase it to more than 999 items.

When we read the record from the data file, we stored it in a string REC$. The quantity field has just been updated only in the string. It will be updated only on the disk when we choose option 3 to EXIT. Before we exit, let's change the description field. Type 2 and press RETURN. Choose a new description and type it in. Now type 3, and the record will be updated on the disk.

To be sure that the record has been changed properly, you can choose the display/update option to display record 50. By now you see the advantage of random access updating again: You don't have to read the first 49 records to get to record 50. Using this random access method, we can locate any record in a disk data file instantly.

NOTE: The ATARI BASIC ROUTINES diskette contains DOS 20S. You must boot your system using DOS 20S prior to using the INVENT program.

```
20 REM INVENTORY TUTORIAL PROGRAM TO DEMONSTRATE RANDOM ACCE
SS UPDATING
30 REM *** BY JERRY WHITE ***
40 REM ORIGINALLY PUBLISHED IN A.N.A.L.O.G. 400/800 MAGAZINE

50 DIM SEC(100),BYT(100),REC$(30),DES$(30),CHOICE$(1):CI=0:G
OTO 5000
100 REM *** CREATE INITIAL DATA FILE ***
110 FOR BLANK=1 TO 30:REC$(BLANK,BLANK)=" ":NEXT BLANK
120 CLOSE #1:OPEN #1,8,0,"D:DATAFILE"
130 REC$(4,4)=",":REC$(8,30)=",ITEM DESCRIPTION FIELD"
140 FOR RECORD=1 TO 100
160 IF RECORD<10 THEN REC$(1,2)="00":REC$(3,3)=STR$(RECORD):
GOTO 220
180 IF RECORD<100 THEN REC$(1,1)="0":REC$(2,3)=STR$(RECORD):
GOTO 220
200 REC$(1,3)=STR$(RECORD)
220 REC$(5,7)=STR$(RND(0)*100+100)
240 PRINT #1;REC$:? :? "RECORD ";RECORD:? REC$:NEXT RECORD
260 CLOSE #1:GOTO 5000
300 REM *** CREATE INDEX ***
310 IF CI=1 THEN RECORD=101:GOTO 500
320 TRAP 2000:CLOSE #2:OPEN #2,4,0,"D:DATAFILE":TRAP 40000
360 FOR ARRAY=1 TO 100:NOTE #2,SECTOR,BYTE
380 ? :? "RECORD ";ARRAY;"  SECTOR ";SECTOR;"  BYTE ";BYTE
400 SEC(ARRAY)=SECTOR:BYT(ARRAY)=BYTE:INPUT #2,REC$:NEXT ARR
AY
420 CLOSE #2:CLOSE #3:CI=1
500 REM *** RANDOM ACCESS DATAFILE ***
520 CLOSE #4:OPEN #4,12,0,"D:DATAFILE"
540 ? CHR$(125):? :? "TYPE D TO DISPLAY A RECORD":? :? "TYPE
 U TO UPDATE A RECORD";
560 INPUT CHOICE$:IF CHOICE$="D" THEN 700
580 IF CHOICE$="U" THEN 900
```

```
600 ? CHR$(253):GOTO 540
700 ? :? "TYPE RECORD NUMBER TO DISPLAY";:TRAP 700:INPUT RN:
TRAP 40000
720 IF RN<ARRAY AND RN>0 AND RN=INT(RN) THEN 760
740 ? CHR$(253):? "INVALID RECORD NUMBER":GOTO 700
760 POINT #4,SEC(RN),BYT(RN)
780 INPUT #4,REC$:? CHR$(125):? :? "RECORD ";RN:? :? REC$
800 ? :? "PRESS ANY KEY FOR OPTIONS:":POKE 764,255:CLOSE #4
820 IF PEEK(764)<>255 OR PEEK(53279)<>7 THEN POKE 764,255:GO
TO 5000
840 GOTO 820
900 ? :? "TYPE RECORD NUMBER TO BE UPDATED";:TRAP 900:INPUT
RN:TRAP 40000
920 IF RN<ARRAY AND RN>0 AND RN=INT(RN) THEN 960
940 ? CHR$(253):? "INVALID RECORD NUMBER":GOTO 900
960 POINT #4,SEC(RN),BYT(RN)
980 INPUT #4,REC$:? CHR$(125)
1000 ? :? "RECORD ";RN:? :? REC$
1010 ? :? "TYPE 1 TO UPDATE QUANTITY":? "TYPE 2 TO CHANGE DE
SCRIPTION":? "TYPE 3 TO EXIT"
1020 TRAP 1000:INPUT CHOICE:TRAP 40000
1040 IF CHOICE<1 OR CHOICE>3 OR CHOICE<>INT(CHOICE) THEN ? C
HR$(253):GOTO 1000
1060 ON CHOICE GOTO 1100,1300,1080
1080 POINT #4,SEC(RN),BYT(RN):PRINT #4;REC$:CLOSE #4:GOTO 50
00
1100 ? :? "TYPE POSITIVE NUMBER TO INCREASE ITEMS":? "TYPE N
EGATIVE NUMBER TO DECREASE ITEMS"
1140 TRAP 1100:INPUT NUMBER:TRAP 40000
1160 ITEMS=VAL(REC$(5,7)):ITEMS=ITEMS+NUMBER
1180 IF ITEMS>999 THEN ? CHR$(253):? "ITEMS CANNOT EXCEED 99
9":GOTO 1100
1200 IF ITEMS<0 THEN ? CHR$(253):? "ITEMS CANNOT BE A LESS T
HAN ZERO":GOTO 1120
1220 IF ITEMS<10 THEN REC$(5,6)="00":REC$(7,7)=STR$(ITEMS):G
OTO 1000
1240 IF ITEMS<100 THEN REC$(5,5)="0":REC$(6,7)=STR$(ITEMS):G
OTO 1000
1260 REC$(5,7)=STR$(ITEMS):GOTO 1000
1300 ? CHR$(125):? :? "RECORD ";RN:? :? REC$
1320 ? :? "TYPE NEW DESCRIPTION UP TO 22 POSITIONS"
1340 INPUT DES$:LD=LEN(DES$)
1360 IF LD>22 THEN ? CHR$(253):? "FIELD TOO LONG, EXTRA IGNO
RED"
1380 IF LD=22 THEN 1420
1400 FOR BLANK=LD TO 22:DES$(LEN(DES$)+1)=" ":NEXT BLANK
1420 REC$(9,30)=DES$:GOTO 1000
2000 ? CHR$(253):? :? "DATAFILE NOT ON DISK:TRAP 40000"
2010 FOR WAIT=1 TO 500:NEXT WAIT:GOTO 5000
5000 REM *** INITIAL DISPLAY OF OPTIONS ***
5010 GRAPHICS 18:? #6:? #6;" INVENTORY OPTIONS:":? #6:? #6;"
  1= CREATE FILE"
5020 ? #6:? #6;"  2= DISPLAY/UPDATE":? #6:? #6;"  3= END PRO
GRAM"
5040 CLOSE #5:OPEN #5,4,0,"K:":GET #5,GC:CLOSE #5:GC=GC-48
5060 IF GC<1 OR GC>3 THEN 5000
5080 GRAPHICS 0:POKE 82,1:SETCOLOR 2,0,0:ON GC GOTO 100,300,
6000
6000 GRAPHICS 0:POKE 82,2:? :? "LOADING MENU";:RUN "D:MENU"
```

Chapter 19
Delete BASIC Lines (DELETE.LST)

Before you may use DELETE.LST, the program from which you will delete lines of BASIC code must be in memory. Your program cannot use any of the same line numbers as DELETE.LST (lines numbered 31999 or higher).

Enter D:DELETE.LST. In the immediate mode, enter the BASIC command **GOTO 32000** and press **RETURN**. When you are asked, enter the lowest and highest line numbers to be deleted. All line numbers in the range you specified will be deleted at the rate of approximately 25 per second. During this process, your screen will be blank.

When this process is finished, the screen will display the following message:

"SAVE NEW PROGRAM WITH LIST COMMAND."

If you have other lines to delete, repeat the process as often as needed. When all of the desired lines have been deleted, LIST your new program onto disk with the following command:

LIST"D:FILENAME",0,31999

Be sure to give the NEW command before reentering the file from disk. Test your new program, and LIST or SAVE it on cassette or diskette.

NOTE: DELETE.LST will not delete any line number above 31999. Many utility programs written in BASIC will use the line numbers above 29999. It is therefore good practice not to use these line numbers.

```
32000 GRAPHICS 0:POKE 82,2:TRAP 32000:? "STARTING LINE TO DE
LETE";:INPUT SD:IF SD>31999 THEN 32000
32002 TRAP 32002:? "LAST LINE TO DELETE";:INPUT ED:IF ED<SD
OR ED>31999 THEN 32000
32004 DEL=ED-SD:SC=INT(DEL/20):TRAP 32016
32006 FOR ME=0 TO SC:GRAPHICS 0:POKE 559,0:POSITION 2,3:FOR
D=0 TO 19:LINE=ME*20+SD+D
32008 IF LINE>ED THEN POP :GOTO 32012
32010 ? LINE:NEXT D
32012 ? "CONT":POSITION 2,0:POKE 842,13:STOP
32014 POKE 842,12:NEXT ME
32016 TRAP 40000:GRAPHICS 0:POKE 559,34:? "SAVE NEW PROGRAM
WITH LIST COMMAND.":END
32020 REM DELETE.LST (c) 1981 by Jerry White - Version 1.0 1
0/13/81
```

Chapter 20
(A.LST, B.LST, D.LST, E.LST, F.LST, G.LST, I.LST)

Compare Atari's DOS 20S to the old DOS 1. The new DOS has both advantages and disadvantages. Bugs in the first version have been corrected, and the latest version requires less RAM because its utility package is loaded only when the DOS Command is issued. A big disadvantage will be noticed at this time.

The utilities will be loaded into your program area unless you have a MEM.SAV file. If you use the MEM.SAV option, it will take about 30 seconds to delete the prior MEM.SAV file, save the current program area, and load the DOS utilities (DUP.SYS).

Many of these utilities can be accessed much faster by using small BASIC routines. These routines can be stored on your work diskettes and called using the ENTER command. Each of these BASIC utilities begins at line number 32000. Entering them will not disturb your program in RAM as long as it does not use the same line numbers.

These routines must be listed onto disk; do not use the SAVE command. We will use single-letter file names which correspond to the similar DOS options except for the letter B. For example, we use the DOS option A for a disk directory. The A routine will be listed onto the diskette with the command LIST"D:A". To use the A routine, type **ENTER"D:A"** and press **RETURN**. Then type **GOTO 32000** and press **RETURN**.

Each routine has been kept as short as possible so that it can be used quickly, will use very little memory, and will require very little room on your disk. User input must be entered properly since there is no error trapping. (Error trapping can be added if you decide it's necessary.) If you do make a mistake, it is simple enough to start over by typing **GOTO 32000**.

The B routine is used as a reminder menu of the BASIC utility options available. The D, E, F, G, and I routines do just what the corresponding DOS options do. The user input to these routines via keyboard is slightly different.

When using the D, F, and G options, you must specify the disk drive to be used. For example, to delete a file called "test" from drive 1, you

would type **D:TEST**. If that file is to be deleted from drive 2, the command would be **D2:TEST**. LOCK and UNLOCK (F and G) work the same way.

When using the RENAME and FORMAT DISK (E and I) routines, drive 1 is assumed. To rename that TEST file to TEST10, you would simply type **TEST,TEST10**.

Study these little programs. They demonstrate only a few of the many uses of the X10 command. The A routine will show you how to access the disk directory by opening the directory file with the operand 6.

Using these BASIC programs instead of DOS will save you time and eliminate the possibility of losing a program in memory by calling DOS. You can call in one of these routines after another since the ENTER command causes identical line numbers to overlay new over old. You need not delete one before using another.

These BASIC utilities will come in very handy. You'll only wish you had learned them sooner!

```
32000 CLR :GRAPHICS 0:DIM R$(20):OPEN #1,6,0,"D:*.*":? :? "
 DISK DIRECTORY":?
32010 TRAP 32030:INPUT #1,R$:IF R$(5,16)="FREE SECTORS" THEN
 32030
32020 ? R$:GOTO 32010
32030 ? R$:CLOSE #1:END

32000 GRAPHICS 0:? :? ,"BASIC DISK UTILITIES":? :? ,"A= DISK
 DIRECTORY":? :? ,"B= LIST OF UTILITIES"
32010 ? :? ,"D= DELETE FILE":? :? ,"E= RENAME FILE":? :? ,"F
= LOCK FILE":? :? ,"G= UNLOCK FILE"
32020 ? :? ,"I= FORMAT DISK":? :? ,"ENTER PROGRAM ?.LST":? :
? ,"TYPE GOTO 32000 RETURN":END

32000 CLR :GRAPHICS 0:? :? ,"DELETE FILE":DIM F$(20)
32010 ? :? "ENTER D#:FILE";:INPUT F$:XIO 33,#1,0,0,F$:END

32000 CLR :GRAPHICS 0:? :? "LOCK FILE":DIM F$(20)
32010 ? :? "ENTER D#:FILE";:INPUT F$:XIO 35,#1,0,0,F$:END

32000 CLR :GRAPHICS 0:? :? "UNLOCK FILE":DIM F$(20)
32010 ? :? "ENTER D#:FILE";:INPUT F$:XIO 36,#1,0,0,F$:END

32000 GRAPHICS 0:? :? "FORMAT DISK":? :? "INSERT DISK TO FOR
MAT IN DRIVE 1":? :? "PRESS START WHEN READY"
32010 IF PEEK(53279)<>6 THEN 32010
32020 ? :? "PRESS OPTION KEY TO FORMAT DISK"
32030 IF PEEK(53279)<>3 THEN 32030
32040 XIO 254,#1,0,0,"D1:":END

32000 CLR :GRAPHICS 0:DIM FIN$(20),FOUT$(20),X$(40):? :? "RE
NAME FILE"
32010 ? :? "ENTER OLD NAME";:INPUT FIN$:? :? "ENTER NEW NAME
";:INPUT FOUT$
32020 X$="D:":X$(LEN(X$)+1)=FIN$:X$(LEN(X$)+1)=",":X$(LEN(X$
)+1)=FOUT$:XIO 32,#1,0,0,X$:END
32030 ? R$:CLOSE #1:END
```

Chapter 21
CONSERVING MEMORY

Here are a few memory-saving techniques in Atari BASIC. Always define your most often used numeric constants as variables. I have found that you can save hundreds and even thousands of bytes this way.

For example:

```
10 ? 1+1+1+1+1+1+1+1+1+1
```
uses 85 bytes.

```
10 X=1:? X+X+X+X+X+X+X+X+X+X
```
uses 54 bytes.

Define variables by reading data:

```
10 A=1:B=2:C=3:D=4:E=5:F=6:G=7:H=8:I=9:J=10
```
uses 232 bytes.

```
10 READ A,B,C,D,E,F,G,H,I,J: DATA 1,2,3,4,5,6,7,8,9,10
```
uses 147 bytes.

Use strings instead of arrays whenever possible. Each element of an array costs an additional 6 bytes. Each position within a string costs only 1 byte.

Remove REM statements and keep at least two versions of your program. The remarks are important, so keep a backup version which includes remarks, but removing REMs from your program will save memory and increase efficiency.

Keep the number of variables used to a minimum. You will save 8 bytes for each variable plus one byte for each character in its name.

Concatenate lines whenever possible to increase program speed and save three bytes per line.

Use POKE commands instead of SETCOLOR and Cursor Control Characters instead of POSITION commands to save lots of memory.

Use strings for storing data that must be displayed or printed repeatedly. For example, make a Clear Screen and Display Heading subroutine instead of repeating lines of code.

Chapter 22
PROGRAM SPEED

There are many factors that affect the speed of Atari BASIC. The Graphics Mode being used is one of them.

As a test, we will use a FOR NEXT loop to count to 1000. We will time the loop in Graphics Modes 0 thru 8 by jiffies or 1/60ths of a second. Consider the following one liner where M = MODE:

```
10 GRAPHICS M:POKE 19,0:POKE 20,0:FOR X = 1 TO 1000:NEXT X:?
PEEK(19),PEEK(20)
```

MODE	JIFFIES
0	123
1	103
2	102
3	89
4	90
5	92
6	95
7	101
8	120

If we disable DMA before the loop with a POKE 559,0 and reenable it with a POKE 559,34 after the loop, the result is 84 jiffies no matter which mode is used.

This experiment reveals information that's strange but true. Graphics Mode 8 is faster than Graphics Mode 0. Graphics Mode 7 is slightly faster than Modes 1 and 2. This is due to the fact that text modes must look at the character set for every position on the screen.

If you want to see a much greater difference in the time it takes for a FOR NEXT loop to execute, try the same line at the beginning of a program, then try it at the end of the same program. The bigger the program, the bigger the difference. If the program is rather large, the loop at the end might take 10 times as long as the loop at the beginning.

For this reason, frequently used subroutines should always be placed near the beginning of a program. Introductory displays, DIM statements, and other one time routines should be put near the end.

Chapter 23
USING MEMORY LOCATIONS TO PEEK AND POKE

POKE 16,64:POKE 53744,112 to disable the BREAK key.

P18 = PEEK(18):P19 = PEEK(19):P20 = PEEK(20) to read the clock. (See the TIMER program)

POKE 65,0 to shut off I/O noises like disk beeps.

POKE 77,0 to reset attract mode screen color changes.

POKE 82,X sets left screen margin to X.

POKE 83,X sets right screen margin to X.

ROW = PEEK(84) returns the current row position of the cursor in a graphics window.

COL = PEEK(85) + PEEK(86)*256 returns the current column position of the cursor in a graphics window.

POKE 87,0 fools the O.S. into thinking you are using mode 0 without changing modes.

SA = PEEK(88) + PEEK(89)*256 returns the low address of screen memory or the upper leftmost position on the screen.

OROW = PEEK(90) returns the old or previous cursor row in a graphics window.

OCOL = PEEK(91) + PEEK(92)*256 returns the old or previous cursor column in a graphics window.
OCHR = PEEK(93) returns the character under the graphics window cursor.

NR = PEEK(96) returns the row of the DRAWTO destination.

NC = PEEK(97) + PEEK(98)*256 returns the column of the DRAWTO destination.

TOR = PEEK(106) returns the top of RAM in pages. NOTE: Each page is 256 bytes.

LOM = PEEK(128) + PEEK(129)*256 returns BASIC's low memory pointer.

VN = PEEK(130) + PEEK(131)*256 returns the starting address of the variable name table.

EVN = PEEK(132) + PEEK(133)*256 returns the ending address of the variable name table.

VVT = PEEK(134) + PEEK(135)**256 returns the address of the variable value table.

STA = PEEK(136) + PEEK(137)*256 returns the statement table address.

CSA = PEEK(138) + PEEK(139)*256 returns the address of the current statement.

SAT = PEEK(140) + PEEK(141)*256 returns the string array table address.

STA = PEEK(142) + PEEK(143)*256 returns the run time stack address.

TOM = PEEK(144) + PEEK(145)*256 returns BASIC's top of memory pointer.

ERL = PEEK(186) + PEEK(187)*256 returns the line number where an error or stop occurred.

ERN = PEEK(195) returns the error number.

POKE 201,X sets the print tab width to X.

VUS = PEEK(212) + PEEK(213)*256 returns the value from a USR function.

RD = PEEK(251) returns 0 for radians or 6 for degrees.

JIF = PEEK(540) returns the value of a counter that counts back to zero in 60ths of a second.

POKE 559,0 disables the screen for faster calculations, POKE 559,34 restores the screen.

DLA = PEEK(560) + PEEK(561)*256 returns the address of the display list.

PHV = PEEK(564) returns the horizontal value of the light pen.

PVV = PEEK(565) returns the vertical value of the light pen.

POKE 580,1 causes the system to reboot when SYSTEM RESET is pressed.

POKE 623,X sets the player/playfield priority. POKE 1 sets all players, A;; playfields, and the background. POKE 2 sets players 0&1, all

playfields, players 2&3, and the background. POKE 4 sets all playfields, all players, and the background. POKE 8 sets playfields 0&1, all players, playfields 2&3, and the background.

PD = PEEK(624) returns the value of paddle 0. The following 7 locations return the values of paddles 1 thru 7.

STK = PEEK(632) returns the value of joysticks 0. The following 3 locations return the values of joysticks 1 thru 3.

STG = PEEK(644) returns 0 if joystick 0 trigger is pressed. The following 3 locations return 0 if joystick triggers 1 thru 3 are pressed.

CTR = PEEK(656) returns the cursor row in the text window.

CTC = PEEK(657) + PEEK(658)*256 returns the cursor column in the text window.

IVF = PEEK(694) returns 128 if inverse video or 0 if normal.

SHF = PEEK(702) returns 0 if lower case, 64 if the shift key is pressed, and 128 if the control key is pressed.

POKE 704,X sets the color of player and missile 0 to X. The following 3 locations set the color of players and missiles 1 thru 3.

POKE 708,X sets color register 0 to X. The following 4 locations set color registers 1 thru 4.

OST = PEEK(741) + PEEK(742)*256 returns the O.S. top of memory pointer.

OSB = PEEK)743) + PEEK(744)*256 returns the O.S. bottom of memory pointer.

POKE 752,1 turns the cursor off and POKE 752,0 to turns it back on.

PEEK(753) returns 0 if no key is being pressed.

POKE 755,0 turns off inverse video, POKE 755,2 sets the video to normal, and POKE 755,4 sets vertical reflect.

POKE 756,224 for upper case. POKE 756,226 for lower case.

LAC = PEEK(763) returns the last ATASCII character read or written or a graphics point value.

LKP = PEEK(764) returns the internal code of the last key pressed.

POKE 765,X sets the color for an X10 fill to X.

POKE 766,1 displays control characters. Use POKE 766,0 to react to them.

POKE 767,1 stops screen scroll. Use POKE 767,0 for normal CTRL/1 START/STOP.

POKE 838,163:POKE 839,238 sends the screen display to the printer. POKE 838,166:POKE 839,238 returns to normal.

POKE 842,13 sets normal read mode. POKE 842,12 sets the normal write mode.

POKE 1802,X sets the number of disk drives in your system to X. (See DOS 2 manual for further information.)

POKE 1913,80 turns off the read after write verifications in DOS 20S. POKE 1913,87 resets normal read after write.

POKE 53248,X sets the horizontal position of player 0. Read this location to detect missile 0 for playfield collisions. The next 3 locations are similar for the next 3 players and missiles.

POKE 53252,X sets the horizontal position of missile 0. Read this location to detect player 0 for playfield collisions. The next 3 locations are similar for the next three missiles and players.

POKE 53256,X sets the size of player 0. POKE 0 for normal, 1 for double, and 3 for quadruple width. Read this location to detect missile 0 for player collisions. The following 3 locations are similar for players and missiles 1 to 3.

POKE 53279,0 clicks the console speaker. Read this location to detect OPTION, SELECT, and START buttons. If no button is pressed, PEEK(53279) will return 7, 6 means START is pressed, 5 for SELECT, and 3 for OPTION.

POKE 53760,X SOUND for channel 1 frequency. 53762 = ch2, 53764 = ch3, 53766 = ch4.

POKE 53761,X SOUND for channel 1 control. 53763 = ch2, 53765 = ch3, 53767 = ch4.

POKE 53768,X SOUND CONTROL. Used to set sound filters, distortions, etc.

PEEK(53775) contains 251 if a key is pressed, 255 if no is key pressed, and 247 if the shift key is pressed.

LOADING INSTRUCTIONS
BASIC ROUTINES FOR THE ATARI

To avoid potential problems, please take a few moments to read the following instructions before running any of the BASIC routines.

Disk: First, insert the ATARI BASIC cartridge into the left slot (the center slot on the ATARI 400). Next, insert the BASIC Routines disk into the disk drive. Turn the computer on. The disk will read for a few seconds and BASIC Routines menu will be displayed. To select a routine, type the number of the Routine.

Tape: To load a routine from cassette tape, you will have to cue the tape to the beginning of the desired routine. (If you have an ATARI recorder, it will be necessary for you to cue the tape on a separate recorder.) Listen for the name of the routine; it will be audible just before it begins. To load any routine other than the ones followed by an asterisk (*) (see routine listing below), cue the tape, type CLOAD, hit RETURN and start the tape. The program will then load. Once it has successfully loaded, the screen will display a READY prompt. Type RUN. The program will then execute.

To load a BASIC list format subroutine (indicated with an asterisk (*) on the listing below), the programs on the tape are in the CLOAD format and will have to be saved to a separate blank tape in the LIST format before they can be used with your programs. Follow these steps:

(1) Cue the subroutine you wish to use on a separate cassette recorder as instructed above.

(2) When the program is cued, type CLOAD and then touch RETURN.

(3) Once the program has successfully loaded, the READY prompt will be displayed

(4) Insert your blank tape into the cassette player and type LIST "C:" Hit RETURN. The subroutine will then be saved to your tape in the LIST format.

To execute a subroutine in a program:

(1) Load the program that you will be using the subroutine with and then remove the tape.

(2) Cue the subroutine you wish to use and type: ENTER "C:". Now, press RETURN.

(3) The subroutine and your program will be merged and stored

into the computer's memory. You can now save the combined routines to your own tape by typing CSAVE.

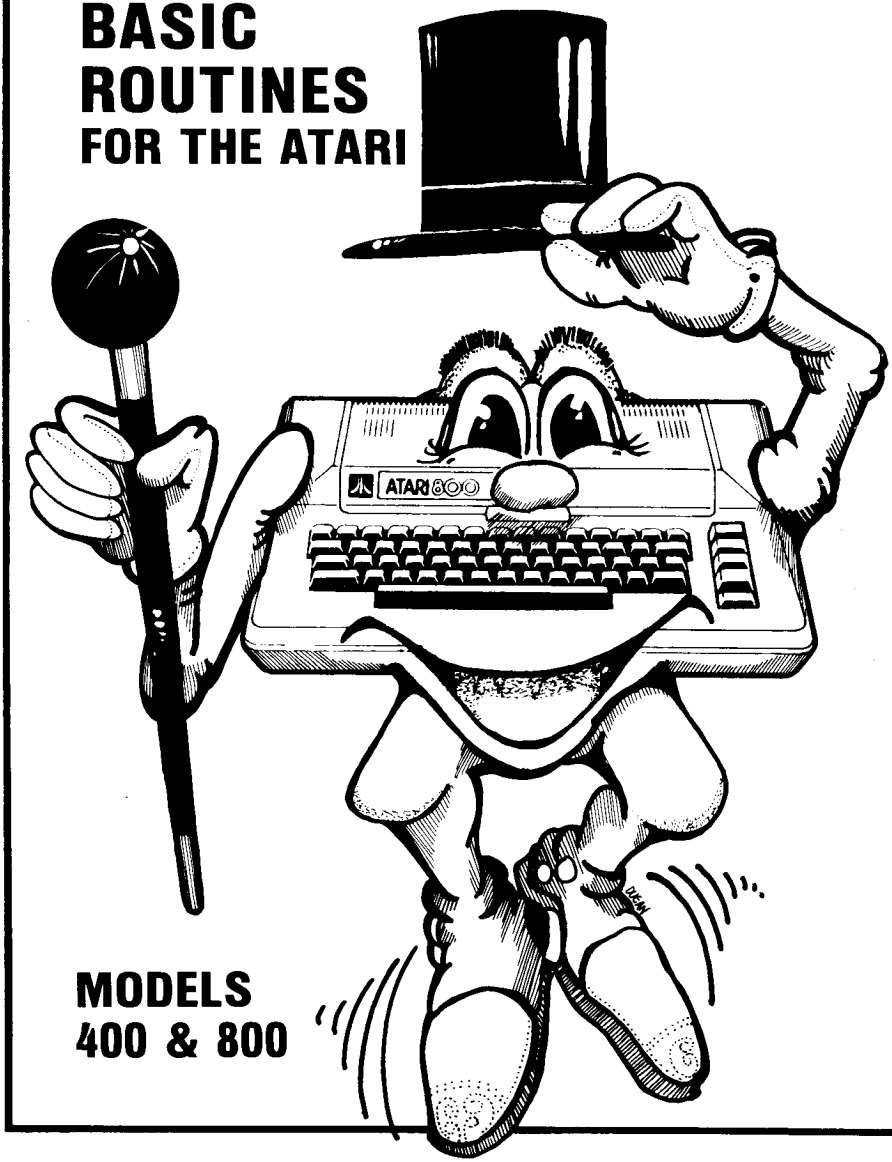
BASIC Routines for the ATARI are recorded in the following order:

BASSNOTE
BINCOV
DICE
ENGLISH
JOYSTICK
GRAPH
INVENT
MUSICEND
PMDEMO
RJUSTIFY
SORTDEMO
SOUNDDEMO
TIMER
HELLO
MODE 123
TABDEMO
GR8TEXT
PADDLE
KEYDEMO
DELETE *
A *
B *
D *
F *
G *
I *
E *

NOTE: Always type LPRINT before you save a program to tape. This ensures the integrity of your program. Programs can be saved by using one of two methods: (1) CSAVE or (2) LIST "C:".

DOCUMENTATION

BASIC ROUTINES FOR THE ATARI



MODELS 400 & 800