PROGRAMMING EXAMPLES

Some miscellaneous examples and discussions

of programming on the ATARI 400/800^{tm} Home Computer System

1) Formatting Dollar Amounts
2) Bubble-Sort
3) Rocksort
4) Real-time Clock
5) Getting Data From the Keyboard

ATARI INC.
CONSUMER PRODUCT SERVICE
PRODUCT SUPPORT GROUP
1312 Crossman Ave.
Sunnyvale CA 94086

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

DEMOPAC #3
Rev.2 5-82/JB

DISCLAIMER OF WARRANTY ON PROGRAMS CONTAINED HEREIN

All computer programs contained herein are distributed on an "as is" basis by Atari, Inc. ("Atari") without warranty of any kind. Any statements concerning the capabilities or utility of the computer programs are not to be construed as express or implied warranties.

Atari shall have no liability or responsibility to the user or any other person or entity with respect to any claim, loss, liability, or damage caused or alleged to be caused directly or indirectly by the computer programs, contained herein. The entire risk as to the quality and performance of such programs is with the user.

Every effort has been made to ensure the accuracy of this document. However, because of ongoing improvements and updating of our computer software and hardware, Atari cannot guarantee the accuracy of printed material after the date of publication and disclaims any liability for changes, errors, or omissions.

Correspondence regarding this pack should be forwarded to Manager of Technical Support, Consumer Product Service, Atari, Incorporated, 1312 Crossman Avenue, Sunnyvale, CA 94086.

FORMATTING DOLLAR AMOUNTS

DEB 5/82

The following program illustrates a method of formatting dollar amounts, on the screen or on a printer. This example accepts input of number amounts between 0 and 9999.99.

The program checks each input number and rounds it off to two decimal places, adding trailing zeroes if necessary. The number is turned into a string, and concatenated onto a dollar sign. There is some error-checking, for non-numeric or out-of-range input. Output is then formatted into evenly spaced columns.

There are two versions of the program. The first version stores the input data in a long-string array in memory, and prints the output on a printer. The second version creates a data file to store the data on cassette or diskette. It then prints the output on the screen.

```
1 REM FORMATTING DOLLAR AMOUNTS
2 REM VERSION 1
3 REM DED 5/82
4 REM this version stores data in a long-string array
5 REM and prints output on a printer.
6 REM ************************************************************
10 REM initialization
20 DIM AMOUNT$(20),DOLLAR$(25),YNS$(3),ARRAY$(1000),SPACE$(12)
30 SPACE$="            ":REM format output for evenly spaced columns
40 COUNT=1:REM set counter for number of inputs
50 TRAP 50:PRINT "VALUE...";:INPUT AMOUNT:TRAP 40000
55 REM *************** format each amount ***************
60 SIGN=SGN(AMOUNT):AMOUNT=ABS(AMOUNT):REM keep track of sign
70 AMOUNT=INT((AMOUNT+5.0E-03)*100)/100:REM round for dollar amount
80 AMOUNT=AMOUNT*SIGN:REM restore sign
90 AMOUNT$=STR$(AMOUNT):REM turn number into string
100 DOLLARLEN=LEN(STR$(INT(AMOUNT))):REM separate dollar from cents
110 CENTSLEN=LEN(AMOUNT$)-DOLLARLEN
120 IF CENTSLEN=0 THEN AMOUNT$(LEN(AMOUNT$)+1)=".00":REM check for trailing zer
130 IF CENTSLEN=2 THEN AMOUNT$(LEN(AMOUNT$)+1)="0"
140 DOLLAR$="$    ":REM string containing dollar sign and 4 spaces
150 TRAP 50:DOLLAR$=DOLLAR$(1,5-DOLLARLEN):REM allow only 4 digit number
160 DOLLAR$(LEN(DOLLAR$)+1)=AMOUNT$:REM concatenate to dollar sign
170 ARRAY$(LEN(ARRAY$)+1)=DOLLAR$:REM add element to array
180 PRINT "ANY MORE";:INPUT YNS$
190 IF YNS$(1,1)<>"Y" THEN IF YNS$(1,1)<>"N" THEN GOTO 180
200 IF YNS$(1,1)="N" THEN GOTO 240
210 IF COUNT=40 THEN ? "ARRAY FULL":GOTO 240
220 COUNT=COUNT+1:GOTO 50:REM get another number
230 REM ************************************************************
235 REM printout routine
240 OPEN #2,8,0,"P:":REM open printer file
250 FIELD=1:REM determines location of element in string array
260 COUNT=0:REM check for end of line
270 TRAP 310:PRINT #2;ARRAY$(FIELD,FIELD+7);SPACE$;
280 FIELD=FIELD+8:COUNT=COUNT+1
290 IF COUNT=4 THEN PRINT #2:GOTO 260:REM execute carriage return at end of lir
300 GOTO 270
310 PRINT #2:CLOSE #2:REM close files when data is finished
320 END
```

```
1 REM FORMATTING DOLLAR AMOUNTS
2 REM VERSION 2
3 REM DEB 5/82
4 REM this version stores data in a data file on cassette or disk
5 REM and prints the formatted output on the screen
6 REM ************************************************************************
10 REM initialization
20 DIM AMOUNT$(20),DOLLAR$(25),YNS$(3)
30 REM OPEN #1,8,0,"C:":REM for cassette use
40 OPEN #1,8,0,"D:PRINT.FMT":REM for diskette use
50 TRAP 50:PRINT "VALUE...";:INPUT AMOUNT:TRAP 40000
55 REM ******************** format each amount **********************
60 SIGN=SGN(AMOUNT):AMOUNT=ABS(AMOUNT):REM keep track of sign
70 AMOUNT=INT((AMOUNT+5.0E-03)*100)/100:REM round for dollar amount
80 AMOUNT=AMOUNT*SIGN:REM restore sign
90 AMOUNT$=STR$(AMOUNT):REM turn number into string
100 DOLLARLEN=LEN(STR$(INT(AMOUNT))):REM separate dollars from cents
110 CENTSLEN=LEN(AMOUNT$)-DOLLARLEN
120 IF CENTSLEN=0 THEN AMOUNT$(LEN(AMOUNT$)+1)=".00":REM check for trailing zer
130 IF CENTSLEN=2 THEN AMOUNT$(LEN(AMOUNT$)+1)="0"
140 PRINT
150 DOLLAR$="$     ":REM string containing dollar sign and 4 spaces
160 TRAP 50:DOLLAR$=DOLLAR$(1,5-DOLLARLEN):REM allow only 4 digit number
170 DOLLAR$(LEN(DOLLAR$)+1)=AMOUNT$:REM concatenate to dollar sign
180 PRINT #1;DOLLAR$:REM print formatted string to data file
190 PRINT "ANY MORE";:TRAP 50:INPUT YNS$
200 IF YNS$(1,1)="N" THEN GOTO 230
210 IF YNS$(1,1)<>"Y" THEN IF YNS$(1,1)<>"N" THEN GOTO 190
220 GOTO 50
230 CLOSE #1
235 REM ************************************************************************
240 REM get data from data file and print in columns on screen
250 REM OPEN #1,4,0,"C:":REM use for cassette; position tape first
260 OPEN #1,4,0,"D:PRINT.FMT":REM use for diskette
270 POKE 82,0:REM move margin out for even columns
280 OPEN #2,8,0,"S:"
290 ? :? :PRINT #2;"        FORMATTED DOLLAR AMOUNTS";:? :?
300 TRAP 330:INPUT #1;DOLLAR$:REM bring in record
310 PRINT #2;DOLLAR$,:REM comma inserts spaces for columns
320 GOTO 300
330 CLOSE #1:CLOSE #2:REM close files when data is finished
340 END
```

```
1 REM BUBBLE-SORT
2 REM PY/JB 4/82
3 REM the following program illustrates a simple sort process
4 REM :the user inputs numbers, which are kept in a numeric array.
5 REM :the array is then sorted and printed out in order.
6 REM ************************************************************
10 DIM A(100):REM this array holds the data to be sorted
20 PRINT "HOW MANY ITEMS TO SORT";:INPUT TOTAL
30 FOR I=1 TO TOTAL
40 PRINT "ENTER A NUMBER...";:INPUT NUMBER
50 A(I)=NUMBER:REM assign data element to array
60 NEXT I
65 REM print out unsorted list for comparison
70 FOR I=1 TO TOTAL
80 PRINT A(I)
90 NEXT I
100 REM ************************************************************
101 REM sort the array
105 PASS=0:REM keep track of how many times through the list
110 FLAG=0:REM flag=1 indicates that more sorting is necessary
120 FOR I=1 TO TOTAL-1
130 IF A(I+1)>=A(I) THEN 180:REM if this item is less than the next,
135 REM they are in the right order, so skip to the next item
140 TEMP=A(I):REM if they're in the wrong order, store the item temporarily
150 A(I)=A(I+1):REM in order to
160 A(I+1)=TEMP:REM switch the order
170 FLAG=1:REM set the flag to show that a change was made
180 NEXT I:REM check the next item
190 IF FLAG=1 THEN PASS=PASS+1:PRINT "PASS=";PASS:GOTO 110
191 REM after each pass, check the flag
192 REM to see if any changes were made: if so, try again.
195 REM ************************************************************
199 REM print out the sorted list
200 PRINT "SORTED LIST..."
210 FOR I=1 TO TOTAL
220 PRINT A(I)
230 NEXT I
240 PRINT "--END OF PROGRAM--"
250 END
```

```
1 REM ROCKSORT
2 REM WB/JB 4/82
3 REM sort a string of up to 80 characters
4 REM by making the biggest ones 'fall through' to the bottom.
5 REM there is background music, and the sort is timed.
6 REM ************************************************************
10 DIM SORT$(80),BUF$(1),TONE(80):REM set up variables
20 GRAPHICS 0:POKE 82,0:REM clear screen, set left margin at column 0
30 FOR I=1 TO 80 STEP 2:READ T:TONE(I)=T:TONE(I+1)=T:NEXT I
31 REM set up array with tone data for the sound statement
40 DATA 29,31,33,35,37,40,42,45,47,50,53,57,60,64,68,72,76,81,85,91,96
41 DATA 102,108,114,121,128,136,144,153,162,173,182,193
42 DATA 204,217,217,230,230,243,243
45 REM ************************************************************
50 PRINT "80 CHARACTER STRING TO BE SORTED..."
55 INPUT SORT$
60 LAST=LEN(SORT$):REM keep track of where to stop sorting
70 GRAPHICS 2:PRINT #6;SORT$:REM display string in large letters
80 POKE 752,1:REM disable cursor
90 POKE 18,0:POKE 19,0:POKE 20,0:REM initialize real-time-clock
99 REM ************************************************************
100 REM the following section contains the actual sort,
101 REM along with the background music
110 FOR I=LAST-1 TO 1 STEP -1:REM outer loop
120 SOUND 0,TONE(I),10,10:REM first voice of background music
130 FLAG=0:REM this flag turns 1 if any changes are made during the sort
140 FOR J=1 TO I:REM inner loop
150 SOUND 1,TONE(J),10,8:REM second voice of background music
155 REM --here's the sort itself--
160 IF SORT$(J,J)>SORT$(J+1,J+1) THEN BUF$=SORT$(J,J):SORT$(J,J)=SORT$(J+1,J+1
SORT$(J+1,J+1)=BUF$:FLAG=1
165 REM if top item is larger, exchange places with next item, and set flag
170 POSITION 0,0:PRINT #6;SORT$:REM display latest version of string
180 NEXT J:REM inner loop
190 IF FLAG=0 THEN I=1:REM check flag-if no changes, skip to last loop
200 NEXT I:REM outer loop
205 REM ************************************************************
206 REM --finish up--
210 POKE 752,0:REM re-enable cursor
220 SOUND 1,0,0,0:REM turn off second voice
230 FOR I=29 TO 0 STEP -1:SOUND 0,I,10,10:NEXT I:REM 1st voice signals end
240 GOSUB 1000:REM call subroutine which figures elapsed time
250 PRINT "ELAPSED TIME ";HH;":";MM;":";SS;
260 END
265 REM ************************************************************
998 REM the following subroutine retrieves the real-time clock values
999 REM and turns the results into hours(HH), minutes(MM) and seconds(SS)
1000 SS=(PEEK(18)*256*256+PEEK(19)*256+PEEK(20))/60
1010 HH=INT(SS/(60*60))
1020 SS=SS-HH*60*60
1030 MM=INT(SS/60)
1040 SS=INT(SS-MM*60)
1050 RETURN
```

```
1 REM CLOCK
2 REM WB/DM/JB 4/82
3 REM this program sets and displays a real time clock in large characters
4 REM and sounds chimes on the quarter hour, half hour and hour
5 REM ************************************************************
6 REM synchronize with real-time
10 GRAPHICS 0:PRINT "     SET THE REAL-TIME CLOCK":PRINT
20 TRAP 20:PRINT "HH:";:INPUT HH
30 TRAP 30:PRINT "MM:";:INPUT MM
40 TRAP 40:PRINT "SS:";:INPUT SS
50 TRAP 40000:REM turn off error trap
60 PRINT :PRINT "     PLEASE STAND BY..."
70 SECONDS=HH*60*60+MM*60+SS:REM number of seconds in set-time
80 T=INT(SECONDS*59.923334):REM number of clock-ticks, from NTSC clock-rate
90 MSB=INT(T/(256*256)):REM most significant byte
100 T=T-MSB*256*256:NSB=INT(T/256):REM next-most significant byte
110 LSB=T-NSB*256:REM least significant byte
120 POKE 18,MSB:POKE 19,NSB:POKE 20,LSB:REM set the clock locations
130 GRAPHICS 2+16:REM mode 2 with no text window for time-display
135 REM ************************************************************
136 REM --read clock and display time--
140 T=INT((PEEK(18)*256*256+PEEK(19)*256+PEEK(20))/59.923334)
145 REM turn clock-ticks into hours/minutes/seconds
150 HH=INT(T/(60*60)):REM hours
160 T=T-HH*60*60:MM=INT(T/60):REM minutes
170 SS=T-MM*60:REM seconds
180 IF HH=24 THEN POKE 18,0:POKE 19,0:POKE 20,0:REM reset to 0 at midnight
190 POSITION 5,6:PRINT #6;HH;":";MM;":";SS,:REM display time
195 POKE 77,0:REM disable attract mode
198 REM ************************************************************
199 REM --chimes--
200 REPEAT=0:TONE=121:IF SS<>0 THEN GOTO 140:REM sound chime on 0 seconds
210 IF MM=0 THEN REPEAT=4:GOSUB 500:REPEAT=HH:GOTO 300:REM chimes + hours
220 IF MM=15 THEN REPEAT=1:GOSUB 500
230 IF MM=30 THEN REPEAT=2:GOSUB 500
240 IF MM=45 THEN REPEAT=3:GOSUB 500
250 GOTO 140:REM go get new time value
290 REM ************************************************************
299 REM sound hours
300 FOR DELAY=1 TO 300:NEXT DELAY
310 FOR R=1 TO REPEAT:FOR VOLUME=15 TO 0 STEP -1
320 FOR WAIT=0 TO 40:NEXT WAIT
330 SOUND 0,243,10,VOLUME:SOUND 1,182,10,VOLUME
340 NEXT VOLUME:NEXT R
350 GOTO 140:REM go get new time value
390 REM ************************************************************
399 REM the following subroutine sounds the Westminster chimes
500 RESTORE 600+REPEAT:REM read the proper data line
510 FOR BELL=1 TO 5*REPEAT:REM 5 tones in each repetition
520 READ TONE:GOSUB 560:REM read and sound each note
530 NEXT BELL
540 RETURN
550 REM ** this small subroutine sounds each note
560 FOR VOLUME=15 TO 0 STEP -1
565 FOR WAIT=1 TO 10:NEXT WAIT
570 SOUND 0,TONE,10,VOLUME
580 NEXT VOLUME
590 RETURN
599 REM ** here is the data for the tones
601 DATA 72,81,91,121,0
602 DATA 91,72,81,121,0,91,81,72,91,0
603 DATA 72,91,81,121,0,121,81,72,91,0,72,81,91,121,0
604 DATA 91,72,81,121,0,91,81,72,91,0,72,91,81,121,0,121,81,72,91,0
```

GETTING INFORMATION FROM THE KEYBOARD
Alternatives to INPUT
JB 5/82

There are several reasons why you may need to get keyboard input in your program, and yet not wish to use the INPUT statement. You may not want to press RETURN after every entry, for example, or you may not want the input prompt. In any case, here are some alternatives to the BASIC INPUT statement.

There are three methods for getting input from the keyboard without an ordinary INPUT statement. You can: (1) look directly at the memory location where the keycode for the last key pressed is stored; (2) open the keyboard as a device, with the OPEN statement, and GET# from that file; or (3) INPUT # from the file. Each of these methods has advantages and disadvantages. You should choose the method or combination of methods that suits your needs.

To get directly from the keyboard buffer, check the value of PEEK(764). This location returns a keycode, rather than a letter or ATASCII value. The keycode is an arbitrary code which is unique for each key. It reflects whether SHIFT or CONTROL is pressed, and in some cases whether SHIFT and CONTROL are both pressed. If you use this method, you must translate the keycode in your own program. There is a translation table on page 50 of Tech User Notes (CO16555), or you can just PEEK the location and create your own table. This method is useful if you are simply checking for a particular key, i.e. PRESS "C" CONTINUE. The program for this would be:

```
10 PRINT "PRESS C TO CONTINUE"
20 IF PEEK(764)<>18 THEN GOTO 20
30 ...the rest of your program
```

Checking CH, the keyboard buffer, bypasses the keyboard handler altogether. A disadvantage is that you have to interpret the keycode yourself. An advantage is that you don't have to press RETURN, and in bypassing the handler, you get rid of the keyboard beep. There is no other way of turning off the beep, short of disconnecting the keyboard speaker.

There are two more types of keyboard input. If you open the keyboard as a file, using the OPEN statement, you can either GET# or INPUT# from that file. GET# returns an ATASCII code number for the character, and INPUT# behaves like an ordinary INPUT, except that there is no prompt, and the characters are not displayed on the screen, but go directly into the INPUT variable. Here are examples of GET# and INPUT#.

```
10 OPEN#1,4,0,"K:" : REM this opens the file
20 GET#1,X:REM it waits for you to hit a key
30 CLOSE#1:REM close the file
40 PRINTX,CHR$(X):REM print code and character
```

There is no prompt with GET#, and you do not have to press RETURN. You can GET only one character at a time, and you must remember that a number is returned, the ATASCII value of the letter.

```
10 DIM A$(20):GR.2+16:REM no text window
20 OPEN#1,4,0,"K:" :REM open file
30 INPUT#1,A$:REM it waits for RETURN
40 POSITION 5,5:PRINT#6;A$:REM display on screen
50 GOTO 50 :REM keep image on screen
```

There is still no prompt, and the characters are not actually printed until you press RETURN. Regular characters are returned, so you should use a string variable.

These methods can be combined to suit your needs. You might wish to check CH for a change before opening a keyboard file, for example. Remember that CH doesn't change back; you might want to POKE a 255 back into it after you read it. When using a full-screen graphics mode, avoiding the INPUT statement keeps the screen from returning to Graphics 0 for the input prompt.