# The SORCERER'S APPRENTICE

Vol. 1, No. 1
May, 1979

A publication of the
Sorcerer User's Group

Send questions, comments
to Dave Bristor
1530 Washtenaw
Ann Arbor, MI 48104

## MISCELLANEOUS NOTES

This is the #1 newsletter of The Sorcerer User's Group. Issue #2 will appear if this issue generates enough interest. I have a lot of interest in printing this paper; I feel it is necessary to communicate with other Sorcerer users and find out what they are doing.

This is not a very big issue. The Sorcerer's Apprentice will continue to grow only with your support. Please send in programs, hardware designs, or whatever. I will print virtually anything. Communication of ideas is the object here. If you have a program to send, I would prefer a listing and a cassette if possible. Copying programs from the screen of my TV is very error prone, as I have found out. Cassettes can be returned only if sufficient postage is included. Nobody can be paid for any material submitted; this is a non-profit organization. I am in it only for fun. Computing is still just a hobby for me.

A little about myself. I am David (Dave) Bristor. I am a junior in computer engineering at the University of Michigan in Ann Arbor, Michigan. My system consists of a 16K Sorcerer, a Panasonic tape recorder, a Pixie-Verter, and a Sony TV. I would like to see material printed on these pages that would run on any system, not just the double-disk, line printer jobs. On the other hand, I would be delighted to print a complete operating system that allows the Sorcerer to be used as an intelligent terminal. As I said earlier - I will print virtually anything.

Does anyone out there have a printer available? I would like to be able to type the Apprentice onto a cassette, send it to someone with a printer, and get back the cassette and a listing that could be taken to the printers. Typewriters and I don't get along very well. A printer would also be convenient for obtaining listings for the Apprentice.

I have seen a rough draft of the Sorcerer Technical Manual. It has a memory map, a parts list, some info on the USR function, and some other goodies. I don't yet know when this valuable document will be available, but I'll let you know. Additionally, I've put some information on USR in this issue; see the program description.

I am sending copies of this issue, along with letters of explanation, to various companies in the industry that are somehow connected with the Sorcerer. Hopefully this will allow me to clue you in on new Sorcerer related products as soon as they become available. If any of you are planning to market something for the Sorcerer, let me know by sending a short description and I'll let people know through this column. Major magazines are also receiving copies; through them we should be able to add to our ranks.

# PROGRAM OF THE MONTH

contributed by Dave Bristor
Listing on last page

This program is not very impressive as it stands. But with a little creativity on the part of the user, parts of it can become important subroutines in larger programs. In addition, it explains some techniques useful in any program. The program is fairly well documented, but I'd like to explain some of the methods used.

Many of you have probably heard of structured programming; possibly you have heard also that structured programming is not possible with BASIC. While there is some degree of truth to this, the readability of BASIC programs can be improved by using the statement separator, a colon (:), to indent lines of code. This is especially useful in FOR-NEXT loops, where the body of the loop is indented to separate it from the rest of the program. Subroutines can also be indented one or two spaces to aid in their identification in listings. Indentation can be very handy when listing long programs, and it may help others in reading your code also.

All this program does is move a dot (graphic-5) around the screen. This could be done with a PRINT statement, but instead I chose the fancier route of employing the USR function along with a machine language subroutine. I learned a lot from this that I'd like to pass on.

The USR function has the form: 80 A=USR(0). The variable A is a dummy variable; it should not be used elsewhere in the program. Any variable may be used, A is just for the purposes of this example. When executed, the statement causes a CALL to the machine language subroutine whose address is at locations 259 and 260 (decimal). This address must be supplied by the user. Return to the BASIC program can be made through the RET instruction. (CALL and RET are Z-80 instructions similar to the BASIC GOSUB and RETURN statements.)

The GET statement is not in the Sorcerer's BASIC vocabulary. I first found out about it from experience with a PET. It has the form: 20 GET A. When executed, the computer makes a quick scan of the keyboard. If a key is currently down, it's value is given to the variable A. If no key is pressed, the value of A is unchanged. The GET statement is also useable with the string variables; i.e. GET A$. In the example program, the combination of the USR statement and the machine language subroutine that accompanies it make the equivalent of a GET statement. When the USR function is executed in our program, program control is transferred to the subroutine, which executes as if it were a GET statement. The source code in assembler and resulting machine code in hex follow for those interested:

```
0000  CD 18 E0  START    CALL  KEYBRD      CALL KEYBOARD INPUT ROUTINE
0003  32 0A 00            LD    (VALUE),A  STORE VALUE FROM INPUT
0006  C9                 RET              RETURN TO BASIC
0007             VALUE    DEFS  01         STORAGE SPACE FOR CHARACTER
```

# LISTING

```
1 REM PROGRAM MOVES DOT (GRAPHIC-5) AROUND SCREEN
2 REM RUN PROGRAM, HIT KEYS 2,4,6,8 FOR DOWN,LEFT,RIGHT,UP
3 REM MOVEMENT OF DOT
5 REM LINES 10-50 LOAD MACINE LANGUAGE SUBROUTINE AT 0-6 HEX
10 DATA 205,24,224,50,7,0,201
20 FOR I=0 TO 6
30 :  READ A
40 :  POKE I,A
50 NEXT I
54 REM STORE SUBROUTINE ADDRESS
55 POKE 259,0:POKE 260,0
60 LOC=-3040:REM -3040 IS CORRECTED CENTER OF SCREEN
70 POKE LOC,132:REM PUT DOT AT CENTER OF SCREEN
80 A=USR(0):REM CALL MACHINE LANGUAGE SUBROUTINE
90 C=VAL(CHR$(PEEK(7))):REM CONVERT FROM HEX TO DECIMAL
100 ON C/2 GOSUB 200,300,400,500:REM MAKE APPROPRIATE MOVE
110 GOTO 80:REM INFINITE LOOP PROGRAM, BREAK WITH CONTROL-C
120 END
200 REM MOVE DOWN
210 IF LOC+64   -2049 THEN RETURN:REM RETURN IF MOVE WOULD PUT DOT
211 REM OFF SCREEN
220 POKE LOC,32:REM CLEAR OLD LOCATION OF DOT
221 REM OMIT LINE 220 FOR TRAIL OF DOTS
230 LOC=LOC+64:REM SET LOC TO NEW VALUE
240 POKE LOC,132:REM PUT DOT ON SCREEN AT LOC
250 RETURN
300 REM MOVE LEFT
310 IF LOC-1   -3968 THEN RETURN
320 POKE LOC,32
330 LOC=LOC-1
340 POKE LOC,132
350 RETURN
400 REM MOVE RIGHT
410 IF LOC+1   -2049 THEN RETURN
420 POKE LOC,32
430 LOC=LOC+1
440 POKE LOC,132
450 RETURN
500 REM MOVE UP
510 IF LOC-64   THEN RETURN
520 POKE LOC,32
530 LOC=LOC-64
540 POKE LOC,132
550 RETURN
```

The POKE statement is used in the example to move the dot around the screen. Each character position on the screen has it's own address ranging from 61568 for the upper left corner to 63487 for the lower right corner. The center of the screen lies halfway between 62495 and 62496; I chose 62496 as the center for the example. With statements such as POKE and PEEK, the address must be less than 32768. To POKE into a higher memory location, subtract 65536 from the actual address. Thus 61568 becomes -3968 and so on.

---

That's about it for issue #1. Your comments, suggestions, and ideas are more than welcome. I am especially in need of printable material. And to continue receiving copies of The Sorcerer's Apprentice, send another SASE towards the end of June. That is the tentative date of release for the next issue. Until then, happy computing!