



No. 1, April 1988

\$3.00

*For beginning
programmers with no prior
programming experience*

CONTENTS

- 3 Teach Yourself BASIC
- 13 Browsing BASIC
- 23 Teach Yourself
QuickBASIC

PUBLISHER'S STATEMENT: *The BASIC Teacher* is published monthly by Different Worlds Publications, 2814 - 19th Street, San Francisco, CA 94110. Contents copyright © 1988 by Different Worlds Publications. All rights reserved. Contents may be copied and distributed freely. Address all correspondences to *The BASIC Teacher*, 2814 - 19th Street, San Francisco, CA 94110.

SUBSCRIPTION INFO: A 12-issue sub in the U.S. and Canada is \$36. Overseas subs are \$42 by surface mail, \$54 by air.

PRINTED IN THE U.S.A.



WELCOME TO the first issue of *The BASIC Teacher*. This issue contains opening articles on three BASIC strands that will continue in future issues:

"Teach Yourself BASIC" begins with a tutorial on Microsoft BASIC (BASICA and GW-BASIC for MS-DOS computers such as the Tandy 1000 TX, IBM PC, and a multitude of clones). It starts at ground zero, square one, the beginning. No previous programming experience required. In future issues, this strand will include problems for you to solve and one or more solutions to previously posed problems.

"Browsing BASIC" provides discussions about randomly selected Microsoft BASIC and QuickBASIC topics. The opening article explores the use of function keys to provide shortcuts, assigning new functions to the keys, and disabling functions previously assigned to the keys. You can define function-key operations to help you do what you want to do.

"Teach Yourself QuickBASIC" is a beginner's series about Microsoft's revolutionary new QuickBASIC, version 4, which ushers in a new era of computer languages. Future articles in

this strand will slowly lead you through the features of this structured programming language. If you have some background in Microsoft's BASICA or GW-BASIC, you will find the transition to QuickBASIC is gentle and easy. If you have no previous experience in using programming languages, begin here.

Additional strands will be added in future issues of *The BASIC Teacher*. Here are some possibilities:

"Imitations of Life" — A strand about the art and science of simulation and simulation games.

"Adventures in FileLandia" — A strand that surely, but slowly, explores the world of files.

"BASIC for Math & Science" — A strand that explores the world of problem-solving in the scientific world.

"Exploring Graphics" — Opening and closing windows, color, geometric shapes, animation, etc.

You, our readers, are welcome to send in suggestions for strands or specific areas of computer application in which you are interested.

Special Reader Services

FOR THE further enjoyment of your Volkscomputer, we recommend the following books for your computer library shelf:

The books below are available at local bookstores. Or, as a special reader service, you can order them thru us by sending a check or money order to:
The BASIC Teacher, 2814 - 19th Street, San Francisco, CA 94110.

Please add \$2 for the first book and \$1 for each additional book to cover postage and handling charges. California residents must add appropriate sales tax.

Using QuickBASIC
By Don Inman and Bob Albrecht
(Osborne/McGraw-Hill, 436pp, \$19.95)

Here's an excellent programming guide to Microsoft's newest version of QuickBASIC by the authors of *The BASIC Teacher*. The book approaches QuickBASIC's programming environment in three stages so beginning and experienced BASIC programmers can find the appropriate level of instruction.

DOS Made Easy
By Herbert Schildt
(Osborne/McGraw-Hill, 385pp, \$18.95)

Previous computer experience is not necessary to understand this concise, well-organized introduction that's filled with short applications and exercises. The book walks you thru all the basics, beginning with an overview of a computer system's inner components and a step-by-step account of how to run DOS for the first time.

The ShareWare Book
Using PC-Write, PC-File, PC-Talk
By Emil Flock, et al
(Osborne/McGraw-Hill, 688pp, \$14.95)

Covers the most popular "free" programs: PC-Write, a word processor; PC-File, a database manager; and PC-Talk, a telecommunications program. These programs are available thru user groups or bulletin-board services in return for a nominal registration fee. The book has all the details on how you can obtain these program disks.



Introduction

RELAX, MAKE YOURSELF COMFORTABLE.
Get ready to go on an *Adventure in Learning*.

You can teach yourself how to read and understand BASIC, the People's Computer Language, used by more people than any other **computer language**.

You can learn how to use Microsoft GW-BASIC, also known as Microsoft BASICA, built-into or bundled with more than 20,000,000 IBM PC, Tandy 1000, or compatible computers—today's *Volkscomputers*.

A **computer language** is a language used to communicate with a computer.

Compared to human languages—such as English, Spanish, or Japanese—a computer language is very simple. BASIC has a small **vocabulary** (list of words it knows), and a simple **syntax** (rules of grammar that it follows).

When you begin to understand a little BASIC, you may wish to tell the computer to do what *you* want it to do the way *you* want the computer to do it.

You may write your own original, never-before-seen-on-Earth-or-anywhere-else **programs**—*your* programs!

A Program

A PROGRAM is simply a set of instructions, a plan for doing something. You may have already used or created a program. For example:

- A recipe for baking cookies.
- Instructions for opening a combination lock.
- Directions on how to get to someone's house.
- Instructions for assembling a model or a toy.

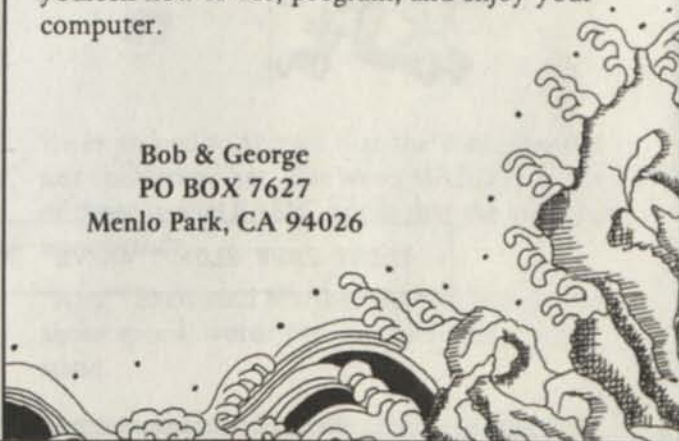


A **BASIC program** is a set of instructions that tells the computer what to do and how to do it in the language the computer understands—BASIC. A set of instructions to make the computer do what *you* want it to do, following the rules of BASIC, is called a **program**—*your* program.

YOU CAN DO NOTHING WRONG

YOU CANNOT harm the computer by typing a mistake. You may make some, but this is a natural part of exploring and learning. Risk it! Try it and find out what happens. You can learn more from your own patient exploration than from this or any other book. So explore, enjoy, and tell us about your discoveries as you teach yourself how to use, program, and enjoy your computer.

Bob & George
PO BOX 7627
Menlo Park, CA 94026



We Assume You Know a Little DOS

WE ASSUME that you know how to use the **Disk Operating System (DOS)** for your computer. Ours is called **MS-DOS**, which means Microsoft Disk Operating System. You use DOS to load BASIC into the computer.

If you do not know a little DOS, or know a little and want to learn more, we recommend the following book:

- *DOS Made Easy*
by Herbert Schildt
Published by Osborne
McGraw-Hill, 2600
Tenth Street, Berkeley,
CA 94710.



We use a Tandy 1000TX with MS-DOS and GW-BASIC, version 3.20, both resident on a single 3½" disk. To get started, we insert the MS-DOS disk in drive A and turn on the computer. When we see the MS-DOS prompt (the letter "A" followed by a right arrow), we type **BASIC** and press the ENTER key. If that does not work on your system, try typing **BASICA** or **GW BASIC** or **GW-BASIC** . . . or look it up in the reference manual that came with your computer.

Soon we see the opening GW-BASIC screen. The upper left part of the screen looks like this:

GW-BASIC 3.20

(C) Copyright Microsoft 1983,1984,1985,1986

TANDY 1000 GWBASIC 3.20

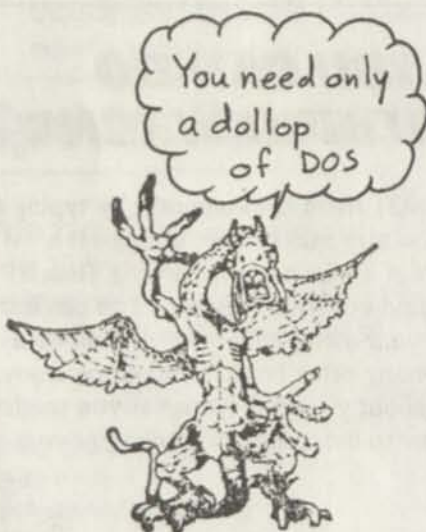
Tandy Version 03.20.01

Licensed to Tandy Corporation

60525 Bytes free

Ok

← Blink, blink, blink—this is the cursor.



The bottom of the screen is the **key line**, which shows labels for the **function keys**. On our keyboard, these keys are labeled F1, F2, F3, and so on, across the top of the keyboard. On your computer they may be elsewhere.

On the screen, the key line looks like this:

```
1LIST 2RUN 3LOAD" 4SAVE" 5CONT 6,"LPT1 7TRON 8TROFF 9TRON 10SCREE
```

For now, just ignore the function keys and the key line. We will tell you more about them later.

Clearing the Screen

WE LIKE to start with an empty screen. Well, almost empty. Here is a way to clear, or erase, the screen:

- Hold down the CTRL key and press the "L" key.

Poof! The screen is empty, except for the cursor in the upper left corner and the key line at the bottom of the screen.

The cursor

—

The key line

1LIST 2RUN 3LOAD"...

When you see the blinking cursor, you know that it is your turn to do something. The computer will (blink, blink) wait patiently (blink, blink) until you (blink, blink) *do* something.

Blink, blink, blink . . .

The blinking cursor tells you it is your turn to do something.



DO THIS:

- Type your name.
- Press the ENTER key.

Well, we do not know your name.

Here is what happened when Mariko typed her name:

- Mariko typed M A R I K O

As Mariko typed her name, the cursor moved to the right.

MARIKO_

- Then she pressed the ENTER key.



MARIKO
Syntax error
Ok

—

We explained to Mariko that the computer did not understand her. The word MARIKO is not of those special BASIC words that the computer understands.

"Aha!" exclaimed Mariko, and she began to learn about special words the computer *does* understand.

Things to Try

Go ahead, press any key.

- Type your name and press the ENTER key.

Ignore any **syntax error** or other error messages.

- Use the CTRL key and the "L" key to clear the screen.
- Hold down the CTRL key and press the BREAK key. You will use this key combination frequently to tell the computer to stop doing what it is doing. Usually, when you hold down CTRL and press BREAK, you will see the computer's cheerful **Ok** message and the cursor.
- Hold down the ALT key and press a letter key or any other key.
- Press lots of keys. Press combinations of keys. Hold down a key.



EXPERIMENT!

Oops? Did the computer freeze and refuse to cooperate?

If things go wrong, as well they might, and nothing seems to work, try the following:

- Hold down the CTRL key and press the BREAK key. If this works, the computer will display **Ok** and, just below it, the cursor. You have regained control.

But if that does not work, you might have to start over. That's okay. Here are some ways to start over:

- Press the CTRL, ALT, and DELETE keys together, all at the same time. This causes the computer to reload DOS. When you see the prompt, type **BASIC** and press the ENTER key. (The DELETE key may be labeled DEL on your computer.)
- Our computer has a big red **reset key** on the front panel. Press the reset key and it reloads DOS. At the prompt, type **BASIC** and press the ENTER key.
- If all else fails, remove the DOS disk, turn the computer off, insert the DOS disk, and then turn on the computer. Yes, it will load DOS. When you see the prompt, type **BASIC** and press the ENTER key.
- If even that does not work, yell for help! There are so many people using computers that someone might hear you and rescue you from your predicament.

BEEP

HERE IS your first BASIC keyword, a word that BASIC understands: **BEEP**.

Your computer has a built-in beeper. You have probably heard it many times.

Make the computer go beep.

- First, clear the screen. Hold down the CTRL key and press the "L" key.

—

- Type **BEEP** and press ENTER.

BEEP
Ok
—

Did you hear a beep?

If you did not hear a beep, perhaps you typed VEEP or BEP or BEAP or BLEAP.

BASIC keywords must be spelled correctly.

However, it is okay to type them in lower case. For example, you can type **beep** and press the ENTER key to make the computer go beep. Go ahead, do it.

*If you want to use the magic of BASIC, you must learn the few words BASIC understands. Some people call these keywords or reserved words. **BEEP** is a special BASIC word, a keyword that tells the computer to go beep. Soon you will learn more special BASIC keywords.*



- Type **beep** and press ENTER.

beep
Ok
—

It is even okay to type **Beep** or **BeeP** or any mixture of upper case and lower case letters.

However . . .

PLEASE REMEMBER

We will always use all upper-case letters for BASIC keywords.



PRINT

YOUR SECOND BASIC keyword is one of the most useful: **PRINT**.

Use PRINT to print information on the screen.

Your computer knows the date and time of day. Well, it does if you set the correct date and time when it first loads DOS. As you will see later, you can set the date and time while in BASIC. You can also tell the computer to print whatever date and time it thinks is current.

- Clear the screen by holding down the CTRL key and pressing the L key.

—

- Type **PRINT DATE\$** and press ENTER.

Here is what we saw when we did it:

```
PRINT DATE$
01-25-1988
Ok
—
```

- Type **PRINT TIME\$** and press ENTER.

Here is what we saw when we did it:

```
PRINT TIME$
08:27:39
Ok
—
```



Of course, if you misspell PRINT, you will see the dreaded **syntax error**. That is okay, just try again. If you misspell DATE\$ or TIME\$, you may see nothing on the line below the line you typed.

- Type **PRINT TYES\$** and press ENTER.
We see an "empty line":

```
PRINT TYES$

Ok
—
```

In addition to the date and time, the computer keeps track of the number of seconds since midnight. Well, midnight according to the computer.

- Type **PRINT TIMER** and press ENTER.

We did it a few seconds after we typed PRINT TIME\$.

```
PRINT TIMER
30468.65
Ok
—
```

Hmmm . . . is that about right? We did it a few seconds after finding out that the time (TIME\$) was 08:27:39. Let's check it:

- Type **PRINT 3600 * 8 + 27 * 60 + 39** and press ENTER.

```
PRINT 3600 * 8 + 27 * 60 + 39
30459
Ok
—
```

We used the computer to do the arithmetic.




Next, we will tell you more about arithmetic.


Arithmetic

YOU CAN tell the computer to do arithmetic and PRINT the answer.

Well now, if you misplace your \$10 solar-powered calculator, just crank up your Volkscomputer and calculate, calculate, calculate. For practice, try some of the examples on the next two pages.




ADDITION: Use the  key. (Hold down a

SHIFT key and press ).

- Type **PRINT 7 + 5** and press ENTER:

```
PRINT 7 + 5
12
Ok
—
```




MULTIPLICATION: Use the  key. (Use a SHIFT key to type *.)

- Type **PRINT 7 * 5** and press ENTER:

```
PRINT 7 * 5
35
Ok
—
```




SUBTRACTION: Use the  key. (Do not use a SHIFT key.)

- Type **PRINT 7 - 5** and press ENTER:

```
PRINT 7 - 5
2
Ok
—
```



DIVISION: Use the  key. (Do not use a SHIFT key.)

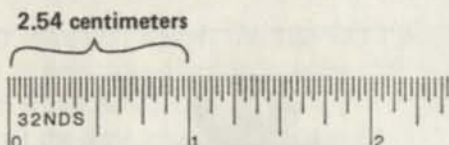
- Type **PRINT 7 / 5** and press ENTER:

```
PRINT 7 / 5
1.4
Ok
—
```



Mariko is 57 inches tall. How tall is she in centimeters? Hmmmm . . . we seem to recall that one inch equals 2.54 centimeters.

- You type `PRINT 57 * 2.54`
- It prints 144.78



Easy! Just multiply the number of inches by 2.54 and print the result. But suppose you know the number of centimeters and want to compute the number of inches?



An ancient ruler named Zalabar measured 100 centimeters from the tip of his nose to the end of his outstretched finger.

How long is that in inches?

- You type
`PRINT 100 / 2.54`
- It prints 39.37008

Call it 39.37. Does that sound familiar? Perhaps you recall that 100 centimeters is equal to one meter, and one meter is equal to 39.37 inches, a little more than one yard.

People usually give their height in feet and inches. If you ask Mariko how tall she is, she will probably tell you she is 4 feet, 9 inches tall. Given feet and inches, it's easy to write a `PRINT` instruction to compute height in centimeters:

- You type `PRINT 4 * 12 + 9`
- It prints 57



Before he reached his full stature, King Kong was once 37' 8" tall. How tall was he then in centimeters?

- You type `PRINT (37 * 12 + 8) * 2.54`
- It prints 1148.08



Aha! Note how cleverly we sneaked in the use of parentheses (). The rules for using parenthesis are very similar to the rules you learned in elementary school math classes. Your computer does the arithmetic inside parentheses first, then does the rest. More about that later.

REMEMBER: 1 inch = 2.54 centimeters
 1 meter = 39.37 inches

In the above, we put in extra spaces to make things easier for you to read and understand. Instead of `PRINT 57 * 2.54`, you can, if you wish, type `PRINT 57*2.54`. Instead of `PRINT 4 * 12 + 9`, you can type `PRINT 4*12+9`.

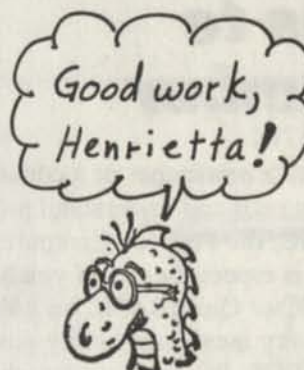
Recently, we took a trip in our ever-faithful car, Henrietta Honda. At the beginning of the trip, Henrietta had 19,832 miles on her odometer. At the end of the trip, her odometer read 20,219. We filled her tank at the beginning and again at the end. She burned 9.3 gallons of gas.

- You type `PRINT (20219 - 19832) / 9.3`
- It prints 41.6129

Well, let's call it about 41.6 miles to the gallon.

Thanks Henrietta!

A kilometer
equals 1000
meters.



Most of the people on Earth use the metric system. Someday, we who live in the U.S.A. will also go metric. Instead of miles, we will use kilometers.

1 kilometer = 0.621371 mile
1 mile = 1.609344 kilometers

How many kilometers did we travel on that trip with Henrietta?

- You type
`PRINT (20219 - 19832) * 1.609344`
- It prints 622.8161

In BASIC, the rules for arithmetic are very similar to the rules we use in "everyday" math. Remember though, to use an asterisk (*) for multiplication and a slash (/) for division.

REMEMBER: 1 kilometer = 0.621371 mile (Okay to use 0.62)
1 mile = 1.609344 kilometers (Okay to use 1.61)

I'm hungry.
How far to the next
bag of oats?

About 10
kilometers.



Things to Remember

IF YOU can read a newspaper or a comic book, you can learn to read and understand programs written in BASIC, the People's Computer Language. This is especially true if you have Microsoft BASIC or QuickBASIC on a Volks-computer, the very inexpensive, very powerful, very useful MS-DOS, PC compatibles—the computers for the rest of us.

BASIC has a small **vocabulary** and a simple **syntax** (grammar). You already know two words of vocabulary—BEEP and PRINT—and a little bit about syntax, including a brief *tete-a-tete* with the dreaded **syntax error**.

You know (we hope!) how to load DOS and BASIC—and how to restart when everything goes wrong.

You have seen, but not yet used, the **key line** at the bottom of the screen, which shows labels for the **function keys** F1, F2, F3, and so on.

You know how to clear the screen by holding down the CTRL key and pressing the L key.

You know that we will always type keywords in all upper case (BEEP, PRINT). However, you may, if you wish, type them in lower case or in a mixture of upper and lower case.

You know how to get the computer to print the date (PRINT DATE\$), the time of day (PRINT TIME\$), and the number of seconds since midnight (PRINT TIMER). You can set the date and time when you first enter MS-DOS. Next time we will show you how to set them within BASIC.

You know how to use the computer as the world's most expensive calculator and do arithmetic using +, -, *, and /.

Best of all, you know:

**YOU CAN DO NOTHING WRONG
MISTAKES ARE OK
PART OF LEARNING
EXPERIMENT
TRY AGAIN
HAVE FUN**



By Don Inman

IN THIS section of *The BASIC Teacher*, we assume that you have a little knowledge of the use of your computer, including being able to load BASIC from MS-DOS. We also assume that you know how to use the following GW-BASIC commands and statements:

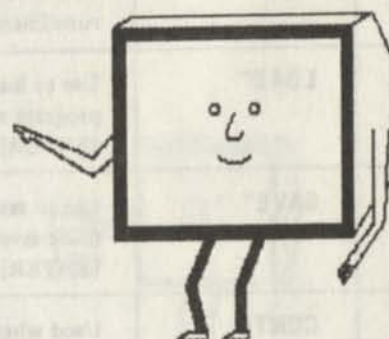
CHR \$()	FOR ... NEXT	LIST	REM	TAB (n)
CLS	GOSUB	LOAD	RETURN	WIDTH
DATA	GOTO	LOCATE	RUN	
DIM	IF ... THEN	PRINT	SAVE	
END	INPUT \$(n)	READ	SPACE (n)	

It would be helpful, although not completely necessary, to know the use of the following commands and statements:

CONT	LPT1	TRON	TROFF	SCREEN
------	------	------	-------	--------

In this issue, we will browse the uses of the **KEY** command which can be used as an immediate command or as a statement within a program. A Tandy 1000 TX computer is used for our demos.

INTRODUCTION



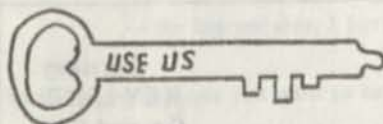
FUNCTION KEYS

WHEN YOU first load GW-BASIC, the screen is set to the text mode with a screen-width of 80 characters. Characters are displayed in white on a black background. You can't help noticing the bottom line (line 25) of the screen. This is the **key line**. It lists the functions that have been assigned to the numbered function keys (F1 thru F10), located at the top of the keyboard on most computers.



KEY LINE

```
1LIST 2RUN 3LOAD 4SAVE 5CONT 6,"LPT1 7TRON 8TROFF 9KEY 10SCREE
```



Do you ignore the function keys when you are programming? You can use them to reduce the number of keystrokes needed to enter programs and also reduce the chance of making typing errors. Pressing one of the function keys gives the same result as typing the command displayed for that particular key.

Some computers have more than ten function keys. The Tandy 1000 TX has twelve numbered function keys. The numbered function keys should not be confused with the four direction (arrow) keys or other user-defined keys, which will be discussed in a future issue. No function has been assigned to keys F11 and F12 on the Tandy 1000 TX. As discussed later, you are free to assign any desired function to these two keys. You may also change the functions assigned to any of the first ten function keys.

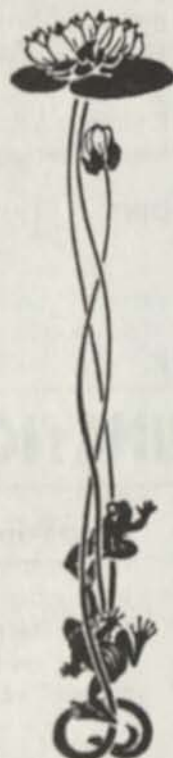
FUNCTION KEYS of Tandy 1000 TX

F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
----	----	----	----	----	----	----	----	----	-----	-----	-----

Here are descriptions of the functions performed by keys F1 thru F10:

Key	Display	Function
F1	LIST	Use to list a program. Must be followed by pressing [ENTER].
F2	RUN←	Use to run a program. Active program runs immediately.
F3	LOAD"	Use to load a program. Name of desired program must follow quote. Then press [ENTER] to load the program.
F4	SAVE"	Use to save a program. Name of program to be saved follows quote. Then press [ENTER] to save the program.
F5	CONT←	Used when a program has been interrupted by STOP. It continues the execution of the program.
F6	,"LPT1:"	Used to select printer as I/O device. I/O command would precede the comma. Command is executed immediately.
F7	TRON←	Turns on tracing function. Executed immediately.
F8	TROFF←	Turns off tracing function. Executed immediately.
F9	KEY	Used for the KEY functions described in this article. Complete the command then press [ENTER].
F10	SCREEN 0,0,0	Sets Text Mode. Executed immediately.

Functions of FUNCTION KEYS



The key line is sometimes distracting when a program is running. Most programmers turn the key descriptions off by a command or within a program by a program statement.

Key descriptions may be turned off by the command:

KEY OFF [ENTER]

Example of turning key descriptions OFF within a program.

```

.
.
110 CLS
120 KEY OFF
.
.

```

Key descriptions may be turned back on by using the command:

KEY ON [ENTER]

Example of turning key descriptions ON within a program.

```

.
.
520 CLS
530 KEY ON
540 END

```



Turning KEY LINE On and Off

Turning the key descriptions back on at the end of a program will remind you of shortcut methods to LIST, SAVE, etc.

The descriptions provided by KEY ON display up to five characters. When displayed in the 40 characters/line mode, only the first five of the key descriptions are displayed.

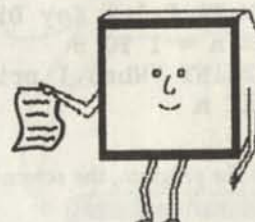
The KEY LIST Command

You may display a list of the complete description of all function keys by entering the command:

KEY LIST [ENTER]

Here is the result of using the KEY LIST command on the Tandy 1000 TX. Notice that no function has been assigned to the F11 or F12 key.

```
Ok
KEY LIST
F1 LIST
F2 RUN←
F3 LOAD"
F4 CONT←
F6 ,"LPT1:"←
F7 TRON←
F8 TROFF←
F9 KEY
F10 SCREEN 0,0,0
F11
F12
Ok
```



ASSIGNING FUNCTIONS TO THE KEYS

You can assign a function to any function key by using the KEY command.

KEY number, string

— a string with at most 15 characters (extras are ignored)

— the key number; 1 for F1, 2 for F2, 3 for F3, and so on.

For example, suppose you want to assign the string "CLS" to function key F11.

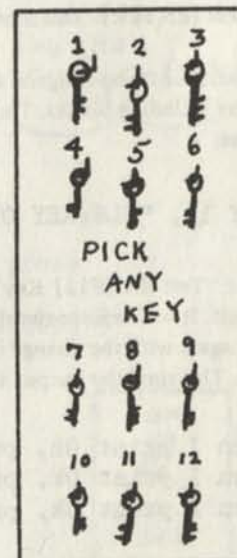
Type:

KEY 11, "CLS" [ENTER]

Now press F11 (or the function key you chose instead of F11).

The screen will look like this:

```
KEY 11, "CLS"
Ok
CLS_
```



Press the [ENTER] key, and *viola!*, the screen is cleared, except for the prompt (Ok) and the cursor (_).

Now that you have assigned CLS to the F11 key, enter and run the following GW-BASIC program that prints directions for clearing the screen.

EXAMPLE

```

1 REM ** Test the [F11] Key **
2 REM ** Browsing BASIC #1 **
3 REM ** Microsoft GW-BASIC File: KeyTest.001 **

100 REM ** Define Screen **
110 SCREEN 0: KEY OFF
120 WIDTH 80: CLS

200 REM ** Print Key Directions **
210 FOR n = 1 TO 3
220   PRINT "When I print Ok, press [F11]."

```

If your keyboard doesn't have F11, use F7 instead. If you do use F7, change all references in the program from F11 to F7.



After you run the program, the screen shows:

```

When I print Ok, press [F11].
When I print Ok, press [F11].
When I print Ok, press [F11].
Ok
_
```

Press F11 and the CLS command and cursor are added, as shown below:

```

When I print Ok, press [F11].
When I print Ok, press [F11].
When I print Ok, press [F11].
Ok
CLS _
```

Now, press [ENTER]. The screen is cleared.

Remember, the string assigned to a function key can have a maximum of 15 characters including blanks. Thus, the assignment to F11 could be longer, as follows:

KEY 11, "CLS:KEY ON"

'clear screen and turn key line on

Leave the "Test the [F11] Key" program in memory and enter this new key assignment. It will replace the shorter string assigned previously. Try the program again with the change in line 160. Press F11 at the end of the program. This time the output screen shows:

```

When I print Ok, press [F11].
When I print Ok, press [F11].
When I print Ok, press [F11].
Ok
CLS:KEY ON _
```



Press the ENTER key to clear the screen and turn the key descriptions back on. Notice again that only the first ten descriptions are displayed on line 25.

To see a description of all 12 keys type:

KEY LIST [ENTER]

Note that F11 now has a function assigned to it—the function that *you* assigned to it. This function will remain assigned to F11 until you change it, or exit BASIC, or turn the computer off.

If you choose a different function, your assignment would appear on this list, opposite the chosen key.

```
KEY LIST
F1 LIST
F2 RUN
F3 LOAD
F4 SAVE"
F5 CONT
F6 ,"LPT1:"
F7 TRON
F8 TROFF
F9 KEY
F10 SCREEN 0,0,0
F11 CLS:KEY ON
F12
```

*This is true
only if you
previously
assigned
CLS:KEY ON
TO F11*

You can use a function key as a short cut key in entering a program line. Assume that, as above, the string "CLS: KEY ON" has been assigned to F11.

Suppose you want to enter the line: 230 CLS: KEY ON.

DO THIS:	SEE THIS:
Type "230"	230_
Press F11	230 CLS:KEY ON_
Press ENTER	230 CLS:KEY ON —

Use this method to enter line 230 in the following short program.

```
1 REM ** Use F11 to Enter Program Line **
2 REM ** Browsing BASIC #1 **
3 REM ** Microsoft GW-BASIC File: KeyTest.002 **

100 REM ** Define Screen **
110 SCREEN 0: CLS: WIDTH 80: KEY OFF

200 REM ** Use F11 to Enter Line 230 **
210 PRINT "Press any key to continue"
220 ky$ = INPUT$(1) 'Wait for a key press

230 CLS: KEY ON 'Enter as: 230 [F11] [ENTER]

240 END
```

Run this program. It clears the screen, turns off the key line, and waits at line 220 with the screen shown at right.

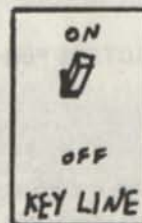
Press most any key and line 230 clears the screen and turns on the key line.

If you are writing a program in which some lines appear many times, use this method to save wear and tear on your fingertips!

Press any key to continue_

Ok
—

1LIST 2RUN 3LOAD...



*If you do not have
F11 and F12, use
any key
F1 through F10.*

ADDING A CHARACTER CODE

When a function was assigned to a function key in previous examples, it was necessary to press the ENTER key to activate the assignment. The ENTER key can be included by using its ASCII character code (13) in the assignment. The CHR\$ function is used to assign CLS:KEY ON [ENTER] to F12. CHR\$(13) is catenated (or concatenated if you prefer) with a plus (+) sign. It counts as only one of the 15 characters permitted in a key assignment.

You can make the assignment with the immediate command:

```
KEY 12,"CLS:KEY ON" + CHR$(13) [ENTER]
```

Now you can press F12 and get a clear screen with the function keys displayed at the bottom of the screen. Now you do not have to press ENTER after pressing F12.

You can use the F12 key to quickly clear the screen when it gets cluttered.

In addition to using such key assignments as immediate commands, you might also find situations where you could use an assignment within a program. You might want to change the function of a key while a program is running.

80 CHARACTERS PER LINE MODE: If F11 and F12 are defined as described earlier, the first ten functions are displayed as shown below:

```
1LIST 2RUN 3LOAD" 4SAVE" 5CONT 6,"LPT1 7TRON 8TROFF 9KEY 10SCREE
```

Press [CTRL] + [T]. You will see F11, F12, and F1 thru F8.

```
11CLS:K 12CLS:K 1LIST 2RUN 3LOAD" 4SAVE" 5CONT 6,"LPT1 7TRON 8TROFF
```

Press [CTRL] + [T] again. The key definitions disappear. Line 25 is blank.

Press [CTRL] + [T] once more. You will see the first ten definitions again.

```
1LIST 2RUN 3LOAD" 4SAVE" 5CONT 6,"LPT1 7TRON 8TROFF 9KEY 10SCREE
```

Using [CTRL] + [T] in the 80 characters per line mode causes key definitions to cycle, one step at a time, thru three displays:

- [1] The first ten definitions are displayed.
- [2] F11, F12, and F1 thru F8 are displayed.
- [3] No definitions are displayed.

40 CHARACTERS PER LINE MODE: You can change from the 80 characters per line mode to the 40 characters per line mode by typing **WIDTH 40** and pressing ENTER.

If the first ten definitions were showing at the bottom of the screen when **WIDTH 40** is typed, you will now see only five definitions at the bottom of the screen.

```
1LIST 2RUN 3LOAD" 4SAVE" 5CONT
```

F12
CLS:KEY ON
+
ENTER

Have the key
ENTER its
Function!

Remember, use
F1 through F10
if you don't
have F12



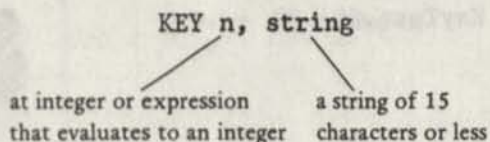
WIDTH 40
WIDTH 80

EXPERIMENT!

Press [CTRL] + [T] several times and watch what happens.

DISABLING THE FUNCTION KEYS

Not only can you change the functions performed by the keys F1 thru F10 (thru F12 for extended keyboards), but you may also disable these functions. We earlier said functions were assigned by:



If the string assigned has no characters (the null string, ""), the function key is disabled. This program disables keys F1 thru F10:

```

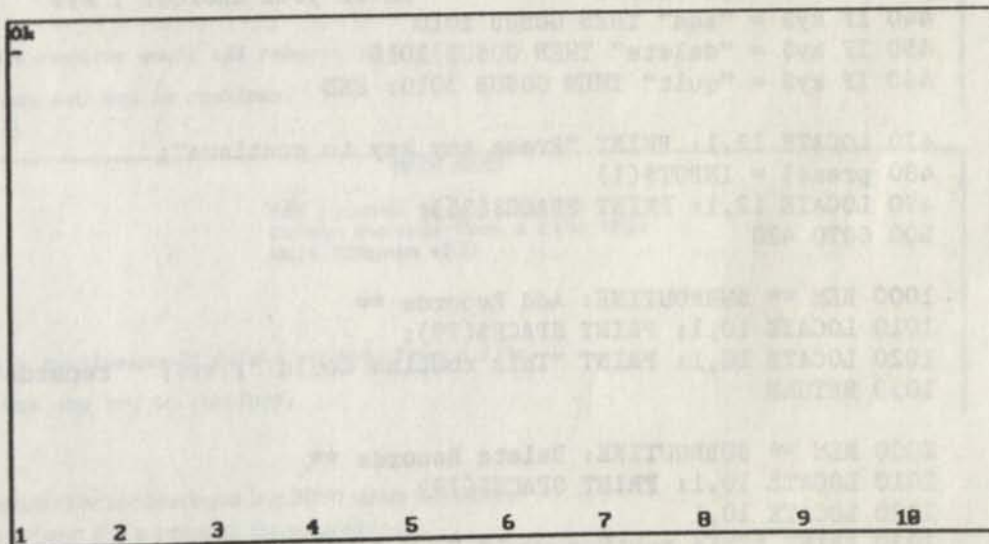
1 REM ** Disable Function Keys **
2 REM ** Browsing BASIC #1 **
3 REM ** Microsoft GW-BASIC File: KeyTest.003 **

100 REM ** Define Screen & Turn Keys On **
110 KEY ON: CLS

200 REM ** Erase Key Definitions **
210 FOR index = 1 TO 10
220   KEY index, ""
230 NEXT index
  
```

'function keys F1 thru F10
'assign null string

When this program is run, the screen is cleared and the function keys are disabled. All you see are the Ok prompt, the cursor, and the ten function numbers at the bottom of the screen—functions are disabled.

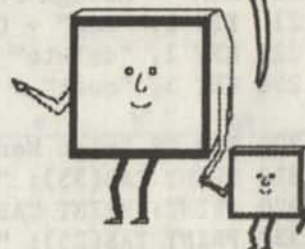


You now have ten function keys with which to play (twelve if you have an extended keyboard). The keys will remain disabled until you assign a new function or leave BASIC and return. The ten standard functions are automatically assigned on entering BASIC.

The following program uses three of the functions disabled by the Disable Function Keys program. Three data items (add, delete, and quit) are READ from a DATA statement and assigned to the function keys F1, F2, and F3.

A menu is printed from which you make a selection. Selections are made by pressing F1, F2, or F3.

Since "LPT1, TRON and TROFF are seldom used, we suggest redefining keys F6, F7, and F8 in most cases.



I did not actually write the subroutines which would enhance your data file. I leave that up to you—or will make it the subject of a future issue if you, our readers, are interested in data-file programming.

```
1 REM ** Function Key Assignments **
2 REM ** Browsing BASIC #1 **
3 REM ** Microsoft GW-BASIC File: KeyTest.004 **

100 REM ** Define Screen **
110 SCREEN 0: KEY OFF
120 WIDTH 80: CLS

200 REM ** Assign Functions to [F1], [F2], [F3] **
210 KEY 1, "add" + CHR$(13)
220 KEY 2, "delete" + CHR$(13)
230 KEY 3, "quit" + CHR$(13)

300 REM ** Print Menu **
310 PRINT TAB(35); "MAIN MENU"
320 PRINT: PRINT TAB(25); "Add records to a file (F1)"
330 PRINT TAB(25); "Delete records from a file (F2)"
340 PRINT TAB(25); "Quit Program (F3)"

400 REM ** Turn Keys On & Get Choice **
410 KEY ON
420 LOCATE 10,1: PRINT SPACE$(79);
430 LOCATE 10,1: INPUT "Enter your choice:", ky$
440 IF ky$ = "add" THEN GOSUB 1010
450 IF ky$ = "delete" THEN GOSUB 2010
460 IF ky$ = "quit" THEN GOSUB 3010: END

470 LOCATE 12,1: PRINT "Press any key to continue";
480 press$ = INPUT$(1)
490 LOCATE 12,1: PRINT SPACE$(25);
500 GOTO 420

1000 REM ** SUBROUTINE: Add Records **
1010 LOCATE 10,1: PRINT SPACE$(79);
1020 LOCATE 10,1: PRINT "This routine would "; ky$; " records to a file."
1030 RETURN

2000 REM ** SUBROUTINE: Delete Records **
2010 LOCATE 10,1: PRINT SPACE$(79)
2020 LOCATE 10,1
2030 PRINT "This routine would "; ky$; " records from a file."
2040 RETURN

3000 REM ** SUBROUTINE: Quit Program **
3010 CLS
3020 RETURN
```



A menu is printed by lines 310 thru 340.

Line 410 turns on the key functions at the bottom of the screen, and line 430 asks for a selection from the menu.

```

MAIN MENU
Add records to a file (F1)
Delete records from a file (F2)
Quit Program (F3)

Enter your choice: _

1add+ 2delete 3quit+ 4      5      6      7      8      9      10

```

When a key (F1, F2, or F3) is pressed, the appropriate subprogram is called by line 440, 450, or 460.

Press F1 and this message is printed:

The subprogram at line 1000 would be written to add records to a data file.

```

MAIN MENU
Add records to a file (F1)
Delete records from a file (F2)
Quit Program (F3)

This routine would add records to a file.
Press any key to continue.

```

Press F2 and this message is printed:

The subprogram at line 2000 would be written to delete records from a data file.

```

MAIN MENU
Add records to a file (F1)
Delete records from a file (F2)
Quit Program (F3)

This routine would delete records from a file.
Press any key to continue.

```

Press F3 to leave the program. The subroutine at line 3000 clears the screen. When the computer returns from this subroutine the program ends.

Remember, function keys F1, F2, and F3 still have the functions assigned by this program and the other keys are disabled. Type **SYSTEM** to access MS-DOS. When you return to GW-BASIC the normal functions will be assigned to keys F1 thru F10.

SYSTEM

OTHER SHORTCUTS FOR BASIC KEYWORDS

If you are lazy like I am, you may browse GW-BASIC for other shortcuts. Anything that reduces the amount of typing also reduces the chance for typing errors when entering programs or commands. For instance, ? can be entered as a shortcut for **PRINT** in GW-BASIC. Many times I have typed PRONT, PRINDT, or some other nonsensical variation. It is pretty hard to misspell a question mark.

Tucked away in the back of my Tandy 1000 TX Quick Reference Manual, I found the following list of GW-BASIC keywords that have shortcut keys when used with the ALT key. Here is a table of these shortcuts:

Keyword	Press	Keyword	Press
AUTO	[ALT] + [A]	NEXT	[ALT] + [N]
BSAVE	[ALT] + [B]	OPEN	[ALT] + [O]
COLOR	[ALT] + [C]	PRINT	[ALT] + [P]
DELETE	[ALT] + [D]	**	[ALT] + [Q]
ELSE	[ALT] + [E]	RUN	[ALT] + [R]
FOR	[ALT] + [F]	SCREEN	[ALT] + [S]
GOTO	[ALT] + [G]	THEN	[ALT] + [T]
HEXS	[ALT] + [H]	USING	[ALT] + [U]
INPUT	[ALT] + [I]	VAL	[ALT] + [V]
**	[ALT] + [J]	WIDTH	[ALT] + [W]
KEY	[ALT] + [K]	XOR	[ALT] + [X]
LOCATE	[ALT] + [L]	**	[ALT] + [Y]
MOTOR*	[ALT] + [M]	**	[ALT] + [Z]

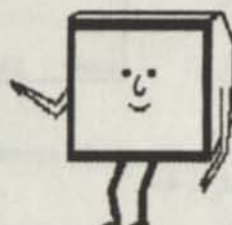
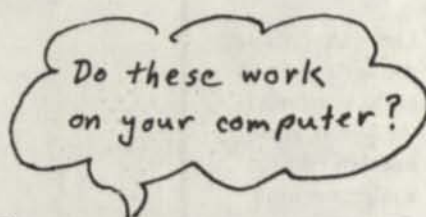
*MOTOR is a reserved keyword but is not recognized by this version of GW-BASIC.

**Sorry, I guess they couldn't find keywords beginning with J, Q, Y, or Z.

You can use these ALT key combinations to enter the keyword in a program statement or as a command when the cursor is blinking at you in anticipation of a command. Remember, [ALT] + [letter] means hold down the ALT key while you press the letter key.

Once again, if you are lazy like me, you will be tempted to use these shortcut keys. However, if your memory is as bad as mine, you will never remember which key represents which keyword. At least they were consistent in assigning keys that correspond to the first letter of the keyword.

We hope this initial GW-BASIC browsing has been helpful to you. You may have learned something new. You may have had your memory jogged about things you already knew. Or, maybe you are one of those "power programmers" who have been using these shortcuts all along. If so, be patient. We will browse a little deeper in future issues.



If you have suggestions for topics to be browsed, send them to:

The BASIC Teacher
"Browsing BASIC"
PO Box 7627
Menlo Park, CA 94026

By Bob Albrecht
and George Firedrake



ON MAY 1, 1964 at 4 a.m., John Kemeny and a student simultaneously entered and ran separate BASIC programs at Dartmouth College. Thus was born BASIC, the first computer language designed to be easy to learn and use by just about anyone.

In 1975 or thereabouts, thanks to Bill Gates, BASIC became available on the first personal computers. In one giant step and a series of short hops, computers moved from the cloistered realm of the professional programmer into the hands of enthusiastic amateurs, who collectively created a new form of programming called "spaghetti code." BASIC programs were a tangled skein, but the sauce was heady . . .

In those days of yore, memory was dear, BASIC was primitive, and programs were crunched into the smallest possible space—unreadable by anyone not a true believer. Fortunately, things got better. Computers got better, memories got bigger, BASIC got better and better. Now there is

QuickBASIC 4.0, the best BASIC yet. It is easy to learn, easy to use, and very, very capable. You can learn to write programs in QuickBASIC, programs that tell the computer to do what *you* want it to do the way *you* want it done.

QuickBASIC 4 ... the Package

QuickBASIC 4 comes in an impressive package containing three books and three 5¼" disks or two 3½" disks. The disks are not copy-protected; you can make copies for your own use. **DO IT!** Make copies of the original disks, then put the original disks in a safe place away from the computer, perhaps an inconvenient place so you are not tempted to use them, except for making copies. Use the original disks that come in the package only for making copies.



The three QuickBASIC books contain an enormous amount of information. QB4 is succinctly described in 1241 pages of documentation—obviously not written for beginners. That is why we are writing this stuff in *The BASIC Teacher*—to help you begin to learn QuickBASIC. Later, you will appreciate the Microsoft books more and more, as your knowledge of QuickBASIC grows.

If you are a beginner, start with the Microsoft book called *Learning and Using Microsoft QuickBASIC*. Page 7 encourages you to make copies of the disks. Do it! Pages 8 and 9 briefly describe the files on the three disks. Relax. The only files you need to get started are the QB.EXE file and the QB.HLP file. Ignore pages 9 thru 15. We will use a much simpler way to begin.



Read pages 19 thru 27, skip 28 thru 32, read pages 33 thru section 2.2.4 on page 38. Now go to Chapter 3. The table of contents is on page 63. This chapter will help you get started.

What to Do After Opening the Box

First, immediately, before you do anything else:

- Protect all disks from catastrophes such as accidental erasure. If you are using 5¼" disks, put tape over the write-protect notch; if you are using 3½" disks, slide the write-protection switch to the "open" position so so you can see through the little rectangular hole.

After protecting all the original disks from catastrophes such as accidental erasure, do the following things:

- Make copies of all the disks. You can use the DOS command called **DISKCOPY** to do this. Are we correct in assuming you know a dollop of DOS?
- Make at least two copies of the original disks. More are better.
- Put the original disks in a safe place away from the computer, perhaps an inconvenient place so you are not tempted to use the original disks, except for making copies. Use the original disks that came with the package only to make copies.
- Carefully label the copies. On the label, copy the essential information from the original disk which, of course, you had already protected against accidental erasure (you did, didn't you?). Include the date and anything else you feel is relevant, useful, or inspiring.
- Make a directory for each copy. You can use the MS-DOS command called **DIR** to do this.
- If you have a printer, print the directory for each disk and put a copy with the disk. The easiest way to do this is to print the directory to the screen (use the **DIR** command), then use the **Print Screen** feature to print the screen to the printer (just hold down the **SHIFT** key and press the **PRINT** key).

- If you do not have a printer, take pencil or pen in hand and copy the information off the screen. Keep a directory with each disk.

You now have the original disks in a safe place and two or more sets of back-up disks, all properly labeled. Each disk is accompanied by a directory so you can quickly find the right disk.

Make Some QuickBASIC Work Disks

For ease in learning QuickBASIC, make several QuickBASIC Work Disks.

- Use DOS to **FORMAT** several disks. Use the **/S** option in order to make your QB Work Disks **self-booting**. On a single drive system, you can do it by typing **FORMAT A: /S**. This formats the disk and also copies certain files from DOS to the formatted disk. You will be able to use this disk alone, without having to first load DOS.
- Copy two files from the QuickBASIC disks. Use the DOS command called **COPY** to do this. We trust you know a dollop of DOS. If you have trouble, send us a **self-addressed stamped envelope** and we will try to help. Copy the two files from the copies you made of the original QuickBASIC disks. You *did* make copies, didn't you? Copy only the following two files: **QB.EXE** and **QB.HLP**.

That's it. You now have a QB Work Disk. Make several copies of this QB Work Disk. If you use the DOS command **DIR** to see the directory on the disk, you should see it has the following files:

COMMAND.COM	From DOS
QB.EXE	From QB
QB.HLP	From QB

Now using QuickBASIC is easy! Just put the QB Work Disk in disk drive A and turn on the computer. When you see the familiar A-prompt, type **QB** and press the **ENTER** key.

Next time, we'll do it together and take you **Window Shopping**.



No. 2, July 1988

\$3.00

*For beginning
programmers with no prior
programming experience*

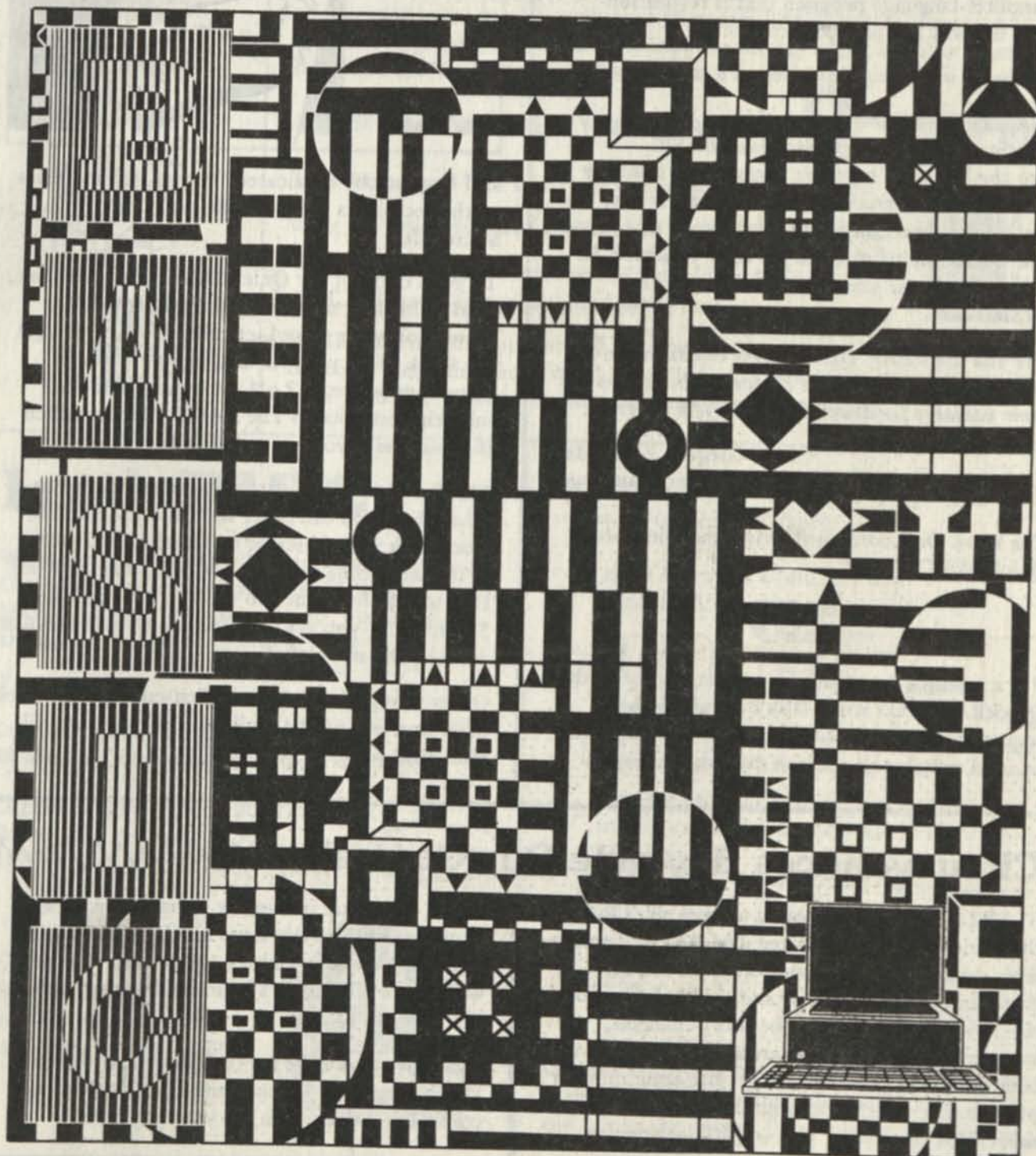
CONTENTS

- 3 Teach Yourself BASIC
- 9 Browsing BASIC
- 19 Teach Yourself
QuickBASIC

PUBLISHER'S STATEMENT: *The BASIC Teacher* is published monthly by Different Worlds Publications, 2814 - 19th Street, San Francisco, CA 94110. Contents copyright © 1988 by Different Worlds Publications. All rights reserved. Contents may be copied and distributed freely. Address all correspondences to *The BASIC Teacher*, 2814 - 19th Street, San Francisco, CA 94110.

SUBSCRIPTION INFO: A 12-issue sub in the U.S. and Canada is \$36. Overseas subs are \$42 by surface mail, \$54 by air.

PRINTED IN THE U.S.A.



GOOD NEWS FOR TEACHERS!

MICROSOFT CORPORATION has authorized *The BASIC Teacher* to make available to educators a limited quantity of *QuickBASIC 4.0 Work Disks*. This is the least-expensive way for teachers to get a copy of the exciting new computer-language program that is revolutionizing the way people use computers.

Microsoft is making this offer on a trial basis:

"Bob, as we have discussed, this is a trial of exciting possibility—getting QuickBASIC into the hands of teachers. Please keep in touch as this program proceeds. After the users have had these disks for a couple weeks, I would like to talk with some of them to get their feedback on the product, and your approach to teaching BASIC."

"As you are aware, QB4 is being enhanced to be as easy to use as possible. I hope we both gain some valuable feedback from this test program."

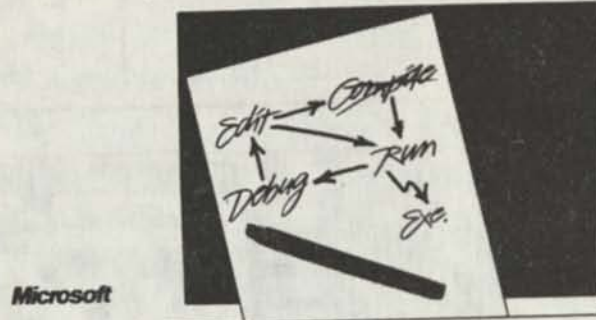
—Gregory E. Lobdell
Product Manager, Languages

The Work Disk contains the two most-important QuickBASIC files:

QB.EXE
QB.HLP

For a nominal fee, teachers can get a copy of the QuickBASIC 4.0 Work Disk by making the request on their school letterhead. The disks may be used only by the person making the request

Microsoft QuickBASIC 4.0



and may not be duplicated. Names and addresses of the recipients will be made available to Microsoft.

To get a copy of the QuickBASIC 4.0 Work Disk, send a check or money for \$12 along with the request on your school letterhead. When you get the Work Disk, you will also receive a special coupon good for \$12 off the regular \$36 subscription price to *The BASIC Teacher* which you can use if you are not already a subscriber.

Also for those educators who are not already a subscriber, you can order the QuickBASIC 4.0 Work Disk and subscribe to *The BASIC Teacher* at the same time for only \$30—\$12 for the Work Disk and \$18 for the sub (50% off the regular \$36 price!)—you get both for less than the regular price of the sub alone!

Order now, this is a limited first-come, first-served offer and will not be available after all the available disks have been spoken for.

Circumstances Beyond Our Control

STARTING A new endeavor always takes longer than originally expected and *The BASIC Teacher* is no exception. This issue comes to you later than our original schedule and we apologize for that. Subscribers will not be short-changed, however, as subscriptions are sold on an issue-by-issue basis and you will still get the same number of issues for which you originally paid, i.e., if you subscribed for a year, you will still get twelve

issues. Issues are in number order and your sub expires with a certain issue number (see mailing label). If you are a charter subscriber, your sub will expire with issue 12 (presently scheduled as the May 1989 issue).

Again, we apologize for the lateness of this issue. Please forgive us for circumstances beyond our control and thank you for your patience.



INTRODUCTION



IF YOU can read a newspaper or a comic book, you can learn to read and understand programs written in BASIC, the People's Computer Language. BASIC has a small **vocabulary** and a simple **syntax** (grammar). Last time, you learned some special words that Microsoft BASIC knows. They are called **keywords** or **reserved words**. Here are the keywords introduced and described last time:

BEEP DATES\$ TIMES\$ TIMER PRINT



You also learned how to use the computer to do arithmetic, using the four arithmetic operations: $+$ $-$ $*$ $/$

You know how to clear the screen by holding down the **CTRL** key and pressing the **L** key.



You have probably encountered the **syntax error** message. This simply means the computer didn't understand you. That's OK, just try again—and learn more BASIC keywords that the computer does understand.

Load BASIC

WE ASSUME you know how to load the Disk Operating System (DOS) and BASIC. Do so now. If DOS asks for the date and time, please enter them.

For example:

Current date is Tue 1-01-1980
Enter new date (mm-dd-yy): 4-27-88

Current time is 0:00:27.90
Enter new time: 13:30

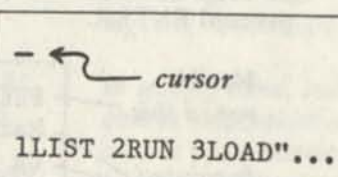
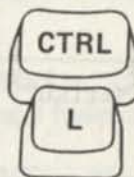
1:30 p.m.

Later we'll show you how to set the date and time from within BASIC.



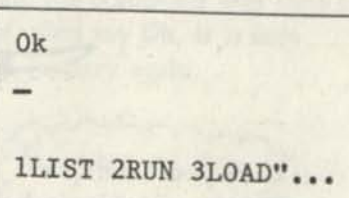
WE LIKE to begin with a clean screen. Here are two ways to clear the screen:

- Hold down the **CTRL** key and press the **L** key.



You see only the blinking cursor at the top of the screen and the key line at the bottom of the screen.

- Type **CLS** and press the **ENTER** key.



CLS is a BASIC keyword that tells the computer to clear the screen. After doing so, it says "Ok" and blinks the cursor. Your turn to do something.

KEY OFF, KEY ON

THE BOTTOM of the screen is the **key line**, which shows labels for the **function keys**. On our keyboard, these keys are labeled F1, F2, F3, and so on across the top of the keyboard. On your computer they may be elsewhere.

You can turn the key line off and on:

- To turn off the key line, type **KEY OFF** and press ENTER.
- To turn on the key line, type **KEY ON** and press ENTER.



Go ahead . . . do it a few times. If you misspell **KEY OFF** or **KEY ON**, you will see a **syntax error**, followed by the friendly **Ok** and the blinking cursor. Try again.

PRINT

YOU CAN tell the computer to **PRINT** your name by typing the keyword **PRINT** followed by your name enclosed in quotation marks ("").

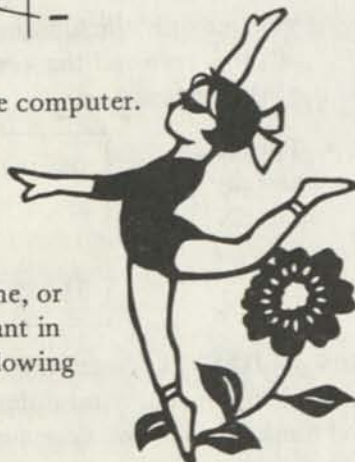
Mariko will demonstrate:

- Mariko typed **PRINT "MARIKO"** and pressed ENTER.

Mariko typed this — **PRINT "Mariko"**
 It printed this — **Mariko**
 — **Ok**
 —

By "it," we mean the computer.

Try it with your name, or put anything you want in quotation marks following the word **PRINT**.

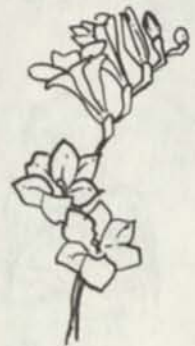


DATE\$ =

IF YOU forgot to set the date when you loaded DOS, you can do it now in BASIC. We are writing this on April 27, 1988. So we'll set the date to today, then verify it by printing the date on the screen:

- First clear the screen.
- Type **DATE\$ = "4-27-88"** and press ENTER.

```
DATE$ = "4-27-88"
Ok
PRINT DATE$
04-27-1988
Ok
—
```



You can enter the date in several ways.

Some are shown below:

DATE\$ = "4/27/88"

DATE\$ = "4-27-1988"

DATE\$ = "4/27/1988"



And here are some ways to make mistakes:

DATE\$ = "1-32-88"
Illegal function call

Oops! Too many days in that month.

DATE\$ = "13-27-88"
Illegal function call

And too many months in that year.

DATE\$ = "4 27 88"
Illegal function call

Needs slashes (/) or dashes (-).

DATE\$ = 4/27/88
Type mismatch

Forgot the quotation marks ("").

DATE = "4-27-88"

Forgot the \$.

We tried to set the date to January 1, 1970 (**DATE\$ = "1/1/70"**), but got an **Illegal function call** message. Then we tried January 1, 1980 (**DATE\$ = "1-1-1980"**). That was OK.

What is the earliest date allowed on your computer? The latest date?

TIME\$ = " " "

OUR COMPUTER uses a 24-hour clock. So 6:00 p.m. is 18:00:00. In this system, the time is measured from midnight, which is 00:00:00.



What time is it?

(No, it's not 5 until 7.)



12-Hour Clock	24-Hour Clock	Comments
00:00:01 AM	00:00:01	One second after Midnight
00:01:00 AM	00:01:00	One minute after Midnight
01:00:00 AM	01:00:00	One hour after Midnight
06:00:00 AM	06:00:00	Time to get up
09:00:00 AM	09:00:00	Many companies begin work
10:30:00 AM	10:30:00	Coffee break?
12:00:00 N	12:00:00	High noon
3:00:00 PM	15:00:00	School's out!
7:00:00 PM	19:00:00	Dinner?
11:59:59 PM	23:59:59	One second before Midnight

Let's set the time and **PRINT** it. First, set the time to midnight:

- Type **TIME\$ = "00:00:00"** and press ENTER.
- Type **PRINT TIME\$** and press ENTER.

```
TIME$ = "00:00:00"
Ok
PRINT TIME$
00:00:05
Ok
-
```



As you can see, it took us five seconds to type **PRINT TIME\$** and press ENTER.

You can set the time to midnight by typing **TIME\$ = "0:0:0"** or even by typing **TIME\$ = "0"**. If you omit the seconds or minutes, the computer will automatically supply zeroes.



Go ahead, try a few. If you don't make any mistakes, try some of ours:

- | | |
|---|--|
| TIME = "12:00:00"
Type mismatch | <i>Forgot the dollar sign in TIME\$.</i> |
| TIME\$ = 10:23:37
Type mismatch | <i>Forgot the quotation marks.</i> |
| TIME\$ = "12:60:00"
Illegal function call | <i>Oops!
Too many minutes.</i> |
| TIME\$ = "24:00:01"
Illegal function call | <i>Too many hours. A second after midnight is "00:00:01".</i> |
| TIME\$ = "12 - 0 - 0"
Illegal function call | <i>In setting time, use colons (:) instead of dashes (-) or slashes (/) to separate the hours, minutes, and seconds.</i> |
| TIME\$ = "15/30/00"
Illegal function call | |

If you make a mistake, the computer will type a cryptic error message, then say **Ok**. It is very patient and forgiving! Just try again.



You sure are easy to get along with!

Number Patterns

MATHEMATICS IS a rich world of **patterns**. You can use BASIC to explore this world. Try the following number pattern. Remember to press ENTER after you type something.

- You type `PRINT 11 * 11`
It prints `121`
- You type `PRINT 111 * 111`
It prints `12321`
- You type `PRINT 1111 * 1111`
It prints `1234321`
- You type `PRINT 11111 * 11111`
It prints `1.234543E+08`

Usually the computer will print up to seven digits. If a number is too big to fit into seven digits, it is printed as a **floating point number** like the one shown above. We'll tell you more about floating point numbers later.

But first, here is a way to get the computer to compute and print bigger numbers, up to 16 digits. To do this, put a number sign (#) at the right end of one of the numbers.

- You type `PRINT 11111 * 11111#`
It prints `123454321`
- You type `PRINT 111111 * 111111#`
It prints `12345654321`
- You type `PRINT 11111111 * 11111111#`
It prints `123456787654321`
- You type `PRINT 111111111 * 111111111#`
It prints `1.234567898765432D+16`

Oops! The last answer was too big to fit into 16 digits, so the computer printed it as a double-sized floating point number. What do you think is the exact answer? Will the pattern continue?

Powers of Numbers



BASIC CAN do yet another arithmetic operation.

It can compute a **power of a number**.

For example, $5^2 = 5 \times 5 = 25$ is "5 to the second power" or "5 squared."

To compute a power of a number, use the \wedge symbol.

To type \wedge hold down SHIFT and press



- You type `PRINT 5^2`

It prints `25`

Next, compute 5 to the third power, also called 5 cubed:

- You type `PRINT 5^3`

It prints `125`

Of course, you can also use multiplication (*) to compute a power of a number:

- You type `PRINT 5 * 5`

It prints `25`

- You type `PRINT 5 * 5 * 5`

It prints `125`

Here are some more examples of powers of numbers:

- You type `PRINT 2^5`

It prints `32`

- You type `PRINT 10^6`

It prints `1000000`

- You type `PRINT 10^9`

It prints `1E+09`



The Mysterious K

COMPUTERS USE a very simple code, called **binary**, to represent information. Binary is very simple; it uses only two symbols, 0 and 1. The symbols, 0 and 1, are called **binary digits**, or **bits**.

In a typical personal computer, information is stored in the **memory** of the computer. The memory consists of many thousands of bits organized as bunches of bits in **memory locations**.

One memory location can hold eight bits of information. A bunch of eight bits is called a **byte**. So . . . one memory location can hold eight bits, or one byte. The memory of a typical personal computer has many thousands of memory locations.

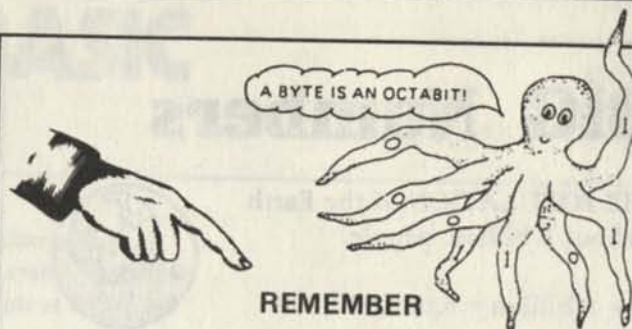
- One memory location can store eight bits.
- A group of eight bits is called a byte.
- So, a memory location can store one byte.
- A computer memory has many thousands of locations. So the memory can store many thousands of bytes.

Perhaps you have heard about the mysterious **K**. People say a computer has 128K or 256K or 512K—or more—bytes of memory.

- 1K bytes equals 2^{10} bytes equals 1024 bytes.

Use the computer to change 1K bytes or 256K bytes or 512K bytes to ordinary numbers.

- You type `PRINT 2^10`
It prints **1024**
- You type `PRINT 256 * 2^10`
It prints **262144**
- You type `PRINT 512 * 2^10`
It prints **524288**



$$1\text{K bytes} = 2^{10} \text{ bytes} = 1024 \text{ bytes}$$



Perhaps you have heard the ancient story about the wise person who did a great service for a king. The king asked her what reward would be appropriate. Her request was simple. She asked only for grains of wheat, computed as follows:

On the first square of a chessboard, one grain of wheat. On the second square, two grains of wheat. On the third square, four grains of wheat. And so on, doubling at each new square.

On square number **n**, there are to be 2^{n-1} grains. Let's find out how many grains on square 16:

- You type `PRINT 2^15`

It prints **32768**

Inexorably, the grains pile up. How many on square 64?

- You type `PRINT 2^63`

It prints **9.223372E+18**

Yup, that's a lot of wheat, more wheat than existed in all the kingdoms everywhere. The king realized that he had been duped.

The king was: a) chagrined b) overjoyed
c) amused d) befuddled
e) angry f) livid
g) _____ Your choice.

Please pick one of the above and write the end of the story.

THE POPULATION of the Earth is about 5 billion people.



- 5 billion = 5,000,000,000

Tell the computer to print that very large number on the screen:

- You type `PRINT 5000000000` ←

It prints **5E+09**

*No commas
please!*

The computer printed this very large number as a floating point number.

Read it like this:

- 5E+09 is five times ten to the ninth power.

In math notation, we write it like this:

5×10^9

Floating point notation is simply a shorthand way of expressing very big numbers. In floating point notation, a number is represented by a **mantissa** and an **exponent**. The mantissa and exponent are separated by the letter **E**.

$$\begin{array}{ccc} 5 & E & + 09 \\ \downarrow & \downarrow & \downarrow \\ \text{mantissa} & E & \text{exponent} \end{array}$$

Here is another BIG number in good old everyday notation and also in floating point notation:

One trillion. We usually write it like this:

1,000,000,000,000

- You type `PRINT 10000000000000`

It prints **1E+12**

- You type `PRINT 10^12`

It prints $1E+12$

NO COMMAS PLEASE!

When you type big numbers, do not use commas, as you usually do in writing numbers. BASIC does not understand this use of commas. Commas have a very special use in PRINT statements. Please be patient. We will get to it later.

WELL, AS we have seen, BASIC does a good job on very big numbers. It is equally adept with very small numbers.

Recently, we have had occasion to chase several snails. We became curious about snail speed. The results of our first experiment indicate that the speed of a frightened snail is about 0.0000079 miles per second.

- The speed of a snail is about 0.0000079 miles per second.

- You type `PRINT 0.0000079`

It prints 7.09E-06



BASIC printed this very small number as a **mantissa** and an **exponent**, separated by the letter E.

7.09E-06
mantissa exponent

The exponent
is negative.

Read it like this: *Seven point zero nine times ten to the minus six*. In math, science, or other hi-tech books, you might see this number written as 7.09×10^{-6} .

UNIVERSE STUFF

Hydrogen is universal stuff. It began with the big bang that created the universe. It is here, there, everywhere. The hydrogen atom is very small and very light.

- The mass of the hydrogen atom is about 1.67×10^{-27} kilograms.

- You type **PRINT 1.67E-27**

It prints $1.67\text{E-}27$

Yes, you can enter numbers in **Floating point notation**. Saves time and finger fatigue:

[illegible]

NEXT TIME: *Tiny Programs.*

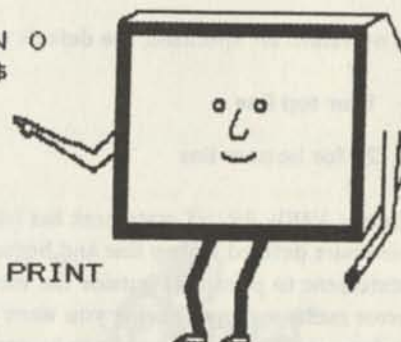
By Don Inman

LAST TIME, we browsed the **KEY** command and used it to change the functions assigned to keys F1 thru F10 (also F11 and F12 if available). In this issue, we'll use what we learned about function-key assignments as we browse **viewports** and other **windowing** techniques.

We assume you have added the **KEY** statement to the list of BASIC keywords we started with. We also used **SCREEN 0** to define the text screen and **STRING\$** to define a string of ASCII-code characters. Here is the list of BASIC keywords you should now know.

CHR\$	GOSUB	LOAD	RUN
CLS	GOTO	LOCATE	SAVE
DATA	IF...THEN	PRINT	SCREEN 0
DIM	INPUT\$	READ	SPACE\$
END	KEY	REM	TAB
FOR...NEXT	LIST	RETURN	WIDTH

These words
are the
key!



In this issue we will add these keywords to the list:

COLOR	SCREEN(row, column)	VIEW PRINT
LINE INPUT	STRING\$	

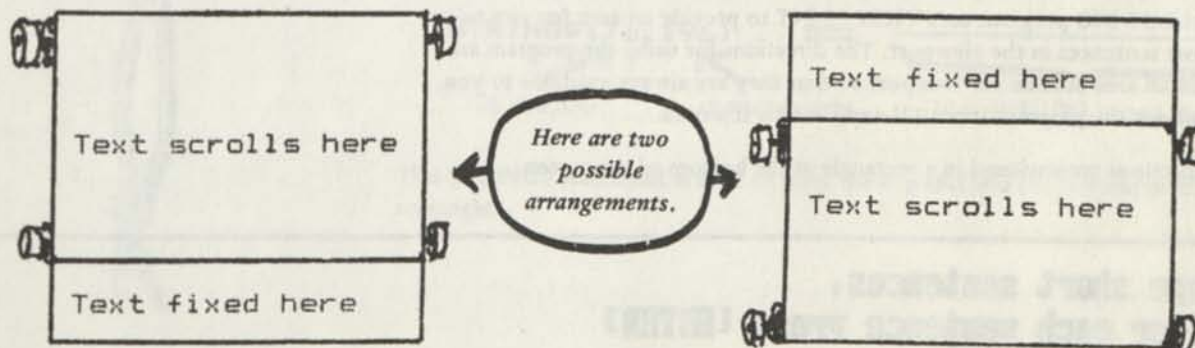
We will also use the logical operator **OR**.

SIZING YOUR CANVAS

WE WILL limit the discussion to a text screen that is 80 columns wide and 24 lines high (25 if you turn off the key line and use line 25). The methods used here can also be applied to text screens of other sizes.

When a screen becomes full, text scrolls upward—the top line moves off the screen as a new line is entered at the bottom. This poses some problems when you want some specific text to remain on the screen.

It is possible to block off an area of the screen so that you can keep the text in one area of the screen fixed while the text in another part of the screen scrolls.



This technique is useful when you want to keep a menu or some instructions for the use of a program on the screen while the program input and/or output is in an area where the screen scrolls. The text screen can be partitioned in this way by a **VIEW PRINT** statement.

VIEW PRINT

A **VIEW PRINT** statement specifies a **top line** and a **bottom line**. These two values (ranging from 1 to 24) determine where the scrolling area of the screen is located.

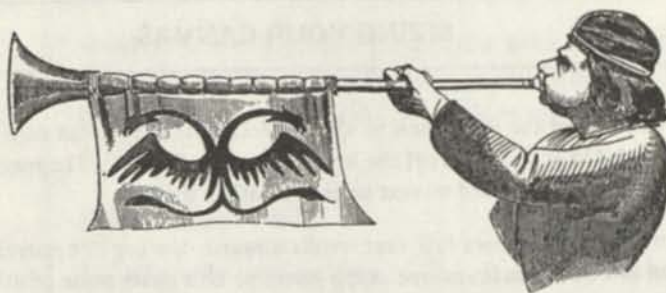
VIEW PRINT [top line TO bottom line]

If no values are specified, the default values used are:

1 for top line

24 for bottom line

Once a **VIEW PRINT** statement has been executed, all text will occur in the viewport defined by top line and bottom line. If you try to use a **LOCATE** statement to print text outside the viewport, you will get an **illegal function call** error message. Any printing you want outside the viewport should be done before you define a viewport or by temporarily defining a new viewport. The last **VIEW PRINT** statement executed controls the viewport used for printing text.



Using VIEW PRINT

THE FOLLOWING program uses **VIEW PRINT** to provide an area for you to enter short sentences in the viewport. The directions for using the program are printed in an area outside the viewport so that they are always available to you. Enter and run the program—then I'll explain how it works.

The instructions are enclosed in a rectangle at the bottom of the screen.

Type short sentences.
After each sentence press [ENTER]
Press Q to quit.


```

1 REM ** Viewport Demonstration **
2 REM ** Browsing BASIC #2 2/1/88 **
3 REM ** Microsoft GW-BASIC File: Viewprt.001 **

```

```

100 REM ** Define Screen **
110 SCREEN 0: CLS : KEY OFF: WIDTH 80

```

```

200 REM ** Define Instruction Strings **
210 Text$(1) = CHR$(218) + STRING$(76, 196) + CHR$(191)
220 Text$(2) = CHR$(179) + " Type short sentences." + SPACE$(54) + CHR$(179)
230 Text$(3) = CHR$(179) + " After each sentence press [ENTER]" + SPACE$(42) +
  CHR$(179)
240 Text$(4) = CHR$(179) + " Press Q to quit." + SPACE$(59) + CHR$(179)
250 Text$(5) = CHR$(192) + STRING$(76, 196) + CHR$(217)

```

```

300 REM ** Print Instructions **
310 FOR row = 1 TO 5
320   LOCATE row + 19, 2: PRINT Text$(row);
330 NEXT row

```

```

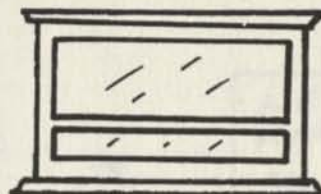
400 REM ** Print in Viewport **
410 VIEW PRINT 1 TO 19
420 FOR TypeIt = 1 TO 50
430   LOCATE 19, 1: LINE INPUT Sentence$
440   IF Sentence$ = "q" OR Sentence$ = "Q" THEN TypeIt = 50
450 NEXT TypeIt

```

```

500 REM ** Restore Edit Screen **
510 VIEW PRINT
520 CLS : KEY ON
530 END

```



The instructions for the program's use are assigned as elements of a string array (Text\$(n)). An array was used so that the text can easily be placed where desired on the screen.

Included in the text are ASCII character-codes that provide an outline for the rectangular area that contains the instructions. To draw the top and bottom of the rectangle, a single shape (CHR\$(196)) is used consecutively many times. Rather than print individual codes, you can "string" them all together with the **STRING\$** function.

For example—to string together 76 consecutive CHR\$(196) characters:

$\text{STRING\$}(76, 196)$ =

76 in a row character code 76 CHR\$(196) joined together

This STRING\$ statement is used to form the top and also the bottom of the rectangle.

Top corners are formed by:

┌ CHR\$(218) ┐ CHR\$(191)

Sides are formed by:

| CHR\$(179) | CHR\$(179)

Bottom corners are formed by:

└ CHR\$(192) ┘ CHR\$(217)

The elements of the Text\$ array are placed by the **FOR...NEXT** loop.

```
310 FOR row = 1 TO 5
320   LOCATE row + 19, 2: PRINT Text$(row);
330 NEXT row
```

Fixed area in rows
20 (1 + 19) thru
24 (5 + 19).

Text\$(1) is printed on line 20.

Text\$(2) is printed on line 21.

Text\$(3) is printed on line 22.

Text\$(4) is printed on line 23.

Text\$(5) is printed on line 24.

Line 410 contains the **VIEW PRINT** statement that restricts the scrolling area to lines 1 to 19.

```
410 VIEW PRINT 1 TO 19
```

Top line = 1; bottom line = 19.
LEAVE INSTRUCTIONS ALONE!

The **FOR...NEXT** loop (lines 420 to 450) provides an opportunity for you to carry out the program's instructions. The upper bound of the loop is set for 50 sentences which will be more than enough to make the text scroll. In fact, you may get tired of typing long before the loop reaches 50. The **FOR...NEXT** loop contains a way for you to escape. Line 440 tells sets the control variable (TypeIt) to 50 if you type the letter q as a sentence—**Q [ENTER]**. When you type **Q [ENTER]** as a sentence, the computer will think you have typed 50 sentences and will exit the loop.

Sometimes it's OK to fool the computer!



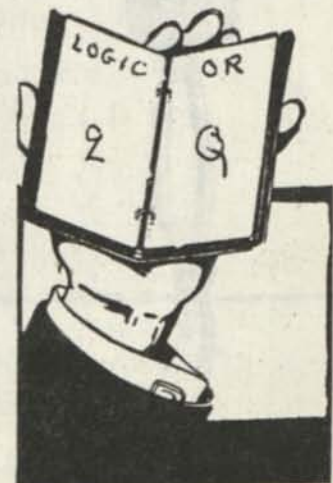
```
420 FOR TypeIt = 1 TO 50
430   LOCATE 19, 1: LINE INPUT Sentence$
440   IF Sentence$ = "q" OR Sentence$ = "Q" THEN TypeIt = 50
450 NEXT TypeIt
```

Text is entered at row 19 (see line 430). When the **ENTER** key is pressed, this text line is scrolled up one row making room for a new line at row 19. The **LINE INPUT** statement accepts an entire line of input—up to the carriage return caused by pressing **ENTER**. The input is assigned to Sentence\$.

The **OR** keyword used in line 440 is called a **logical operator**. If either of the conditions **Sentence\$ = "q"** OR **Sentence\$ = "Q"** is true, then the value of TypeIt will be set to 50 even if you haven't typed 50 sentences.

The **INSTR** function could be used in line 440 as follows:

```
440 IF INSTR("qq", Sentence$) <> 0 THEN TypeIt = 50
```

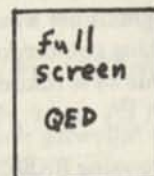


Notice that **VIEW PRINT** is used again in line 510. This time, the top line and the bottom line are not given. Therefore, the default values (top line = 1 and bottom line = 24) are used—this frees the entire screen for the statements that conclude the program.

```
510 VIEW PRINT
520 CLS: KEY ON
530 END
```



to



The screen shot below shows a full page of text. The cursor is at the beginning of line 19. When the next line of text is entered, the top line will scroll off the screen. All other text lines in the viewport will move up one row.

This is the first sentence I typed.
This is the second.

I skipped a line.

I skipped two lines.

Now three.
I'n getting lazy.

Now the first sentence is getting near the top.

Type short sentences.
After each sentence, press [ENTER].
Press Q to quit.

Cursor on line 19
is waiting for a
new Sentence\$.

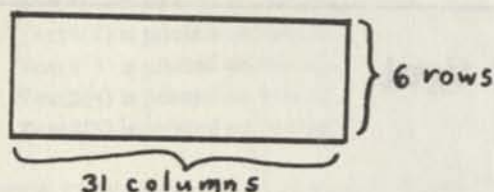


Looking Through a Window

YOU HAVE probably noticed that many of the new commercial software programs use windows that overlap the output screen to provide a means of making choices for forthcoming actions. The problem in creating your own windows is restoring the original screen when the pull-down window is removed.

The following program uses the key assignment techniques described in "Browsing BASIC No. 1." The functions of keys F1 (LIST), F3 (LOAD), and F4 (SAVE) are slightly altered to provide use from within a program. These keys are then used to make appropriate selections from a pull-down menu window.

An area of 6 rows by 31 columns is used to display the pull-down window—but the characters that will be covered by the window must be preserved.



The characters to be overlapped by the menu window are saved as ASCII characters in six strings, one string for each row of characters that will be covered by the window. When these characters have been safely stored, the window is pulled down. Choices from the menu window are:

Press F1 to LIST the current program

Press F3 to LOAD a new program

Press F4 to SAVE the current program

Press ESC to leave the current program



Each of the keys F1, F3, F4, and ESC are active while the menu is displayed.

- If you press F1, the window is removed, the original characters are replaced, and the command: **LIST**— is displayed. Press the ENTER key, and the program is listed.
- If you press F3, the window is removed, the original characters are replaced, and the command: **LOAD**— is displayed. Type the file name of the file you wish to load, press ENTER, and the new file is loaded.
- If you press F4, the window is removed, the original characters are replaced, and the command: **SAVE**— is displayed. Type the file name to be used to save this program, press ENTER, and the current program is saved under that name.
- If you press ESC, the window is removed, the original characters are replaced, the BASIC prompt (Ok) is displayed with the cursor below it.

ENTER and run the program.




```

1 REM ** Window Demonstration **
2 REM ** Browsing BASIC #2 2/2/88 **
3 REM ** Microsoft GW-BASIC File: Window.001 **

```

```

100 REM ** Initialize **
110 DIM Block%(6, 31)
120 KEY 1, " LIST"
130 KEY 4, " SAVE" + CHR$(34)
140 KEY 3, " LOAD" + CHR$(34)
150 CLS

```

Assign functions
to F1, F4, and F3

```

200 REM ** Print Messages on Screen **
210 PRINT "This program demonstrates the use of windows."
220 PRINT "When you press a key, a window is pulled down"
230 PRINT "This message."
240 LOCATE 5,15: PRINT "PRESS A KEY NOW";
250 Ky$ = INPUT$(1)

```

Original
message

```

300 REM ** Use the Window **
310 GOSUB 1010
320 GOSUB 2010
330 GOSUB 3010

```

```

400 REM ** End of Program **
410 END

```

Copy original!

```

1000 REM ** SUBROUTINE: Copy Screen **
1010 LOCATE 1,19,0
1020 FOR row = 1 TO 6
1030   FOR column = 1 TO 31
1040     Block%(row, column) = SCREEN(row, column + 18)
1050   NEXT column
1060 NEXT row
1070 RETURN

```

Print the
window

```

2000 REM ** SUBROUTINE: Print Window **
2010 LOCATE 1,19: COLOR 4,3: PRINT SPACE$(31);
2020 LOCATE 2,19: PRINT " To List program      Press F1 ";
2030 LOCATE 3,19: PRINT " To SAVE program      Press F4 ";
2040 LOCATE 4,19: PRINT " To LOAD program      Press F3 ";
2050 LOCATE 5,19: PRINT " To leave program     Press ESC ";
2060 LOCATE 6,19: PRINT SPACE$(31);
2070 RETURN

```

```

3000 REM ** SUBROUTINE: Replace Original Screen **
3010 COLOR 7,0
3020 LOCATE 15,5: PRINT "PRESS A KEY TO RESTORE SCREEN";
3030 Ky$ = INPUT$(1): LOCATE 15,5: PRINT SPACE$(29)
3040 FOR row = 1 TO 6
3050   FOR column = 1 TO 31
3060     LOCATE row, column + 18
3070     PRINT CHR$(Block%(row, column));
3080   NEXT column
3090 NEXT row
3100 RETURN

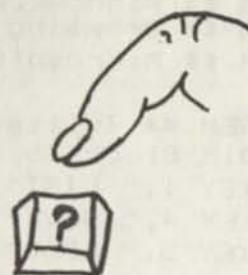
```

Replace
original!

Here is the text that is displayed before the menu window is pulled over it.

This program demonstrates the use of windows.
When you press a key, a window is pulled down
This message.

PRESS A KEY NOW



The following screen shows the menu window pulled down.

This program demon
When you press a k
This message.

To List program	Press F1
To SAVE program	Press F4
To LOAD program	Press F3
PREs To leave program	Press ESC

PRESS A KEY TO RESTORE SCREEN



When the menu is displayed, press one of the keys F1, F3, F4, or ESC. If you press F1, F2, or F3, the menu window will be removed. Then the original text will be replaced and the program will end. BASIC's Ok prompt will appear. Just below the prompt the appropriate text (LIST_, LOAD_, or SAVE_ will appear. At this time you would do one of the following:

- If **LIST_** is displayed, press the ENTER key and the program will be listed.
- If **LOAD_** is displayed, type the file name of the program you want to load at the prompt. Then press ENTER.
- If **SAVE_** is displayed, type the file name under which you want to save the current program. Then press ENTER.

If you press ESC from the menu, the menu window will be removed. The original text will be replaced and the program will end. BASIC's Ok prompt is displayed. The cursor appears immediately below the prompt.

The important keys in **Window Demonstration** are the three subroutines which control the window actions.

Copy Screen Subroutine

THIS SUBROUTINE copies the original characters that lie in the window area into string arrays. Line 1010 places the cursor at the first row (1) and first column (19) of the window area.

1010 LOCATE 1, 19, 0 ← The 0 turns the cursor off so that its movement (as the characters are scanned) is not distracting.

row 1 column 19

The nested FOR...NEXT loop (lines 1020 to 1060) copies the characters in this order:

for each row (1 to 6), the character in each column (19 to 40) is copied.

outside loop inside loop

```
FOR row = 1 TO 6
  FOR column = 1 TO 31
    Block%(row, column) = SCREEN(row, column + 18)
```

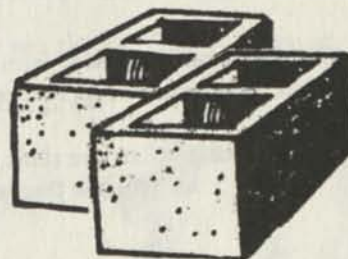
Block%(row, column) ↪ SCREEN(row, column + 18)

Block%(1, 1) is assigned the character at SCREEN(1, 19)
 Block%(1, 2) is assigned the character at SCREEN(1, 20)
 Block%(1, 3) is assigned the character at SCREEN(1, 21)
 .
 .
 .
 Block%(2, 1) is assigned the character at SCREEN(2, 19)
 Block%(2, 2) is assigned the character at SCREEN(2, 20)
 .
 .
 .
 Block%(6, 1) is assigned the character at SCREEN(6, 19)
 .
 .
 .
 Block%(6, 31) is assigned the character at SCREEN(6, 49)

SCREEN(row, column + 18) returns the ASCII code of the character in position row, column + 18. For example:

If row = 5 and column = 10, SCREEN (row, column + 18) would return the ASCII code of the character in position 5, 28.

Since the two values are enclosed in parentheses, BASIC recognizes that you are not specifying a SCREEN mode (which does not use parentheses). When all characters in the window area have been stored in the Block% array, the computer returns to the main program.



Print Window Subroutine

THIS SUBROUTINE uses only one unusual technique. Under usual circumstances, the entire screen is set to specified foreground and background colors. In this program, we want the window to be distinct from the rest of the screen. Therefore, the colors specified for the window are different than the rest of the screen. The entire screen is originally set to white characters on a black background. By changing the foreground color and background color—but not clearing the screen, the new colors are set for printing the window. However, the rest of the screen retains its original white and black colors.

```
2010 LOCATE 1, 19: COLOR 4, 3: PRINT SPACE$(31);
```

From this point (until a new COLOR statement is executed), a read (4) foreground color will be printed on a cyan (3) background.

Replace Original Screen

This subroutine first resets the original colors so that the replacement will match the rest of the screen.

```
3010 COLOR 7,0
```



White on Black

Line 3020 then prints a prompt.

```
3020 LOCATE 15, 5: PRINT "PRESS A KEY TO RESTORE SCREEN";
```

Line 3030 accepts a one-key entry. When it is received, the message of line 3020 is erased.

```
3030 Ky$ = INPUT$(1): LOCATE 15, 5: PRINT SPACE$(29)
```

The FOR . . . NEXT loop at lines 3040 to 3090 uses the ASCII character-codes (retrieved from Block%(row, column)) to place the characters back in their original positions.

```
3040 FOR row = 1 TO 6
3050   FOR column = 1 to 31
3060     LOCATE row, column + 18
3070     PRINT CHR$(Block%(row, column));
3080   NEXT column
3090 NEXT row
```



When finished, a return is made from the subroutine. The program ends with the appropriate menu response on the screen. Press ENTER to carry out the action.

Problem to ponder: Can you think of a quicker way than that used in the Window Display program to display and erase a window?



Introduction

MICROSOFT'S QuickBASIC is the best version of BASIC we have used in 24 years of writing "Teach Yourself BASIC" books for beginners. It's also the most useful, powerful, and capable.

We'd love to hear from people who would like to learn QuickBASIC. We'd love to hear from teachers who are helping kids learn how to use this great *real-life* problem-solving tool . . . or would like to begin doing so. Write to Bob & George, *The BASIC Teacher*, 2418 - 19th Street, San Francisco, CA 94110. If you want a reply, please enclose a self-addressed, stamped envelope.

Together we can create a **ShareWare for Schools Supernova**, a bonanza of *very* low-cost school computing.

WORK DISKS

LAST TIME you made copies of all the disks that came in the QuickBASIC package. You did, didn't you? If not, do so now.

Last time, you made several QuickBASIC Work Disks (QB Work Disks). You did, didn't you? If not, do so now.

We assume you have not one, not two, but a bunch of QB Work Disks, each containing three files, as follows:

COMMAND.COM

QB.EXE

QB.HLP

from MS-DOS

from QuickBASIC

from QuickBASIC



*These QB Work Disks
will make learning and
using QuickBASIC
ever so much easier.*

BEGIN


PUT A QuickBASIC Work Disk into disk drive A and turn on the computer. Whirr goes the disk drive. Soon you see something like this:

```
Current date is Tue 1-01-1980
Enter new date (mm-dd-yy):
```

Type the date and press the ENTER key. Next you see:

```
Current time is 0:00:37.84
Enter new time:
```

Type the time of day and press ENTER. You will then see a Microsoft copyright message and the MS-DOS "A prompt" and a blinking cursor:

```
A>_  blink, blink, blink, . . .
```

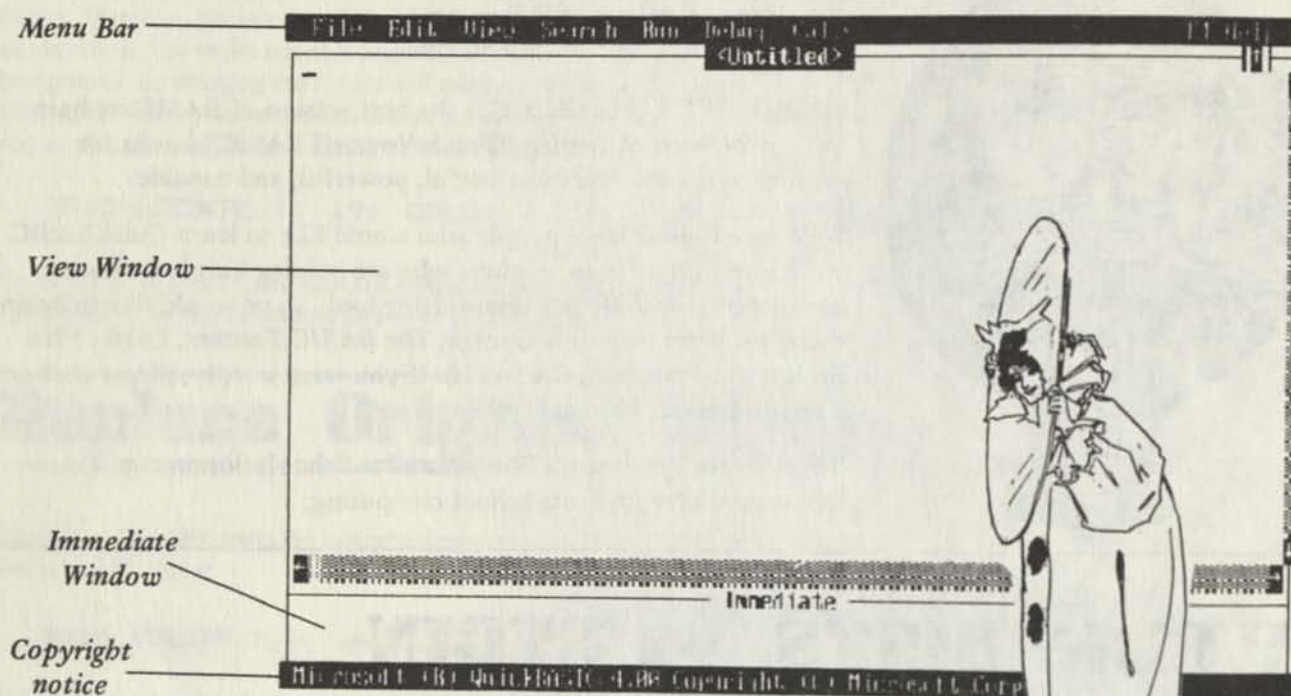
- When you see the familiar A prompt,
- Type **qb** and press ENTER.

The show begins.



House lights dim, curtain rises,

Act 1, Scene 1 begins. You see the QuickBASIC opening screen:



This screen is homebase, work central, the control center. You can go from here to many other interesting places in the QuickBASIC environment. You can usually return here by pressing the ESC key. Do it now:

- Press the **ESC** key.



Zap! QuickBASIC returns you to the control center, work central, the hub of the QuickBASIC environment. Oh, you didn't see anything happen? Well, QB is fast, very fast.

Besides, you were already in QB Control Center.

REMEMBER: To return to QB Control Center, press the ESC key.

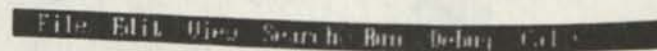
Let your eyes roam about the screen. Please notice a few things:

- The screen has two **windows**.
- Most of the top part of the screen is the **View Window**. At the top of this window, near the center of the screen, you see

QB Control Center

Untitled. Later, you will use the View Window to write, view, and edit QuickBASIC programs.

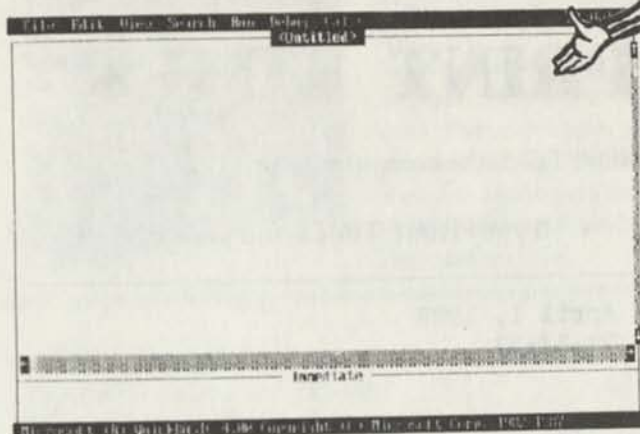
- Near the bottom of the screen you see a much smaller window, long and narrow. It is the **Immediate Window**. You will use the Immediate Window to tell the computer to do something *right now* . . . immediately.
- Across the top of the screen you see a **Menu Bar**. It has the names of other windows you soon will see:



That's all you need to know right now. Next time, you will go **Window Shopping**. Now read on and learn how to use the Immediate Window to tell the computer to do something immediately.

DO IT NOW

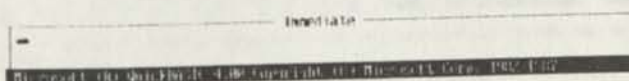
PRESS THE ESC key to make sure you are in QB Control. The screen probably looks like this:



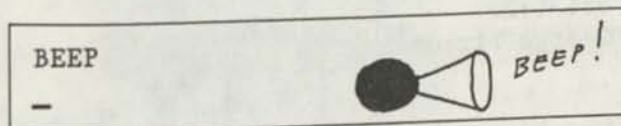
When you first enter QB Control, the cursor is in the upper left corner of the **View Window**, as shown above. Later, you will compose programs in this window.

But first, use the **Immediate Window** to do some things now, immediately . . .

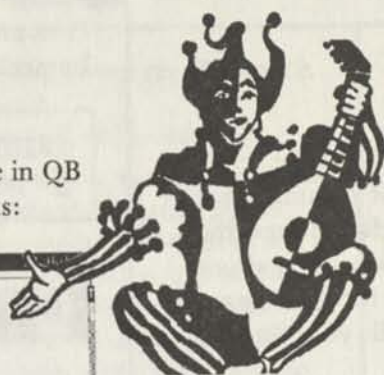
- Press the **F6** function key. The cursor moves into the Immediate Window.



- Press the **F6** key again. The cursor returns to the View Window.
- Put the cursor in the Immediate Window—press **F6**.
- Type **BEEP** and press the **ENTER** key. You should hear a beep.

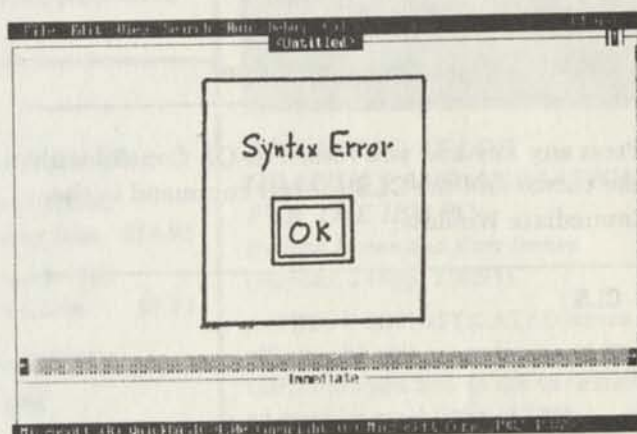


You told the computer to **BEEP**. So it beeped. Yes, it is okay to type **beep** or **Beep** or even **BeeP**. However, if you type **BEAP** or **Boop** or **bleap**, you won't hear a beep. Instead, you will see a **Syntax Error** box in the middle of the view window.



Sir or madam, you can see the cursor going blink, blink, blink in the upper left corner of the View Window.

To see the Syntax Error box, type **Boop** and press **ENTER**.



Press the **ENTER** key. The Syntax Error box will disappear. The cursor will return to the Immediate Window and blink patiently on the word **Boop** the computer does not understand.

You can use the **DELETE** key, the **BACKSPACE** key, and the arrow keys to move the cursor and correct errors.

- Use the arrow keys to move the cursor within a window.
- Use the **BACKSPACE** key to delete characters to the left of the cursor. Each time you press **BACKSPACE**, the cursor moves one space to the left and deletes the character in that place.
- Use the **DELETE** key to delete the character at the cursor position. Characters to the right of the cursor will move left one position.

Use the **DELETE**, **BACKSPACE**, and arrow keys to correct the error (change **Boop** to **Beep**), then press **ENTER** and hear another beep.

CLS

FOR YOUR next easy command, tell the computer to clear the screen. The command for this is **CLS** or **cls**. Type **CLS** or **cls** and press the ENTER key. QB Control will disappear and you will see an empty screen, except for one line at the bottom of the screen that tells you "Press any key to continue."

Press any key to continue

Press any key and you return to QB Control with the cursor and the **CLS** (or **cls**) command in the Immediate Window.

CLS
—

Note that only one command is shown in the Immediate Window. Each new command "pushes" the previous command out of the window. This happens when you press the ENTER key. Later you will learn how to expand the Immediate Window so it will hold more commands at one time.

PRINT DATE\$

WHEN YOU began this session, did you set the date and time? You can use the **PRINT** keyword in commands to print the date and time. You can also print your name or any other message.

- Type **PRINT DATE\$** and press the ENTER key.

April 1, 1988

Press any key to continue



Press any key to return to QB Control. The Immediate Window contains your last command.

PRINT DATE\$
—

PRINT TIME\$

NOW TELL the computer to print the time.

- Type **PRINT TIME\$** and press ENTER:

April 1, 1988
08:37:23

Press any key to continue

Return to QB Control (press any key). Your last command is in the Immediate Window.

PRINT TIME\$
—

PRINT " " "

TELL THE computer to print your name. To do so, type the word **PRINT**, followed by your name enclosed in quotation marks, then press ENTER.

George Firedrake did it like this:

- George typed **PRINT "George Firedrake"** and pressed the ENTER key.

April 1, 1988
08:37:23
George Firedrake

Press any key to continue

Well, that's all for this time. Play around in the Immediate Window or press **F6** to return to the View Window.

NEXT TIME: Window Shopping.

Special Reader Services

See How to Order
on next page



QuickBASIC: The Complete Reference
By Steven Nameroff
(Osborne/McGraw-Hill, 700pp, \$24.95)

A comprehensive guide to Microsoft's extremely powerful QuickBASIC compiler, this resource is written for users at all levels of programming ability, from novices to pros. The author has divided the book into sections to help you easily locate the information you need. The book begins with a quick introduction to BASIC programming, followed by a complete command reference section and a discussion of QuickBASIC functions, procedures, files, and graphics.

USING QuickBASIC

By Don Inman and Bob Albrecht
(Osborne/McGraw-Hill, 436pp, \$19.95)

HERE'S AN excellent programming guide to Microsoft's newest version of QuickBASIC by the authors of *The BASIC Teacher*. The book approaches QuickBASIC's programming environment in three stages so beginning and experienced BASIC programmers can find the appropriate level of instruction.



ALSO AVAILABLE:

Using QuickBASIC
Convenience Disk \$14.95
Using QuickBASIC
Teacher's Guide \$9.95

SPECIAL OFFERS:

Book & Disk
\$31.41 (save \$3.49)
Book & Guide
\$26.91 (save \$2.99)
Book, Disk & Guide
\$38.12 (save \$6.73)

Advanced Color Graphics and Animation for the IBM PC



ADVANCED COLOR GRAPHICS AND ANIMATION FOR THE IBM PC

By Don Inman and Kurt Inman
(Hayden, 248pp, \$18.95)

ACHIEVE SOPHISTICATED screen effects with this example-oriented text that shows you how to use the extended graphics capabilities of IBM BASICA for programming displays. Whether you're creating a business chart or an artistic animation, you'll learn the techniques that give your work professional results.

FOR ATARI USERS



ATARI BASIC: XL EDITION

By Bob Albrecht, Leroy Finkel,
and Jerald R. Brown
(Wiley Press, 388pp, ~~\$14.95~~)

THIS CLASSIC shows how to adapt BASIC to Atari's XL series of microcomputers and its 400, 800, and 1200 machines. You need no math, science, or computer background to learn to read and write Atari BASIC. Empha-

ATARI GAMES & RECREATIONS

By Herb Kohl, Ted Kohn, and
Len Lindsay, with Pat Cleland
(Reston, 338pp, ~~\$14.95~~)

THIS BOOK offers a very different approach to introducing programming to the novice computer user. The authors encourage you to develop

sizing good programming style, this self-paced, easy-to-understand guide takes you step-by-step thru simple techniques for creating programs for home, schools, or business applications. Lots of educational and recreational activities and exercises—featuring sound, color, graphics, games, and simulations—making learning easy and fun.

NOW AVAILABLE FOR ONLY \$7.50!

your own ideas for computer games and provide models from which to draw ideas for such games. In addition to games, you'll find a special section on the graphics, sound, and color features of your Atari 400 or 800. The book can also serve as a basic learning guide for kids and adults alike, and as a sourcebook for teachers.

NOW AVAILABLE FOR ONLY \$7.50!

The ShareWare Book
Using PC-Write, PC-File, PC-Talk
 By Emil Flock, et al
 (Osborne/McGraw-Hill, 688pp,
 \$14.95)

Covers the most popular "free" programs: PC-Write, a word processor; PC-File, a database manager; and PC-Talk, a telecommunications program. These programs are available thru user groups or bulletin-board services in return for a nominal registration fee. The book has all the details on how you can obtain these program disks.



DOS Made Easy
 By Herbert Schildt
 (Osborne/McGraw-Hill, 385pp,
 \$18.95)

Previous computer experience is not necessary to understand this concise, well-organized introduction that's filled with short applications and exercises. The book walks you thru all the basics, beginning with an overview of a computer system's inner components and a step-by-step account of how to run DOS for the first time.

4 Tools in One
 (Microsoft, boxed, \$195)

EASY TO learn and use, *Works* contains four of the most popular computer tools:

- **Word Processor:** Produce professional-looking letters, reports, and memos.
- **Spreadsheet:** Develop budgets, forecasts, and balance sheets.
- **Database:** Organize and keep track of names and business records.
- **Communications:** Gain access to a world of information.

Microsoft Works

The package comes with understandable training and reference materials including a separate computer program that actually teaches you how to use the tools. And it operates in a way that makes sense—even to novice users.

The best and easiest-to-use integrated software on the market, *Microsoft Works* does the work of many different programs, and you have to pay for and learn only one.

For IBM PC, IBM PS2, and compatibles. Contains both 3½" and 5¼" disks.

Microsoft QuickBASIC 4.0

(Microsoft, boxed, \$95)

How to Order

SEND CHECK or money order to Different Worlds Publications, 2814 - 19th Street, San Francisco, CA 94110. Add \$2 for first item and \$1 for each additional item to cover postage and handling. California residents add appropriate sales tax.

SATISFACTION GUARANTEED

*Back Issue of
 The BASIC Teacher
 Available*

Issue No. 1 \$3.00



BULK RATE
 U.S. Postage
 P A I D
 San Francisco, CA
 Permit No. 11798

Different Worlds Publications

2814 - 19th Street
 San Francisco, CA 94110

Address Correction Requested

282-0999



No. 3, September 1988

\$3.00

*For beginning
programmers with no prior
programming experience*

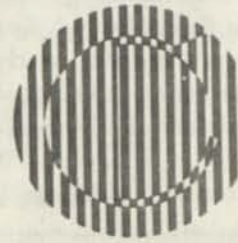
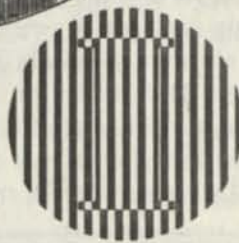
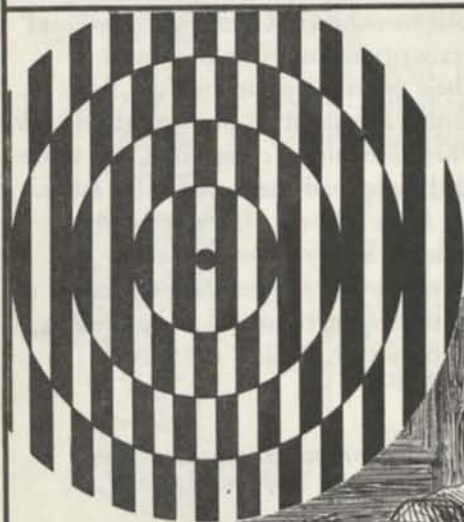
CONTENTS

- 3 Teach Yourself BASIC
- 11 Browsing BASIC
- 19 Teach Yourself
QuickBASIC

PUBLISHER'S STATEMENT: *The BASIC Teacher* is published monthly by Different Worlds Publications, 2814 - 19th Street, San Francisco, CA 94110. Contents copyright 1988 by Different Worlds Publications. All rights reserved. Contents may be copied and distributed freely. Address all correspondences to *The BASIC Teacher*, 2814 - 19th Street, San Francisco, CA 94110.

SUBSCRIPTION INFO: A 12-issue sub in the U.S. and Canada is \$36. Overseas subs are \$42 by surface mail, \$54 by air.

PRINTED IN THE U.S.A.



BASIC stands for Beginner's All-purpose Symbolic Instruction Code. Developed at Dartmouth College by T.E. Kurtz and J.G. Kemeny, BASIC was created to teach students programming. It is widely used on microcomputers and minicomputers for educational and business applications. It is easy to understand and learn and is appropriate for solving small problems. BASIC is the most popular programming language available for computers.

"The Shareware Teacher"

Wendy Wiegard, Dean Brown, and Bob Albrecht have started a new column in the *Shareware Magazine* called "The Shareware Teacher." The column is written "for all those with an interest in designing, writing, and selling good, inexpensive, educational software—for school or for home use." If you have written some teaching programs, but don't know how to get them out to teachers and learners, this column is for you. The authors propose that QuickBASIC 4.0 be used as the standard developmental programming language because it is rich, powerful, portable, and fast. The *Shareware Magazine* is available from PC-SIG, 1030 D East Duane Avenue, Sunnyvale, CA 94086, (800) 245-6717, (800) 222-2996 in California.

National survey says students aren't yet in the computer age

Most U.S. students remain computer illiterates and few schools use computers effectively to teach basics despite a much-heralded "classroom computer revolution," according to a recent survey by the National Assessment of Educational Progress.

The study of 24,000 third-, seventh-, and 11th graders found that access to computers is too limited at school and at home, school curricula haven't changed to make effective use of computers, and teachers are inadequately trained.

The study found that most third graders could identify the main parts of a computer, such as the keyboard, floppy disk, and joystick. By seventh grade, nearly all could. But only 32.3 percent of third graders, 34.3 percent of seventh graders, and 46 percent of 11th graders could correctly answer: "What is the main role of a computer program?" (Answer: "To tell the

computer what to do.") And only three of ten 11th graders knew what an algorithm is ("a step by step process for solving a given type of problem").

Except for word processing, students generally scored poorly on questions dealing with computer applications—making graphics or working with databases, for example—because students said they hardly ever got a chance to practice. About two-thirds of those surveyed said they had never written computer programs.

Despite their generally poor showing, 91.2 percent of third graders and 86 percent of seventh graders said they liked using computers, and more than half said they wished they used them more.

"Although some schools offer excellent computer curricula, many other schools apparently do not provide effective instruction, at least in computer applications and programming, to a large proportion of pupils," the survey concluded.

Critical shortage of computer memory chips burts U.S. businesses

Within the past half year, the supply of DRAM chips has fallen far short of U.S. demand, and DRAM prices have risen to damagingly high levels. Many would-be buyers are delaying their purchases until prices fall, 15 percent of the respondents to a recent PC MagNet survey reported plans to put off purchases until DRAM prices fall to acceptable levels.

These developments aggravate the continued development of the PC, which is dependent on a ready supply of reasonably priced RAM. OS/2 with the Presentation Manager requires a minimum of 5MB RAM, and this requirement shows no signs of getting any smaller.

"If we could get our hands on more DRAM, we would be a much larger company in terms of revenue," says a Sun Microsystems spokesman. "The principal constraint we are under is a shortage of DRAM, not a shortage of business."

Help is due to arrive in the form of 1-megabit chips when their output is predicted to double between the second and fourth quarter in 1988, when manufacturers begin ramping up their production.

☆☆☆☆☆



Introduction

WE BEGAN using BASIC in 1964 when it was very, very new. As year followed year, BASIC became better and better, according to the needs of the people who used it. Today, Microsoft's GW-BASIC and QuickBASIC are truly the People's Computer Languages, running on more than 20 million computers. BASIC continues to get better and better—you can count on it.

BASIC has a small **vocabulary** and a simple **syntax** (grammar). We have already discussed some of the special words that Microsoft BASIC understands. They are called **keywords** or **reserved words**.



Here are the keywords introduced and described previously:

BEEP CLS DATE\$ KEY OFF KEY ON PRINT TIME\$ TIMER

You also learned how to use the computer to do arithmetic, using the four arithmetic operations: + - * /. You know how to read numbers expressed in **floating point notation** such as 5E+09 or 1.67 E-27.

You have probably encountered the **syntax error** message. This simply means the computer didn't understand you. That's OK, just try again—and learn more BASIC keywords that the computer does understand.

Tiny Programs

INSTALL BASIC in your computer and try a tiny program. Your first program is very simple. It tells the computer print Mariko's name on the screen, and continue printing it until someone stops the computer.

```
10 PRINT "Mariko"
20 GOTO 10
```

This program has two **lines**. Each line begins with a **line number**.



```
1st Line: 10 PRINT "Mariko"
2nd Line: 20 GOTO 10
```

↑
line numbers

We'll use the line numbers to identify the lines. So the 1st line is "line 10" and the 2nd line is "line 20."

Each line of the program contains a BASIC **keyword**. The keywords in this tiny program are PRINT and GOTO. We have used PRINT previously; GOTO is new.

Soon you will **enter the program** into the computer's memory. You enter a program by typing it, one line at a time, on the keyboard.

- After typing a line, press the ENTER key.
- If you make a typing mistake before you press ENTER, use the BACK SPACE key to erase the error.
- If you make a mistake and press ENTER, the incorrect line will be in the computer's memory. That's OK. Just retype the line correctly and press ENTER. The new line will replace the old, incorrect line.

*Read the above again
then
GOTO the next page*

Enter the Program

BEFORE YOU enter a new program, tell the computer you are entering a new program. Do this by typing NEW and pressing ENTER. This erases any old program that might be in the computer's memory. We suggest you also clear the screen before you begin.

- Clear the screen. Hold down the CTRL key and press the L key. This erases the screen and puts the cursor in the upper left corner.
- Type **NEW** and press ENTER.

If you misspell NEW, you may see an error message. That's OK, try again. Type NEW and press ENTER.

```
GNU
Syntax error
Ok
—
```



Now enter the program slowly and carefully:

- Type **10 PRINT "Mariko"** and press ENTER.
- Type **20 GOTO 10** and press ENTER.

If you entered the program without making any typing mistakes, the screen looks like this:

```
NEW
Ok
10 PRINT "Mariko"
20 GOTO 10
```



*It's OK to type
NEW, PRINT,
and GOTO in
lower-case letters
as new, print,
and goto.*

If you see a mistake, that's OK. Just ignore it and read on . . . we will soon tell you how to fix mistakes.

LIST the Program

THE PROGRAM is now stored in the computer's memory. You can always find out what program is in the memory by telling the computer to **LIST** on the screen what is stored in memory. Let's do it:

- Clear the screen. Hold down the CTRL key and press the L key.
- Type **LIST** and press ENTER.

Quick as a wink, the computer lists the program on the screen. Ours looks like this:

```
LIST
10 PRINT "Mariko"
20 GOTO 10
Ok
—
```

Now is the time to correct any incorrect lines. Just retype the line, including the line number, and press ENTER. Then LIST the program again to make sure it is correct.

Blackboard Erasers.

53508 Chicago Dustless Eraser, wool felt, cleans the board thoroughly, very durably made. Weight, packed, 4 ounces. Each 6c.



Per doz. \$0.65
Per gross. 6.95

If you see a line with line number 1 or 12 or 19 or any line number other than 10 or 20, please get rid of it:

- To erase a line from memory, type only the line number and press ENTER.

For example, to erase line 19:

- Type **19** and press ENTER.

This erases *only* line 19 from *memory*. All other lines remain in memory. Line 19 might still show on the screen, but it has been erased from the computer's memory. LIST the program again. Is there a line missing? If so type the missing line and press ENTER. Fix any incorrect lines, then move on.

Run the Program

THE PROGRAM is in memory and ready to use. Tell the computer to RUN the program:

- Type **RUN** and press ENTER.

In a twinkling, Mariko's name appears on every line and you see this on the screen:

```

Mariko
Mariko
Mariko
Mariko
Mariko
1LIST  2RUN←  3LOAD
  
```

Wow! That was fast.

The bottom line, of course, is the Key Line.

The computer keeps printing **Mariko** on the line just above the Key Line. Each time it does this, all the other **Marikos** are pushed up one place. The top **Mariko** is "pushed off" the top of the screen. This happens so quickly, however, that only superheroes with ultrafast eyes can see it happen.

Stop the Computer

HOW DO you stop the computer? Easy.

- Hold down the CTRL key and press the BREAK key.

The bottom part of the screen probably looks like this:

```

Mariko
Mariko
Mariko
Break in 10
Ok
1LIST  2RUN←  3LOAD
  
```

To stop the computer:

- Hold down CTRL
- and press BREAK.

The Key Line

How Does It Work?

HOW DOES the program work? Here again is the program:

```

10 PRINT "Mariko"
20 GOTO 10
  
```

Line 10 tells the computer to print the word enclosed in quotation marks which, as you can see, is **Mariko**.

Line 20 tells the computer to go to line 10 and continue.

When you type **RUN** and press ENTER, the computer starts running the program at the *smallest* line number. It does line 10, then line 20, then line 10, then line 20, then line 10, then line 20, and so on, and so on and so on... until someone holds down CTRL and press BREAK.

Follow the arrows:

```

RUN
↓
10 PRINT "Mariko"
↓
20 GOTO 10
  
```

Around and around and around and around... until CTRL+BREAK



REMEMBER: Line numbers tell the computer the order in which to do things. Line numbers do not have to be consecutive integers such as 1, 2, 3, 4, 5, and so on. Instead, it is better to number by tens as we did in the above program. Then, if you wish, you can easily insert or add more lines between ones you already have. For example, in the above program you can add nine more lines between lines 10 and 20 (lines 11, 12, 13, 14, 15, 16, 17, 18, and 19).

Mistrakes

WHEN YOU type, you will probably make mistrakes.

Oops! We mean you will probably make mistakes.

Uh . . . sorry . . . we think you might sometimes press the wrong keez.

Oh well, you know what we mean, don't you?

There are so many ways to make mistakes in entering a program. Here is a program we want to enter in order to remind someone how to stop the computer.

```
10 PRINT "Hold down CTRL and press BREAK"
20 GOTO 10
```

Unfortunately, we entered it as shown on the screen below:

```
NEW
Ok
10 PRINT "Hold down CTRL and press BREAK"
20 GOTO 19
```

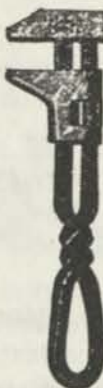
Oops!

Well, you might expect some problems in running this program. The GOTO has no place to go . . . there is no line 19. Here is what happened when we ran the program:

```
RUN
Hold down CTRL and press BREAK
Undefined line number in 20
Ok
```

We could fix line 20 by retyping the entire line correctly. Instead let's do it a different way, using the **cursor control keys**, also called the **arrow keys**.

- RIGHT ARROW moves the cursor right.
- LEFT ARROW moves the cursor left.
- UP ARROW moves the cursor up.
- DOWN ARROW moves the cursor down.



LIST It & Fix It

CLEAR THE screen and LIST the program:

```
LIST
10 PRINT "Hold down CTRL and press BREAK"
20 GOTO 19
Ok
-
```

Use the arrow keys to move the cursor up and over until it is under the 9 in 19.

```
20 GOTO 19
```

cursor

Type a zero (0) to correct the error. This is what you then see:

```
20 GOTO 10
```

cursor

Now press ENTER. The computer stores the corrected line 20 in its memory, replacing the old, incorrect line 20. To see that this is true, clear the screen and LIST the program:

```
LIST
10 PRINT "Hold down CTRL and press BREAK"
20 GOTO 10
Ok
-
```

Now suppose you have just entered the following program:

```
10 PRINT "Merry Christmas, Faather"
20 GOTO 10
```

Oops!

Position the cursor under the second a in Faather and press the BACK SPACE key to erase the first a. Then press ENTER.

Suppose the error is:

```
10 PRINT "Happy Birthday, Moter"
```

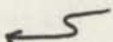
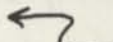
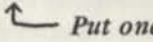
Oops!

Put the cursor under the e in Moter. Press INS, then type an h to correct the speling for Mother. OK? Press ENTER.



Variations of Line 10

NOW YOU know several ways to make corrections or desired changes to the program. Experiment. Try some of these tiny programs. When you type one of these programs, be sure to include the comma (,) or semicolon (;) at the right end of line 10:

- 10 PRINT "Lucy",  *comma*
20 GOTO 10
- 10 PRINT "Lucy";  *semicolon*
20 GOTO 10
- 10 PRINT "Lucy ";  *Put one space here.*
20 GOTO 10

The comma (,) at the end causes a "tab" to the right to the next **standard print position**. There are five standard print positions. The semicolon (;) causes things to be printed close together.

Here are three programs:

1. 10 PRINT "Ha Ha"
20 GOTO 10
2. 10 PRINT "Ha Ha",
20 GOTO 10
3. 10 PRINT "Ha Ha";
20 GOTO 10

Which program above would produce the RUN shown below?

Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha
Ha Ha	Ha Ha	Ha Ha	Ha Ha	Ha Ha

... and so on.

Things to Do

1. TEACH THE computer to cry. Complete the following program to fill the screen with **Boo Hoo**.


```
10 PRINT _____
20 GOTO 10
```


2. Write a program to produce the following run.

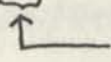
```
Garfield loves lasagna
Garfield loves lasagna
Garfield loves lasagna
Garfield loves lasagna
Garfield loves lasagna
.
.
.
```

We Did It Like This

1. HERE ARE two ways:

```
10 PRINT "Boo Hoo",  comma
20 GOTO 10
```

```
10 PRINT "Boo Hoo ";  semicolon
20 GOTO 10
```

 *3 spaces*

2. Here is our program:

```
10 PRINT "Garfield loves lasagna"
20 GOTO 10
```


Big Letters

YOU CAN tell the computer to display 80 characters across the screen or 40 characters per line. Do it. Tell the computer to display 40 characters per line:

- Type **WIDTH 40** and press ENTER.

When we did it, we saw a big **Ok**.



Now tell the computer to display 80 characters per line.

- Type **WIDTH 80** and press ENTER.

When we did it, we saw a smaller **Ok**.

We usually like 80 characters per line when we enter a program.

Now tell the computer to display 80 characters per line and enter the following program. When you run the program, it tells the computer to shift to 40 characters per line.

```
10 WIDTH 40
20 PRINT "My human likes me"
30 GOTO 20
```

Run the program and you will see "My human likes me" in double-width letters on the screen:

```
My human likes me
My human likes me
My human likes me
My human likes me
My human likes me
My human likes me
My human likes me
```

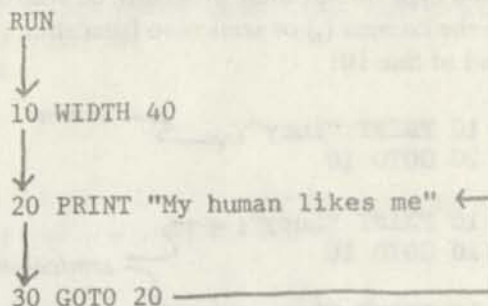
... and so on.

Stop the program (CTRL+BREAK). This will leave the screen in 40 characters per line mode. If you want 80 characters per line:

- Type **WIDTH 80** and press ENTER.

When you run this program, the computer does line 10 only once, then goes around and around with lines 20 and 30. Note that line 30 tells the computer to go to line 20.

Follow the arrows:



Now draw arrows showing how the computer runs the following program:

```

RUN

10 WIDTH 40

20 BEEP

30 PRINT "The time is", TIME$

40 GOTO 30
    
```

Enter the program and run it. You will see something like the following, but in big letters. Watch the seconds change ...

```

The time is 12:37:00
The time is 12:37:00
The time is 12:37:00
The time is 12:37:00
The time is 12:37:01
The time is 12:37:01
The time is 12:37:01
The time is 12:37:01
The time is 12:37:01
The time is 12:37:01
The time is 12:37:01
    
```

... and so on.

The BASIC Rainbow

WHEN YOU first load BASIC, it prints on the screen in gray on a black background. You can tell the computer to print in any of 16 colors, including black. Of course, black print on a black background is somewhat hard to see. Go ahead, tell the computer to print in **red**:

- Clear the screen (CTRL+L)
- Type **COLOR 4** and press ENTER.

You will see **Ok** in red. The cursor remains white.

```
COLOR 4    Tells the computer to use red.
Ok         This is in red.
- ←       The cursor remains gray.
```

Don't clear the screen. Tell the computer to print in light green.

- Type **COLOR 10** and press ENTER.

Now you see:

```
COLOR 4    Tells the computer to use red.
Ok         Red.
COLOR 10   Still in red.
Ok         Light green.
- ←       The cursor stays gray.
```

Don't clear the screen. Tell the computer to print in light blue:

- Type **COLOR 9** and press ENTER.

Now you see:

```
COLOR 4    Tells the computer to use red.
Ok         So this is in red.
COLOR 10   Tells the computer to use green.
Ok         So this is in green.
COLOR 9    Tells the computer to use blue.
Ok         So this is in blue.
- ←       The cursor is still gray.
```

Don't clear the screen. Tell the computer to print in the original screen color, white:

- Type **COLOR 7** and press ENTER.

Now you see:

```
COLOR 4    Red
Ok
COLOR 10   Green
Ok
COLOR 9    Blue
Ok
COLOR 7    Gray
Ok
```



The next experiment is tricky ... black on black:

- Type **COLOR 0** and press ENTER.

Now you don't see **Ok**!

```

.
.
.
COLOR 7
Ok         Gray
COLOR 0
- ←       Oops ... where is Ok?
```

Well, black print on a black screen is quite invisible. You can now type **COLOR 7**, but you won't see it. The cursor moves ... so you know something is happening:

- Type **COLOR 7** and press ENTER.

Now you see:

```

.
.
.
COLOR 7
Ok         Gray
COLOR 0
Ok         Ok is invisible.
           COLOR 7 is invisible.
- ←       Aba! Visible again.
```


PRINT in COLOR

YOU CAN use the COLOR keyword to tell the computer to PRINT in the color of your choice, selected from the following list of 16 colors:

0 black	8 dark gray
1 blue	9 light blue
2 green	10 light green
3 cyan	11 light cyan
4 red	12 light red
5 magenta	13 light magenta
6 brown	14 yellow
7 gray	15 white

Mary Jane, Heidi, and Karl picked red, green, and blue as the colors for their names everywhere on the screen. Here are their programs:

Mary Jane's Program:

```
10 COLOR 4           Red
20 PRINT "Mary Jane ";
30 GOTO 20
```

Heidi's Program:

```
10 COLOR 2           Green
20 PRINT "Heidi ";
30 GOTO 20
```

Karl's Program:

```
10 COLOR 1           Blue
20 PRINT "Karl ";
30 GOTO 20
```

Tom couldn't decide between light green and light magenta. So he did his program like this:

```
10 COLOR 10
20 PRINT "Tom";
30 COLOR 13
40 PRINT "Tom";
50 GOTO 10
```

Remember me!

“ ; ”

Your turn. Pick a color and write your program. If you want to see your name in BIG LETTERS, include WIDTH 40 in your program.

Random Colors

MARIKO COULDN't decide on just one color, or two colors, or more colors. So she decided to let the computer pick the colors at random, using a special keyword called RND with the help of another keyword called INT:

Mariko's Program:

```
10 COLOR INT(15*RND) + 1
20 PRINT "Mariko ";
30 GOTO 10
```

We encourage you to run Mariko's program. Of course it is OK to use your name instead of Mariko's.

Remember, to stop the program:

- Hold down CTRL and press BREAK.

It can stop in any of 15 colors. So tell the computer to use gray:

- Type COLOR 7 and press ENTER.

If you were using BIG LETTERS (WIDTH 40) and want to get back to small letters (WIDTH 80):

- Type WIDTH 80 and press ENTER.

Oh yes, next time we will tell you more about RND and INT.



WE DIGRESS from BASIC in this issue to talk a little bit about your Disk Operating System (DOS). DOS is the master program that coordinates the flow of information from your computer to your disks and from your disks to your computer. Every computer that uses disks must have a disk operating system. Despite its importance, DOS is quite often the most ignored part of a computer system. If you're like me, you know DOS is necessary, but only use it for the most basic things—like formatting and copying disks and disk files. Of course, DOS is also necessary to load the software that we use.

In this issue, we will explore some of the basic tasks that MS-DOS (or PC-DOS) can perform. As you read through the article, you should stop often to try out any DOS commands that are new, or unfamiliar, to you. Seeing them work immediately may entice you to use them in the future.

We'll begin by assuming that you know the following DOS commands in their most elementary forms:

DIR DISKCOPY COPY FORMAT

We will discuss some options to the commands above and add some new DOS features in this article. New terms discussed are:

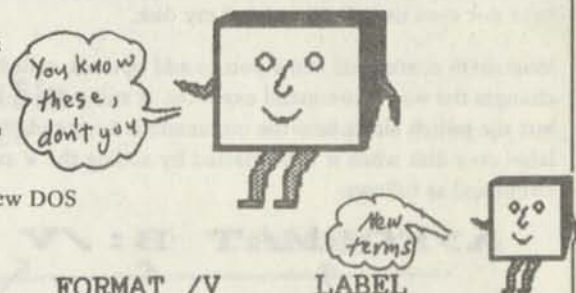
COPY CON DEVICE DRIVER.SYS FORMAT /V LABEL

DOS features discussed here are for MS-DOS, version 3.2 and 3.3. If a feature does not work on an earlier version, this fact will be stated in the discussion. Here is the MS-DOS I am using:

```
Microsoft MS-DOS Version 3.20
(C)Copyright Microsoft Corp 1981, 1986
Tandy Version 03.20.21
Licensed to Tandy Corp.
All rights reserved.
```

Microsoft licenses MS-DOS to many computer manufacturers. Notice that the MS-DOS version 3.20 used here is Tandy version 03.20.21. As you browse the following discussion, remember that there may be slight differences between this version and the version of MS-DOS that you are using.

In writing books and articles, I use a lot of disks and files. This article is saved to a text file named **BROWSE03.DOC**. The extension (DOC) reminds me that this is a document (text file). Other files pertaining to the article contain BASIC programs with the extension **BAS**. Still other files contain artwork (screen prints) used in the article. These files have the extension **PIX** or **PIC** (for picture). I like to have all files of any given article on the same disk so that everything related to the article can be easily found.



LABEL YOUR DISKS

When I first looked at the directory of the disk used for this article, I was mildly irritated when I saw:

```
A>DIR B:
```

```
Volume in drive B has no label
Directory of B:\
```

```
File not found ← This means no files
                  on the disk yet
```



See "Volume in drive B has no label" at the top line? Because of my laziness, I have not even used DOS to label my disk.

Most DOS commands allow you to add options, called **switches**. A switch changes the way a command executes. It still performs the same basic function, but the switch alters *how* the command is executed. For instance, you can put a label on a disk when it is formatted by adding the V switch to the FORMAT command as follows:

```
A>FORMAT B: /V
      ↑      ↑      ↑
      |      |      |
Format disk in drive B  V switch causes the computer to pause
                        & prompts you for a label
```

Here is the sequence of information and prompts when the above command is entered on my Tandy 1000TX.

```
A>FORMAT B:/V
Insert new diskette for drive B
and strike ENTER when ready

Format complete ← Formatting takes place

Volume label (11 characters, ENTER for none)? _ ← Pause for label
```

A label can contain up to 11 characters. I entered BROWSING03 as my label:

```
Volume label (11 characters, ENTER for none)? BROWSING03 ← Label

362496 bytes total disk space }
362496 bytes available on disk } ← More formatting info

Format another (Y/N)?N
```


The label isn't really necessary, but it immediately informs me of the disk contents—this disk contains material for the third article in the "Browsing BASIC" series.

So . . . you forget, or aren't ready, to label the disk when it is formatted. You save a file, or a few files, to the disk and then want to label it. If you have MS-DOS version 3.0 or higher, you can do this with the command:

A>LABEL B:

For example, my disk has been formatted but not labeled. It contains the file BROWSE03.DOC. Here is the directory:

A>DIR B:

Volume in drive B has no label
Directory of B:\

BROWSE03.DOC	4814	5-13-88	9:51a
1 File(s)	357376	bytes free	

I now use the LABEL command:

A>LABEL B:

Volume in drive B has no label
Volume label (11 characters, ENTER for none)? BROWSING03

A look at the directory of the disk in drive B now shows:

A>DIR B:

Volume in drive B is BROWSING03
Directory of B:\

BROWSE03.DOC	4814	5-13-88	9:59a
1 File(s)	357376	bytes free	

A>_

As you can see, the disk now has the label BROWSING03 and contains that part of the article that has been saved at this time. Now that the disk is labeled, let's move on to a more complicated problem:



COPYING A FILE TO THE SAME DRIVE

Three different types of disks are commonly used with today's personal computers: hard disks, 5¼" floppy disks, and 3½" diskettes. We are now in a disk transition period in which both hardware and software are changing from 5¼" drives and disks to 3½" drives and diskettes. This poses some problems when copying disks from one drive to another. The particular problem, discussed here, is copying a file from a disk in one drive to another disk in the same drive.

I have one 5¼" drive and one 3½" drive installed in my Tandy 1000TX so that I can use files saved in either format. Now, suppose I want to save the present version of my BROWSE03.DOC file to a different disk than the current one which is in drive B. I want to save it to a new disk in drive B, the same drive.

DOS will not let me copy a file to the same drive. This is the message I get when I try to copy BROWSE03 to the same drive where it resides:

```
A>COPY B:BROWSE03.DOC B:
File cannot be copied onto itself
0 File(s) copied
```

```
A>_
```

A similar message appears when I try to copy a file from Drive A to Drive A:

```
A>COPY SPOOLER.SYS A:
File cannot be copied onto itself
0 File(s) copied
```

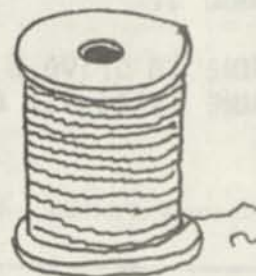
```
A>_
```

This problem can be solved by writing a DEVICE command to the CONFIG.SYS file on the DOS disk. However, if you've ignored DOS capabilities as I have, there is no CONFIG.SYS on your DOS disk. You can create the CONFIG.SYS file in a word processor, save it in ASCII format, and copy it to your DOS disk. However, since this file will be short, it can easily be done within DOS.

DOS has many capabilities, one of which is the ability to write files to itself directly from the keyboard. In this instance, I want to write a CONFIG.SYS file that will give each of my two disk drives two names. I presently have the 3½" drive installed as drive A. The 5¼" drive is installed as drive B. I want the CONFIG.SYS file to add the designation C to drive A and the designation D to drive B. Then I can copy a disk to the same physical drive with the command:

```
A>COPY B: BROWSE03.DOC D:
```

DOS will accept this command even though B and D are the same physical drive. DOS will even tell me when to insert the correct disk as you shall see later in the article. However, we must first create CONFIG.SYS.



WRITING A DOS FILE

The DOS command:

A>COPY CON: CONFIG.SYS

opens a file named CONFIG.SYS for text. The term CON means console, a combination of your monitor and keyboard. The DOS command COPY CON means copy (from the console) the file that follows (CONFIG.SYS). Remember, your DOS system disk should be in drive A and *unprotected* in order to write the file.

The file is created by typing text and pressing the ENTER key at the end of each line of text. The file is closed by typing CTRL Z and pressing ENTER. The file is automatically sent to the disk in drive A.

The file entries are echoed on the screen as the file is created. Here is how the final screen looks:

```
A>COPY CON:CONFIG.SYS
DEVICE = DRIVER.SYS /D:0 /F:2
DEVICE = DRIVER.SYS /D:1 /F:0
^Z
1 File(s) copied
```

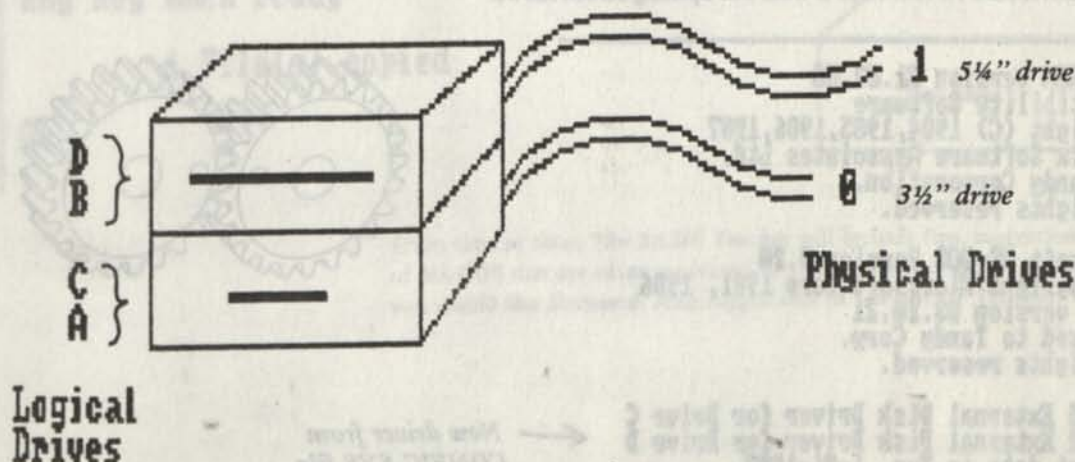
*The ENTER key is pressed
at the end of each line*

*The fourth line results
from pressing CTRL Z*

DRIVER.SYS is an installable device driver that enables your system to use more than two floppy disk drives and more than two hard disk drives. It also allows you to assign more than one logical drive letter to one physical drive.

Keep in mind that disk drives have **physical drive numbers**. Physical floppy drives are numbered from 0 (zero). Since I have two physical drives, they are numbered 0 and 1. The 3½" drive is number zero, and the 5¼" drives is number one. I have no other drives.

We know and use the drives by their **logical letter**. My 3½" drives is usually known as drive A, and my 5¼" drive is usually known as drive B. If I had a third drive, it would be logical drive C. A fourth drive would be logical drive D.



Let's look at each DEVICE command separately:

1. DEVICE = DRIVER.SYS /D:0 /F:2

This command causes MS-DOS to assign the *next available drive letter* (C in this example) to the drive number that follows the /D switch (0). Since C is the next available drive on my system, **logical drive C** is assigned to **physical drive 0**. Now I have assigned two logical drive letters (A and C) to what the computer knows as physical drive 0.

The value following the /F switch specifies the form factor index to this drive. The index values are:

0 = 320/360K floppy	4 = 8" double-density floppy
1 = 1.2Mb floppy	5 = hard disk
2 = 720K floppy (3½")	6 = tape drive
3 = 8" single-density floppy	7 = other

Thus /F:2 tells the computer that the new logical drive (C) is a 720K floppy 3½" drive.

2. DEVICE = DRIVER.SYS /D:1 /F:0

This command causes MS-DOS to assign the *next available drive letter* (D in this example since we previously used C) to the drive number that follows the /D switch (1). Thus, **logical drive D** is assigned to **physical drive 1**. Now I have assigned two logical drive letters (B and D) to what the computer knows as physical drive 1.

The switch value, /F:0, tells the computer that the new logical drive (D) is a 320/360K floppy drive.

Make sure that the CONFIG.SYS file has been saved by taking a directory of the DOS disk. If it is there, you are ready to test the new disk drive designations:

TESTING THE NEW DEVICE

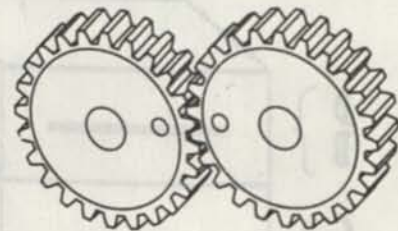
Turn off your computer. Then start your system with the DOS disk that contains the CONFIG.SYS file. Here is what our opening screen shows:

```

BIOS ROM version 01.03.00
Compatibility Software
Copyright (C) 1984,1985,1986,1987
Phoenix Software Associates Ltd.
and Tandy Corporation.
All rights reserved.

Microsoft MS-DOS Version 3.20
(C) Copyright Microsoft Corp 1981, 1986
Tandy version 03.20.21
Licensed to Tandy Corp.
All rights reserved.

Loaded External Disk Driver for Drive C
Loaded External Disk Driver for Drive D
Current date is Tue 1-01-1980
Enter new date (mm-dd-yy): _
  
```



← New driver from
CONFIG.SYS file

Notice the two additional lines just above the display of the current date. The CONFIG.SYS file causes the drivers for the two drives (C and D) to be loaded.

After entering the date and time, our DOS prompt reappeared. We put the disk containing BROWSE03.DOC in drive B and executed the following command to see if we could save the file to the same physical drive using the drive D designation:

```
A>COPY B:BROWSE03.DOC D:
```

```
Insert diskette for drive D: and strike  
any key when ready
```

*File is copied to memory from drive B—
then prompt to insert disk appears*

When a key is struck, the file is copied to the disk in drive D:

```
A>COPY B:BROWSE03.DOC D:
```

```
Insert diskette for drive D: and strike  
any key when ready
```

```
1 File(s) copied
```

```
A>_
```



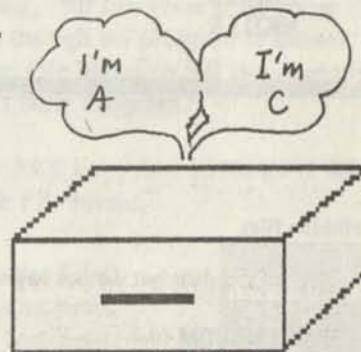
The same method can be used to save a file from drive A to drive C:

```
A>COPY CONFIG.SYS C:
```

```
Insert diskette for drive C: and strike  
any key when ready
```

```
1 File(s) copied
```

```
A>_
```



From time to time, *The BASIC Teacher* will include tips, suggestions, and uses of MS-DOS that are often neglected. If you have particular DOS features that you would like discussed, send suggestions to *The BASIC Teacher*.

DOS COMMANDS

The actions performed by DOS can be divided into two parts:

1. Resident Commands

Perform actions that DOS can always perform. Once DOS is loaded, these commands are always resident in memory. They can be executed immediately, whenever you want.

2. Transient Commands

Small programs, really, that perform special functions. If you want to perform any of these functions, you must have a disk containing these DOS commands in the computer. Transient commands are only brought into memory from a disk when you request them. It is your responsibility to have them available when needed.



The following table shows resident and transient DOS commands:

RESIDENT COMMANDS

BREAK	PATH
CHCP	PROMPT
CHDIR	RENAME
CLS	RMDIR
COPY	SET
CTTY	TIME
DATE	TYPE
DEL	VER
DIR	VERIFY
ERASE	VOL
MKDIR	

TRANSIENT COMMANDS

APPEND	FIND	RECOVER
ASSIGN	FORMAT	REPLACE
ATTRIB	GRAFTABL	RESTORE
BACKUP	GRAPHICS	SELECT
CHKDSK	JOIN	SHARE
COMP	KEYB	SORT
DISKCOMP	LABEL	SUBST
DISKCOPY	MODE	SYS
EXE2BIN	MORE	TREE
FASTOPEN	NLSFUNC	XCOPY
FDISK	PRINT	

DOS has two parts:

1. Hidden files

Stored on the disk but do not appear on a disk directory.

2. COMMAND.COM file

Visible on the disk directory, used to boot up the system.

The hidden files and the **COMMAND.COM** file are copied to a disk when it is **system formatted**. These files occupy almost 70K of memory as can be seen when a 5¼" disk is formatted.



A>format b:/s
Insert new diskette for drive B:
and strike ENTER when ready

Format complete
System transferred

362496 bytes total disk space
68688 bytes used by system
293808 bytes available on disk

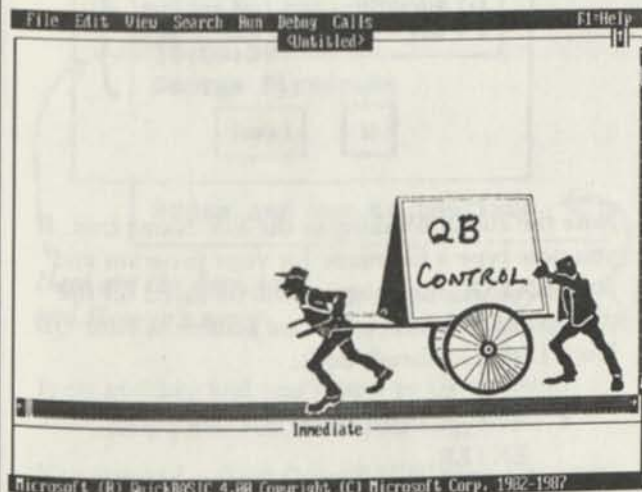
INTRODUCTION

QUICKBASIC IS a great language for beginners. In our small town, we'll give our time to help kids use QB to explore the **right stuff**, the most powerful uses of computers: simulations, games, simulation games, **Imitations of Life**. We will use real-life computers to help kids learn, teach, and explore math, science, and other disciplines. We call it **Computer Kid, USA**.

QuickBASIC is a great language for experts. We think it is the best language for developing **Educational ShareWare**. So, we are helping start a grass-roots effort to use **QB** as the common language for developing a cornucopia of excellent low-cost educational **ShareWare** for home or school use. If this piques your curiosity, read "The Shareware Teacher" column by Wendy Wiegand, Dean Brown, and Bob Albrecht in every issue of **ShareWare Magazine** from PC-SIG, 1030 East Duane Avenue, Suite D, Sunnyvale, CA 94086.

QB Control

WE ASSUME that you know how to load QB into the computer and be rewarded by the sight of a screen like the one shown below . . .



We call this screen the **QuickBASIC Control Center**, or just **QB Control**. It is also known as the **Edit Screen**. You compose and edit programs here, in the large window called the **View Window**.

The File Menu

IF YOU have just loaded QB, there is no QB program showing in the View Window. If there is a program showing, you can erase it by using the **File Menu**. Even though no program is shown, practice using the File Menu to tell the computer to get ready for a **New Program**.

- Press the ALT key, then press the F key to get the File Menu.

When you access the File Menu, the top menu item, **New Program**, is highlighted.

Good, That's the one we want.

Press the ENTER key.



The File Menu disappears. You are ready to enter a new program in the View Window.

Replaces currently loaded

A QB Program

FOR YOUR first QuickBASIC program, we suggest a program to print the date, time, and your name. While gazing at the Edit Screen, George Firedrake typed:

```
CLS
PRINT DATE$
PRINT TIME$
PRINT "George Firedrake"
```

George typed each line of his program and pressed the ENTER key. As he typed, his program appeared in the View Window and was also stored in the computer's memory, ready to be used.

Your turn. Go ahead and enter George's program in your View Window. If you wish, use your name instead of George's! If you type **cls** in lower case, QuickBASIC will change it to upper case when you press ENTER. The same for **print**, **date\$**, and **time\$**. These are QuickBASIC **keywords**. Type them any old way (but spell them correctly), and QB will format them as all caps. That's the way we want it, anyway.

Thanks QB.

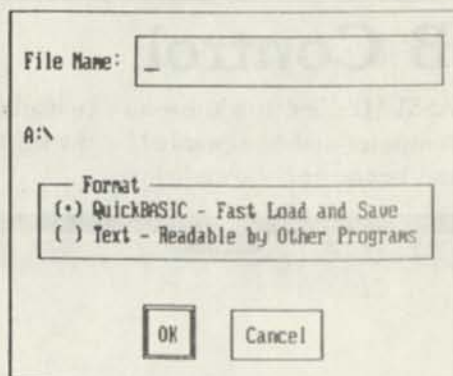


If you make a typing mistake, use the arrow keys, the BACKSPACE key, and the DELETE key to correct the error.

Save the Program

We think it is a real good idea to immediately save a program, even before you use it, print it, or otherwise it. To save a program to a disk, use the File Menu. Here's how:

- Press the ALT key, then press the F key. The File Menu "drops down," with **New Program** highlighted.
- Use the DOWN ARROW key to move the highlight down to **SAVE AS ...**. If you go too far, use the UP ARROW key to move the highlight up the FileMenu.
- When **Save As ...** is highlighted, press the ENTER key. The **Save As ...** Dialog Box pops into view:



Note the cursor blinking in the File Name box. If you now type a file name for your program and press ENTER, the program will be saved on the disk in disk drive A, which we assume is your QB Work Disk. Go ahead, do it:

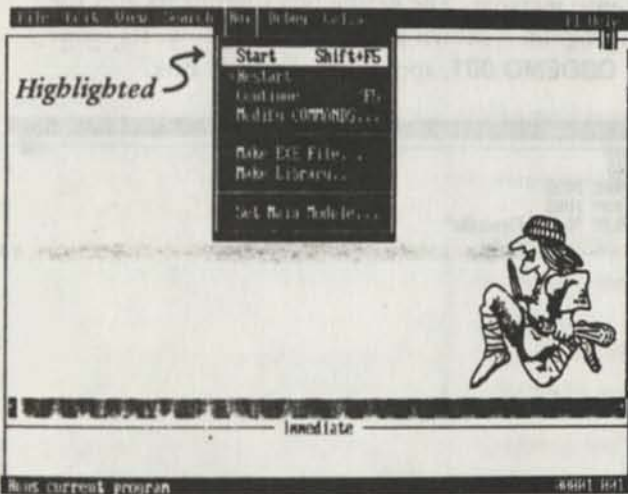
- We typed **QBDEMO.001** and pressed ENTER.

The disk drive light came on, the disk drive clicked a few times, and the **Save As ...** Dialog Box disappeared. We are back in QB Control. However, there is a small difference. In the top-middle part of the View Window, we see the name we gave the program: **QBDEMO.001**.

Run the Program

THE PROGRAM, now called **QBDEMO.001**, is stored on the QB Work Disk. Later, we'll go get it. But now, use the **Run Menu** to run the program.

- Press the ALT key, then press the R key. This highlights the word **Run** in the Menu Bar and also selects the Run Menu.



Ready to run? Do it:

- Press the ENTER key to run the program. Here is what we saw ...

07-04-1988
16:05:39
George Firedrake

Press any key to continue

Here are the date, time, and George's name.

Press any key to return to QB Control.

Press any key and you return to QB Control. Your program is still in the View Window.

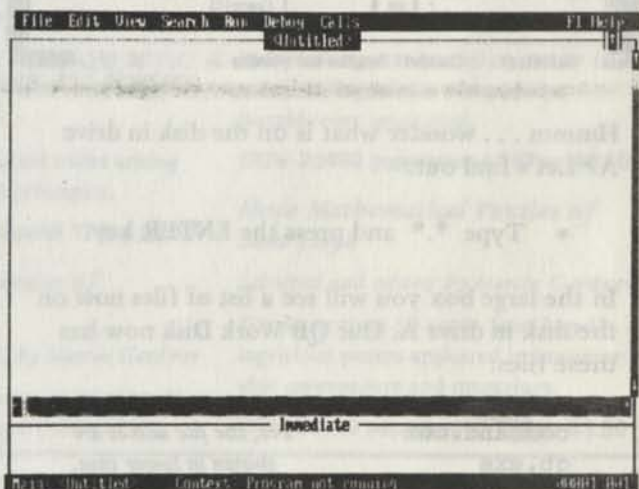
You created a short QuickBASIC program, saved it to the QB Work Disk, and ran it. Your program is still in the memory of the computer, also stored on the disk under the file name **QBDEMO.001**, and visible in the View Window.

Keep up the good work!

Erase the Program

NEXT, ERASE the program from the computer's memory and from the View Window. You can use the New Program selection in the File Menu to do this:

- Press the ALT key, then press the F key. You will see the File Menu with **New Program** highlighted.
- Press the ENTER key. The File Menu disappears and you return to QB Control. The program is gone from the View Window and the Title Box says: **Untitled**.



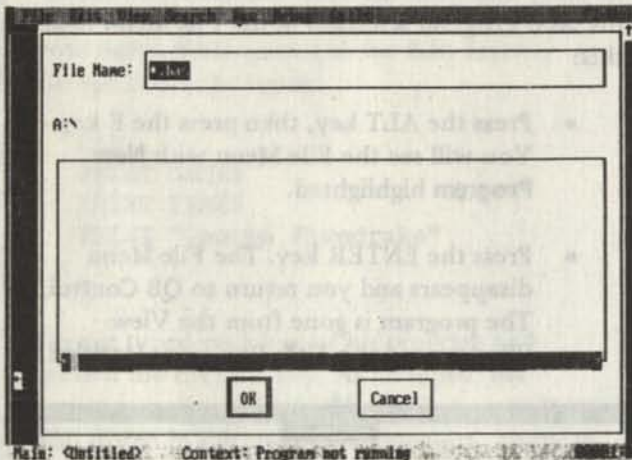
The program has been erased from memory and from the View Window, but is still stored on the disk.

What's on the Disk?

Use the File Menu to find out what is on a disk:

- Press the ALT key, then press the F key. You will see the File Menu with **New Program** highlighted. That's not what we want.
- Press the DOWN ARROW key once. The highlight moves down to **Open Program** ... Good! That's what we want.
- Press the ENTER key. You will see the **Open Program** ... Dialog Box.

The Open Program . . . Dialog Box looks like this:



Hmmm . . . wonder what is on the disk in drive A? Let's find out.

- Type *.* and press the ENTER key.

In the large box you will see a list of files now on the disk in drive A. Our QB Work Disk now has these files:

command.com
qb.exe
qb.hlp
qbdemo.001

Yes, the file names are shown in lower case, even if you typed them in upper case. That's OK.

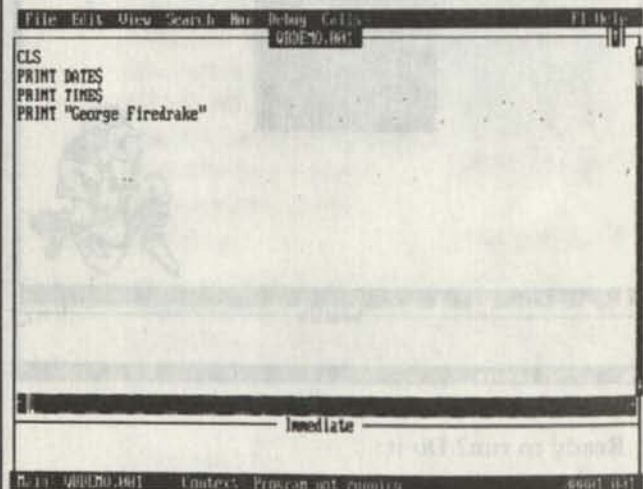
The first three files comprise the original QB Work Disk. The fourth file (**qbdemo.001**) was added this time.

Load QBDEMO.001

LOAD **qbdemo.001** from the disk into memory, as follows:

- Type **qbdemo.001** and press the ENTER key.

The light on drive A comes on, the disk drive whirrs & clicks a bit, and the program is loaded into memory. The dialog box disappears and the program is shown in the View Window. Its name, **QBDEMO.001**, appears in the Title Box.



Now use the Run Menu to run the program. Whenever you want to run the program known as **QBDEMO.001**, use the File Menu to load it from the disk into the computer's memory, then use the Run Menu to run it.

We Hope Everything Went As Planned



WELL, WE hope all the above happened when you did it. If so, you now know how to:

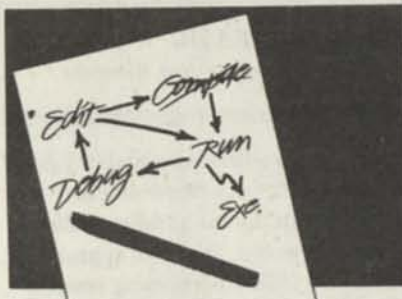
- Create a program in the View Window.
- Use the File Menu to save the program to disk drive A.
- Use the Run Menu to run the program.
- At the end of a run, press any key to return to QB Control.
- Use the File Menu to erase a program.
- Use the File Menu to list the files on the disk in drive A.
- Use the File Menu to load a program from drive A.

Postage & Handling: Add \$2.00 for first item and \$1.00 for each additional item. To shipping points outside U.S. add \$3.00 for first item and \$1.00 for each additional item. Shipped via surface mail.

PC Magazine
**Technical
 Excellence
 Award
 Winner**

Microsoft QuickBASIC 4.0

Microsoft



A powerful, full-featured programming language, QuickBASIC 4.0 takes BASIC into an entirely new dimension by adding source-level debugging, huge arrays, unlimited string space, support for Hercules graphics, and a wealth of other important features. The most impressive new feature of all is the threaded p-code interpreter. QuickBASIC 4.0 uses an incremental compiler that converts each line of source code as it is entered. What makes this system impressive is its ability to stop a program's execution, examine variables and make changes to the source code, and then resume execution. Further, QuickBASIC programs can now call routines written in any of the other Microsoft languages, and vice versa.

"Even the most cynical 'structure' fanatics and BASIC bashers must now agree that BASIC is a serious development language . . . BASIC has indeed come of age."

—Ethan Winter
PC Magazine

MS-11407 QuickBASIC 4.0 . . . \$95.00

ADVANCED COLOR GRAPHICS AND ANIMATION FOR THE IBM PC

By Don Inman and Kurt Inman

(Hayden, 248pp)

ACHIEVE SOPHISTICATED screen effects with this example-oriented text that shows you how to use the extended graphics capabilities of IBM BASICA for programming displays.

The ShareWare Book Using PC-Write, PC-File, PC-Talk

By Emil Flock, et al

(Osborne/McGraw-Hill, 688pp)

Covers the most popular "free" programs: PC-Write, a word processor; PC-File, a database manager; and PC-Talk, a telecommunications program. These programs are available thru user groups or bulletin-board services in return for a nominal registration fee. The book has all the details on how you can obtain these program disks.

OMH-881251 The Shareware Book . . . \$14.95

QuickBASIC: The Complete Reference

By Steven Nameroff

(Osborne/McGraw-Hill, 700pp)

Publication date pushed back to November '88. Please do not order this book until further notice.

Whether you're creating a business chart or an artistic animation, you'll learn the techniques that give your work professional results.

HAY-121 Advanced Color Graphics . . . \$18.95

Back Issues Available

TBT-1 Issue No. 1 . . . \$3.00
 TBT-1 Issue No. 2 . . . \$3.00



Different Worlds Publications

2814 - 19th Street
 San Francisco, CA 94110

Address Correction Requested

BULK RATE
 U.S. Postage
 P A I D
 San Francisco, CA
 Permit No. 11798



No. 4, October 1988

*For beginning
programmers with no prior
programming experience*

CONTENTS

- 3 Teach Yourself BASIC
- 11 Browsing BASIC
- 19 Teach Yourself
QuickBASIC

PUBLISHER'S STATEMENT: *The Teacher* is published monthly by Different Worlds Publications, 2814 - 19th Street, San Francisco, CA 94110. Contents copyright 1988 by Different Worlds Publications. All rights reserved. Contents may be copied and distributed freely. Address all correspondences to *The BASIC Teacher*, 2814 - 19th Street, San Francisco, CA 94110.

SUBSCRIPTION INFO: A 12-issue sub in the U.S. and Canada is \$36. Overseas subs are \$42 by surface mail, \$54 by air.

PRINTED IN THE U.S.A.

BASIC



PUBLIC DOMAIN AND SHAREWARE PROGRAMS

YOU CAN get many perfectly good programs for the price of the disk and cost of distribution alone. You pay registration fees only after you try the program out and either decide you've gotten some enjoyment out of it or if you decide to use it on a regular basis. The cost of registration is usually hundreds of dollars less and many shareware programs have been rated better than their comparable commercial versions.

You can get information on how to purchase public domain and shareware programs from:

PC-SIG	California Freeware
1030D E Duane Ave	1466 Springline Dr
Palo Alto, CA 94086	Palmdale, CA 93550
800-245-6717	(805) 273-0300
800-222-2996 in Calif.	

Micro Star
PO Box 4078
Leucadia, CA 92024
800-443-6103
800-443-7430 in Calif.

Innovative Technology
PO Box 726
Elk City, OK 73648
800-253-4001 ext. 78

Lone Star Software	Software Excitement!
2100 Hwy 360, Ste 1204	PO Box 3072
Grd Prairie, TX 75050	Ctl Pt, OR 97502
800-445-6172	800-444-5457

In the future *The BASIC Teacher* will offer many of the best, most enjoyable, and most useful shareware programs to its readers. If you know any public domain or shareware programs which would be of interest or of use to our readers, please let us know. If you have a program which you would like to share with our readers, send us an evaluation copy for possible distribution by *The BASIC Teacher*.

FAMILY COMPUTER FAIRE

November 26, 1988
RAF Mildenhall Air Force Base,
Suffolk, United Kingdom

Hosted by the Gateway Computer Club, 8 Church End, Brandon, Suffolk IP 27 0JE, United Kingdom, (0842) 814291. The Gateway Computer Club is a multi-computer association that supports Atari, Commodore, Amiga, Apple, Macintosh, IBM, Tandy, Sinclair, BBC, Amstrad, and MS-DOS type computers.



TANDY 1000 TL/SL

Tandy Corporation announces two new models for their personal-computer line. The Tandy 1000 TL features the popular 80286 microprocessor and the Tandy 1000 SL features the 8086 microprocessor. The TL comes with 640K RAM and one 3½" disk drive. The SL comes with 384K RAM and one 5¼" drive. The TL also comes with an internal clock/calendar.

Both models include MS-DOS 3.3 in ROM, a 101-key enhanced keyboard, a parallel and a serial port, and two joystick ports. They both also come with DeskMate 3.0, ten applications in one program: Text Processing, Worksheet, Electronic Filing, Draw, Address Book, Spell Checker, Hangman, PC-Link, and Calendar/Alarm.

Both models also come with sophisticated sound and speech capabilities. They include both a digital-to-analog converter and an analog-to-digital converter. You can record sound, edit it, and play it back. You can also compose and play music.

Available thru Radio Shack.



Introduction

KIDS CAN learn BASIC at an early age and use it to explore math, science, and other disciplines in the most powerful ways: by designing simulations, games, and simulation games . . . Imitations of Life. In our town, we'll do this in after-school courses for 10- to 13-year-old learners. If you are interested in this sort of thing, send a stamped, self-addressed envelope to ComputerKid, USA, PO Box 1635, Sebastopol, CA 95473.

BASIC has a small **vocabulary** and a simple **syntax** (grammar). We have already discussed some of the special words that Microsoft BASIC understands. They are called **keywords** or **reserved words**.

Here are the keywords introduced and described previously:

BEEP	CLS	COLOR	DATE\$	GOTO	KEY	LIST	NEW
OFF	ON	PRINT	RUN	TIME\$	TIMER	WIDTH	

We also used, but did not describe:

INT RND



Number Boxes

IMAGINE THAT, in the computer's memory, there are a bunch of **number boxes**. Each number box can hold one number at any one time. Each number box has a name. Here are some number boxes that we named:

a 7 b 5 c 36

We named these boxes **a**, **b**, and **c**. The number 7 is in box **a**; the number 5 is in box **b**; the number 36 is in box **c**.

Here are more number boxes with a number in each number box:

diameter 24 Price 29.95

(1) What number is in box **diameter**? _____

(2) What number is in box **Price**? _____

ANSWERS: (1) 24; (2) 29.95.

Numeric Variables

THE NAMES of number boxes are called **numeric variables**. A numeric variable can be a single letter or any combination of letters and digits (0 to 9) up to 40 characters in length. The first character must be a letter.

These are okay: **x R2D2**

These are not okay: **2D2R a.b**

The number in a number box is called the **value** of the **variable** that identifies the box. In the number boxes on this page:

The value of **b** is 5.

The value of **diameter** is 24.

BASIC treats all letters in variable names as upper-case letters, even if you type them in lower case. However we will usually show variables in lower case or a MiXtuRe of upper and lower case. Keywords will always be shown in all upper-case letters (BEEP, CLS, COLOR, and so on).

a = 7

HOW DO you put numbers in number boxes?
Easy!

- Clear the screen
- Type **a = 7** and press ENTER

```
a = 7
Ok
-
```

The computer has put the number 7 into box **a**.
Or, in more formal language, the computer has **assigned** the value 7 to the variable **a**.

a = 7
— assign this number
to this variable

The instruction: **a = 7**

Tells the computer to assign the value 7 to the variable **a**.

PRINT a

IS THE number 7 in number box **a**? Is 7 the value of **a**? Find out:

- Clear the screen
- Type **PRINT a** and press ENTER

```
PRINT a
7
Ok
-
```

The instruction:
PRINT a

Tells the computer to print the value of the variable **a**.



b = 5

TELL THE computer to assign 5 as the value of **b**, then print the value of **b**.

- Clear the screen
- You type **b = 5** and press ENTER
- You type **PRINT b** and press ENTER

```
b = 5
Ok
PRINT b
5
Ok
-
```

The value of **b** is now 5. The value of **a**, we assume, is still 7. You can type **PRINT a** to make sure. If not, go ahead and assign 7 as the value of **a** (**a = 7**).

Now add, subtract, multiply, and divide the numbers in **a** and **b**, as follows:

- You type **PRINT a + b**

It prints 12

- You type **PRINT a - b**

It prints 2

- You type **PRINT a * b**

It prints 35

- You type **PRINT a / b**

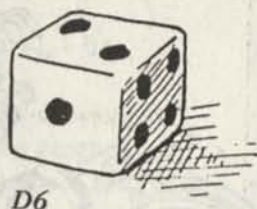
It prints 1.4

If your fingers didn't stumble, the screen probably looks like this:

```
PRINT a + b
12
Ok
PRINT a - b
2
Ok
PRINT a * b
35
Ok
PRINT a / b
1.4
Ok
-
```

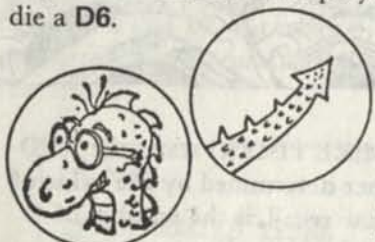

The Mysterious and Unpredictable RND

A **RANDOM** number is a number that is chosen at random from a given set of numbers. A die has six sides, numbered 1 thru 6. Roll one die to get a random number from 1 to



D6

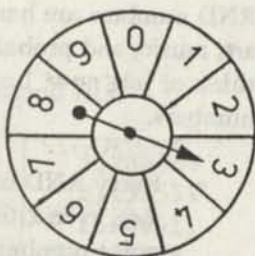
6. These numbers (1 to 6) are *equally probable*. Each number has the same chance of occurring as any other number. Game players call a six-sided die a **D6**.



Flip a coin—two possibilities: heads or tails. If it comes up **HEADS**, call it 0. If it comes up **TAILS**, call it 1. In

this way, flipping a coin gives a random number, 0 or 1. These numbers (0 or 1) are equally probable. Use two or more coins to generate random binary (base 2) numbers.

Use a spinner like the one shown at the right to spin a random decimal digit, 0 to 9. If the spinner is carefully designed and well-constructed, the possible values, 0 to 9, are equally probable. People who play fantasy role-



playing games use a special 10- or 20-sided die for rolling decimal digits. On the 20-sided die, each digit appears twice.



The Amazing Zocchihedron

They told him he couldn't do it, but he did it. Lou Zocchi created the first 100-sided die. Use it to make percentage rolls—equally probable numbers

from 1 to 100. Kids who play fantasy role-playing games love the Zocchihedron!

In BASIC, use the **RND** function to get random numbers. Well, perhaps we should call them **RND numbers** since they aren't quite the same as random numbers. The following tiny program computes and prints ten **RND** numbers on the screen. Note the use of the numeric variable **n** in lines 20 and 40:

```
10 CLS
```

```
20 FOR n = 1 TO 10
```

```
30 PRINT RND
```

```
40 NEXT n
```

This FOR...NEXT loop tells the computer to PRINT ten RND numbers

Enter and run this program. We ran the program twice. Here is what happened:

First Run

```
.1213501
.651861
.8688611
.7297625
.798853
7.369805E-02
.4903128
.1072496
.9505102
```

Ok

Second Run

```
.1213501
.651861
.8688611
.7297625
.798853
7.369805E-02
.4903128
.1072496
.9505102
```

Ok

Random?



Huh!? We got the same numbers both times. So we did it again and . . . got the same numbers again. What's going on here?

Well, you see, the RND function is a procedure, algorithm, or whatever, designed to generate numbers that we can use much like we would use actual random numbers. Fortunately, there is a way to start the RND function in places other than the beginning. It is called **RANDOMIZE**. We added a **RANDOMIZE** statement at line 15 to the original program.

```
10 CLS
15 RANDOMIZE TIMER
20 FOR n = 1 TO 10
30 PRINT RND
40 NEXT n
```



Here are two runs. These runs produced different sets of numbers.

First Run

```
.7321985
.1434736
.3590502
.5516148
.8935375
.7964193
.2393242
.6434587
.670008
.5610022
```

Ok

—

Second Run

```
.5176142
.3028516
.334041
.2672673
.8247051
.7685415
2.608264E-02
.1738084
.4070838
.3653998
```

Ok

—

That's better!



Line 15 (**RANDOMIZE TIMER**) starts the RND function at a number determined by the value of **TIMER** which, please recall, is the number of seconds since midnight, according to the computer. Of course, if you run this program every day at the exact same time, you will get the same bunch of numbers every day. Not too likely!

RND numbers are handy for games, simulations, art, music, and probably other things we can't think of just now. Look over the lists of RND numbers.

- Every RND number is **greater than zero**. Yes, even 2.608264E-02. This is a floating point number, described in "Teach Yourself BASIC No. 2."
- Every RND number is **less than one**. Another way to read 2.608264E-02 is 0.02608264.

From the evidence, it seems that an RND number is greater than zero and less than one. However we haven't shown much evidence, only a few numbers. We suggest you run a bunch more to get more evidence.

It's true. RND numbers are greater than zero and less than one. Another way to say it: RND numbers are **between zero and one**. Or, in yet another way:

$$0 < \text{RND} < 1$$

Unfortunately, that's usually not convenient. Usually we want **integers** in some range. For example, we might want the numbers 1, 2, 3, 4, 5, and 6 at random; or decimal digits 0 to 9; or numbers from 1 to 100.

Hmmm . . . RND is between 0 and 1, but is never 0 or 1. Therefore $10 * \text{RND}$ must be a number between 0 and 10, but never 0 and never 10. Do you agree? If not, run the following program a few times.

```
10 CLS
15 RANDOMIZE TIMER
20 FOR n = 1 TO 10
30   PRINT 10 * RND
40 NEXT n
```

Here are some examples:

First Run

```
4.029291
2.084068
9.266755
5.512639
7.852154
4.495108
2.956749
4.275316
4.584143
7.739081
```

Ok

Second Run

```
6.284968
2.139539
.6948411
6.54688
8.811932
9.428274
4.71975
9.298814
8.576147
8.575566
```

Ok

$$0 < 10 * \text{RND} < 10$$

Yup, all values of $10 * \text{RND}$ are greater than zero and less than ten. Each number has an **integer** part to the left of the decimal point and a **fractional** part to the right of the decimal point.

4.029291
integer part fractional part

9.298814
integer part fractional part

Perhaps you are wondering about .6948411. The integer part is zero (0).

For each number between 0 and 10, the integer part is a single decimal digit, 0 to 9. Wouldn't it be nice if you could tell the computer to throw away the fractional part and keep the integer part?

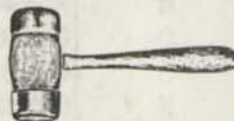
Well, you can. BASIC has a clever and useful function called **INT**. Here are some examples:

INT(3) is 3 **INT(9.298814)** is 9

INT(4.029291) is 4 **INT(.6948411)** is 0

You can use the computer to verify this:

- You type `PRINT INT(3)`
It prints 3
- You type `PRINT INT(4.029291)`
It prints 4



▼
4.029291

If **number** is any positive number or zero, then **INT(number)** is the integer part of **number**. Instead of a number, you can put any numeric variable, function, or expression in parentheses following **INT**.

INT (_____)

Any BASIC number, numeric variable, numeric function, or expression.

So it is OK to write **INT(10 * RND)**.

RND is a random number between 0 and 1.

10 * RND is a random number between 0 and 10.

INT(10 * RND) is a random integer from 0 to 9.

Now run this program to put random digits (0 to 9) on the screen.

```
10 CLS
15 RANDOMIZE TIMER
20 FOR n = 1 TO 10
30   PRINT INT(10 * RND)
40 NEXT n
```

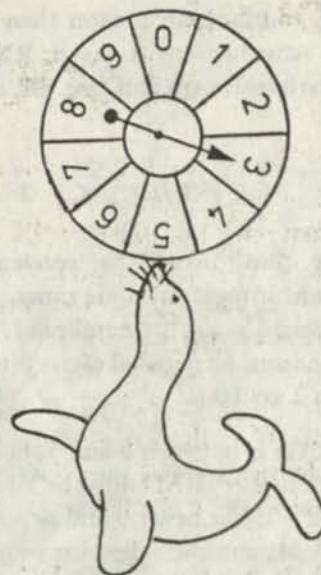
As usual, here are two samples:

First Run

```
8
0
0
4
4
9
6
7
8
2
Ok
—
```

Second Run

```
5
2
1
2
5
1
7
8
6
7
Ok
—
```



THINGS TO REMEMBER:

- **RND** is a number between 0 and 1. Never 0 and never 1.
- **2 * RND** is a number between 0 and 2. Never 0 and never 2.
- **3 * RND** is a number between 0 and 3. Never 0 and never 3.
- **6 * RND** is a number between 0 and 6. Never 0 and never 6.

INT(2 * RND) is 0 or 1.

INT(3 * RND) is 0, 1, or 2.

INT(6 * RND) is 0, 1, 2, 3, 4, or 5.

So you can add one to get numbers from 1 to 6, like rolling a D6 (six-sided die).

Write it like this: **INT(6 * RND) + 1**.

And so on. **RND** and **INT** are great tools for making computer games and simulations.



The Sound of SOUND

YOU CAN use the keyword BEEP to make the computer go beep. Every beep sounds the same ... the same **frequency** (pitch) and the same **duration** (length of time). Now use the keyword **SOUND** to make less monotonous sounds. You can make low sounds, high sounds, and in-between sounds. You can make short sounds and long sounds. The following **SOUND** instruction tells the computer to make a sound of frequency 262 Hertz (cycles per second) for a duration of 18 clock ticks (about one second).

SOUND 262, 18

frequency *duration*

Musicians call this sound middle C. Go ahead, play it on your computer:

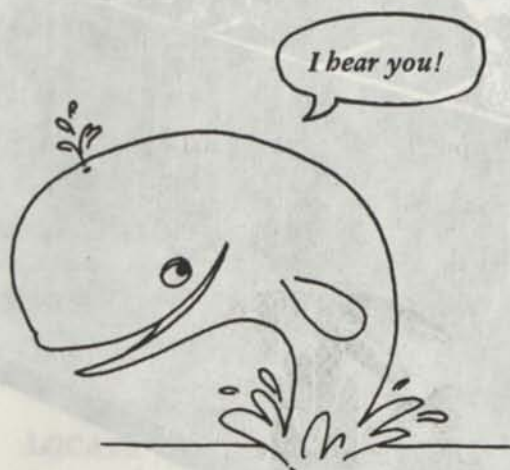
- Type **SOUND 262, 18** and press ENTER.

Now try a low sound. Use 37 as the frequency:

- Type **SOUND 37, 18** and press ENTER.

Then try the highest frequency the computer can use ... much too high for you to hear, unless you are a bat or a dolphin or a whale:

- Type **SOUND 32767, 18** and press ENTER.



EXPERIMENT: You pick the frequency number and the duration number. After experimenting for a while, enter and run the following program:

```
10 CLS
20 SOUND 262, 9
30 SOUND 440, 9
40 GOTO 20
```

What do you hear? A fire truck or ambulance rushing to the rescue? Two sounds, with frequencies 262 and 440, alternate until you use CTRL+BREAK to interrupt. You hear each sound for 9 ticks, about half a second.

Add a third sound, as in the following program:

```
10 CLS
20 SOUND 262, 9
30 SOUND 440, 9
40 SOUND 349, 9
50 GOTO 20
```

Go ahead, make some music. The frequency can be any number from 37 to 32767, but feel free to try numbers outside this range to see what happens. Here is a table of frequency numbers for some well-known musical notes.

Note	Frequency	Note	Frequency
C	130.81	C	523.25
D	146.83	D	587.33
E	164.81	E	659.26
F	174.61	F	698.46
G	196.00	G	783.99
A	220.00	A	880.00
B	246.94	B	987.77
C	261.63	C	1046.50
D	293.66	D	1174.70
E	329.63	E	1318.50
F	349.23	F	1396.90
G	392.00	G	1568.00
A	440.00	A	1760.00
B	493.88	B	1975.50

Strange Music

YOU CAN use **RND** to make strange "music." To hear some, enter and run this program:

```
10 CLS
20 frequency = 2000 * RND
30 duration = 1
40 SOUND frequency, duration
50 GOTO 20
```

The statement: **frequency = 2000 * RND**

Tells the computer to assign a random number between 0 and 2000 as the value of the numeric variable **frequency**. If you get an error message while running this program, change line 20 to:
20 frequency = 2000 * RND + 37.

The statement: **duration = 1**

Tells the computer to assign the value 1 to the numeric variable **duration**.

The statement: **SOUND frequency, duration**

Tells the computer to sound a tone using the values of the numeric variables **frequency** and **duration**.

Run the program, listen for a little while, then use **CTRL+BREAK** to stop it. Change the duration to 0.125 as follows:

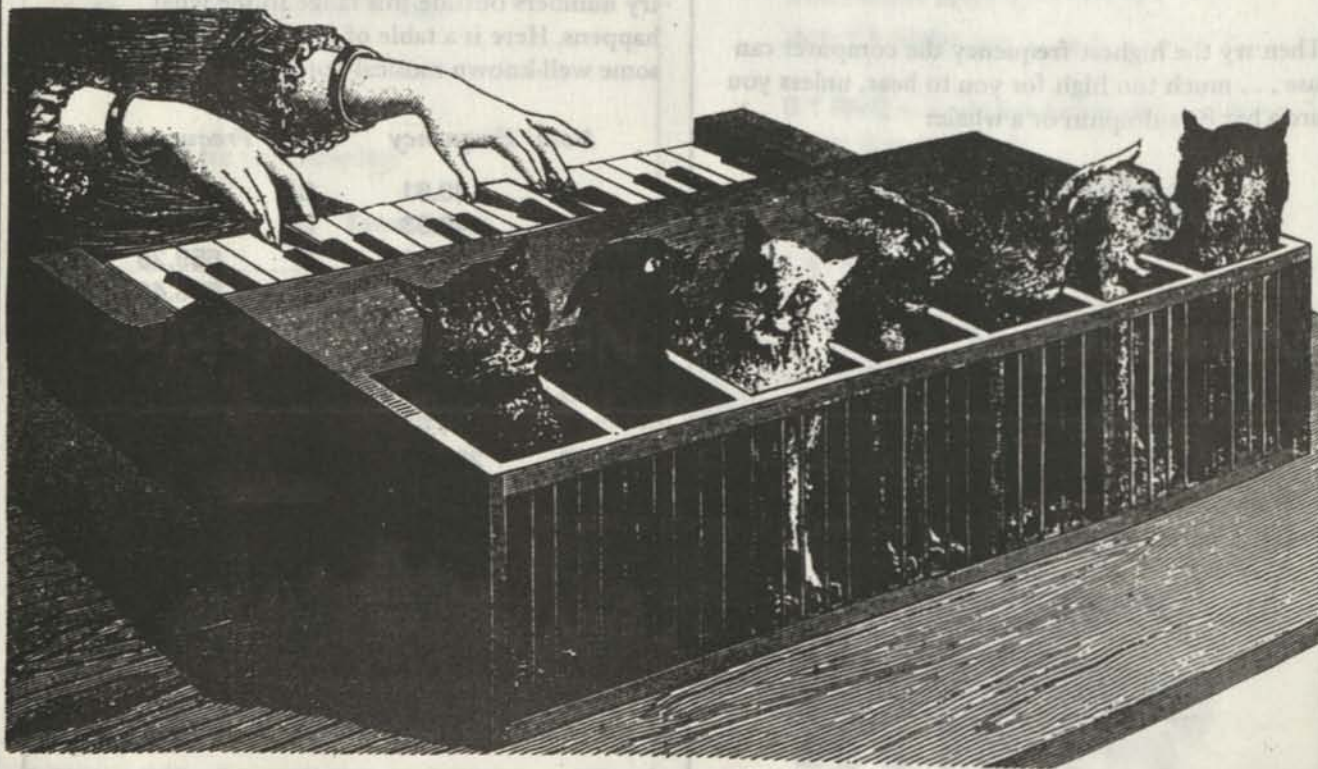
- Type **30 duration = 0.125** and press **ENTER**.

This changes only line 30. **LIST** the program to see your change:

```
LIST
10 CLS
20 FREQUENCY = 2000 * RND
30 DURATION = .125
40 SOUND FREQUENCY, DURATION
50 GOTO 20
Ok
```

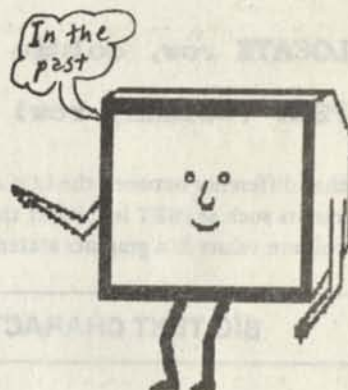
Here is the change

Yes, when you **LIST** a program, the computer shows variables in all upper case, no matter how you typed them. Nevertheless, we will show variables in lower case or a mixture of upper and lower case . . . makes programs a little easier to read, we think.



THE FOLLOWING topics were covered in previous installments of "Browsing BASIC":

- No. 1:** Browsed the **KEY** command and used it to change the function assignments shown on the **key line**. We also looked at the **ALT** key shortcuts for many BASIC keywords.
- No. 2:** Browsed the **VIEW PRINT** statement and experimented with scrolling and fixed portions on the screen. Then we looked at pull-down windows.
- No. 3:** Browsed the **COPY CON** command to write a **DRIVER.SYS** file to assign dual-drive specifications so that a file could be copied to the same physical disk. We also labeled disks.

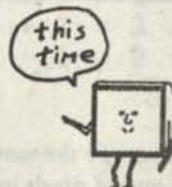


We started in April with a small list of keywords that we assumed you knew how to use. The list grew a little in April, a little more in May, and a little more in June. Here is a list of BASIC and DOS keywords that you should now know:

CHR\$	END	LINE INPUT	SAVE
CLS	FOR...NEXT	LIST	SCREEN(row, col)
COLOR	FORMAT	LOAD	SCREEN 0
COPY	GOSUB	LOCATE	SPACE\$
DATA	GOTO	OR	STRING\$
DEVICE	IF...THEN	PRINT	TAB
DIM	INPUT	READ	VIEW PRINT
DIR	INPUT\$	REM	WIDTH
DISKCOPY	KEY	RETURN	
DRIVER	LABEL	RUN	

This month we will add these keywords to the list:

INPUT	LINE	PSET
LEN	MID\$	SCREEN 1



TEXT AND GRAPHIC POSITION RELATIONSHIPS

THE FIRST two "Browsing BASIC" articles discussed topics that are used in the text screen mode (SCREEN 0). We will now browse the use of text, along with graphics, on a graphics screen (SCREEN 1).

Quite often you will want to add some text to a graphics screen. For instance, if you create a graph, you will want to add some text to describe what the graph represents. You will want to display labels, scales, and a title for the graph.

You have used the **LOCATE** statement to position characters to be printed on the display.

LOCATE row, column

Notice the order
used to specify
row and column

ORDER, ORDER!

Unfortunately when BASIC locates graphic pixels, it reverses the order of the row and column values used in positioning text characters. This difference in order is seen when comparing a **LOCATE** (text character) statement with a **PSET** (graphics pixel) statement:

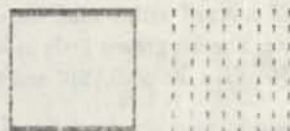
LOCATE row, column *row first, then column*

PSET (column, row) *column first, then row*

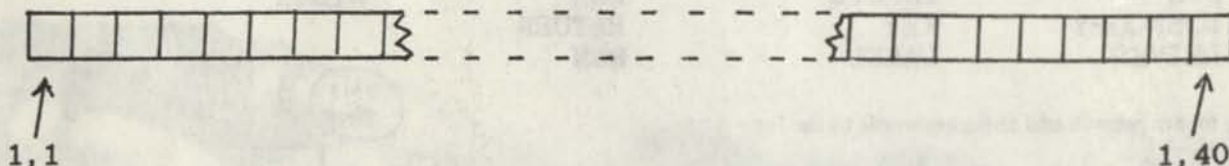
Another difference between the **LOCATE** statement and graphics location statements such as **PSET** is the fact that parentheses are used to enclose the row and column values in a graphics statement.

BIG TEXT CHARACTER, BUT SMALL PIXEL

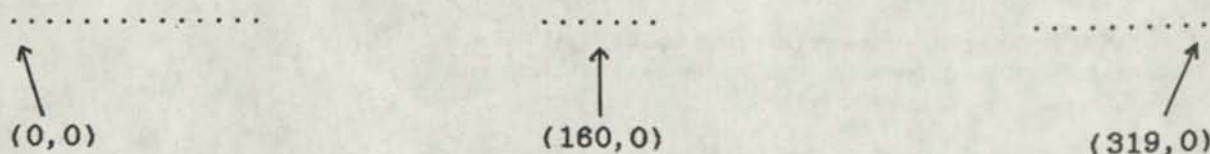
A graphics pixel set by a **PSET** statement is much smaller than a text character. Each text character occupies the same space as an eight by eight array of pixels. Therefore the number of a pixel position in graphics does not correspond to the same number used for a text position.



We will use **SCREEN 1** to demonstrate the differences in text and graphics numbering systems. **SCREEN 1** is called the **medium resolution graphics mode**. The text size used in this mode uses a maximum of 40 characters per line. The top text line of the screen is specified as row 1. The left-most character position in a line is specified as column 1. Thus the top line consists of position 1,1 through 1,40.



Since eight graphics pixels occupy the same width as one text character, there are 8 times 40 or 320 columns of pixels in each graphics row. The top graphics row is specified as row 0 (zero). The left-most pixel position in a row is specified as column 0 (zero). The top row of graphics pixels consists of positions (0,0) through (319,0).

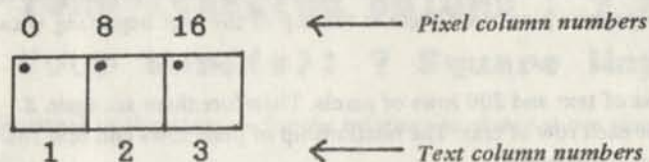


Not only are the **unit sizes** different for text and graphics, but the origin for numbering units is also different: (1,1) for text and (0,0) for graphics. Of course, you must also remember that the order of specifying row and column are reversed for text and graphics.

You can use a method of trial and error to find the correct **LOCATE** statement that will place the text where you want it in relation to the graphics. However, there is a direct relationship between text positions and graphics positions which can be used to calculate corresponding positions. Let's look first at column relationships.

CORRELATING COLUMN POSITIONS

Starting with the left-most column position, we have the following relationship between the first three text columns and the graphics columns that would correspond to the upper-left corners of the text positions:



You can see that for each increase of one text column, the corresponding pixel column (upper-left corner of the text position) increases by eight. However the pixel column numbering begins with one less than the corresponding text column. Therefore to locate the pixel position of the upper-left corner of the corresponding text position, you must do the following arithmetic operation:

$$1. \text{ PixelColumn} = 8 * (\text{TextColumn} - 1)$$

Examples:

For TextColumn = 1: PixelColumn = $8 * (1 - 1) = 8 * 0 = 0$
 For TextColumn = 2: PixelColumn = $8 * (2 - 1) = 8 * 1 = 8$
 For TextColumn = 3: PixelColumn = $8 * (3 - 1) = 8 * 2 = 16$

The converse relationship is:

$$2. \text{ TextColumn} = \text{PixelColumn} / 8 + 1$$

Examples:

For PixelColumn = 0: TextColumn = $0 / 8 + 1 = 0 + 1 = 1$
 For PixelColumn = 8: TextColumn = $8 / 8 + 1 = 1 + 1 = 2$
 For PixelColumn = 16: TextColumn = $16 / 8 + 1 = 2 + 1 = 3$

From Equation 1: PSET(PixelColumn, PixelRow) is the same as:

$$\text{PSET}(8 * (\text{TextColumn} - 1), \text{PixelRow})$$

From Equation 2: LOCATE TextRow, TextColumn is the same as:

$$\text{LOCATE TextRow, PixelColumn} / 8 + 1$$

CORRELATING ROW POSITIONS

Now that we have solved the column relationships, let's look at how corresponding row positions can be calculated.

Pixel Text

0	1
8	2
16	3

⋮

176	23
184	24
192	25

Once again, text row positions are numbered from the top of the page beginning with number 1. Graphics pixel rows begin at the top of the page beginning with number 0 (zero).

There are 25 rows of text and 200 rows of pixels. Therefore there are again 8 rows of pixels for each row of text. The relationship of pixel rows and text rows is the same as that for columns.

$$\text{PixelRow} = 8 * (\text{TextRow} - 1)$$

$$\text{TextRow} = \text{PixelRow} / 8 + 1$$

Examples:

```

For TextRow 1: PixelRow = 8 * (1 - 1) = 0
For TextRow 21: PixelRow = 8 * (21 - 1) = 160
For TextRow 25: PixelRow = 8 * (25 - 1) = 192
For PixelRow 0: TextRow = 0 / 8 + 1 = 1
For PixelRow 80: TextRow = 80 / 8 + 1 = 11
For PixelRow 192: TextRow = 192 / 8 + 1 = 25
  
```

That's enough heavy thinking for now. From here on, let's let the computer do the calculating.

PUTTING PIXELS AND TEXT TOGETHER

Since you know a character and an eight by eight array of pixels occupy the same area, you can draw a square around a character. The square should be a little larger than eight by eight to give a nice, clean appearance. You can use the relationships we have been discussing:

```

pcol = 8 * (tcol - 1)
prow = 8 * (trow - 1)
  
```

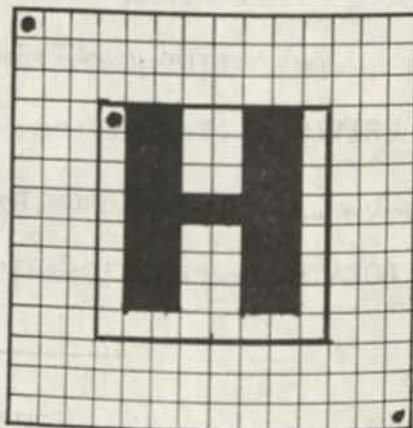
where:

```

pcol = PixelColumn
prow = PixelRow
tcol = TextColumn
trow = TextRow
  
```

To provide the space around the character, draw a rectangle whose upper-left corner is three pixels left and three pixels above the upper-left corner of the character.

The lower-right corner of the rectangle would be at position $pcol + 7 + 3$, $prow + 7 + 3$. Thus the upper-left and upper-right corners of the rectangle would be at $(pcol - 3, prow - 3)$ and $(pcol + 10, prow + 10)$, respectively.



You could have the computer print a string of letters, each enclosed in a rectangle. To do so you would have to tell the computer where to start printing the string (row and column). Of course, you would also have to tell it what string you want to print. The input to such a program could look like this:

Text starting row : ? 12
Text starting column : ? 2

Your word(s): ? Square Words

The output of the program, for the information shown above would look like this:

S **q** **u** **a** **r** **e** **W** **o** **r** **d** **s**

```

1 REM ** Squares Around Letters **
2 ' Browsing BASIC #4 5/15/88
3 ' Microsoft GW-BASIC File: FNCYWORD.001

100 REM ** Set Graphics Screen and Get Information **
110 SCREEN 1: CLS: KEY OFF
120 INPUT "Text starting row : "; trow%
130 INPUT "Text starting column : "; txtcol%
140 PRINT: INPUT "Your word(s): "; Strng$

200 REM ** Clear Screen and Print Words **
210 CLS: prow% = 8 * (trow% - 1)
220 FOR num% = 1 TO LEN(Strng$)
230   tcol% = txtcol% + 3 * (num% - 1)
240   LOCATE trow%, tcol%: Letter$ = MID$(Strng$, num%, 1)
250   PRINT Letter$
260   IF Letter$ <> " " THEN GOSUB 1010
270 NEXT num%

300 REM ** Wait for Keypress, Restore Screens **
310 ky% = INPUT$(1)
320 CLS: SCREEN 0: CLS: KEY ON: WIDTH 80
330 END

1000 REM ** SUBROUTINE: Square the Letters **
1010 pcol% = 8 * (tcol% - 1)
1020 LINE (pcol% - 3, prow% - 3)-(pcol% + 10, prow% + 10), , B
1030 RETURN

```

Run the Squares Around Letters program with your own entries. The program is designed to print on a single line, so start your words near the left side of the screen. No provision is made for a line feed. Therefore, you will get the message "Illegal function call in 240" if the line extends beyond text column 40.

After you make the initial entries, the screen is cleared and the pixel row position is calculated at line 210 by our trusty text row to pixel row relationship.

```
210 CLS: prow% = 8 * (trow% - 1)
```

The text column position (tcol%) moves three places to the right for each letter in the string you entered so that there will be plenty of space between the enclosed letters.

```
220 FOR num% = 1 TO LEN(Strng$)
230   tcol% = txtcol% + 3 * (num% - 1)
```

We used a new keyword (**LEN**) in line 220. The **LEN** function returns the number of characters in a string. Therefore if you enter a string (Strng\$) of 12 characters, **LEN(Strng\$)** = 12.

The computer examines your string letter by letter by means of **MID\$**, another new keyword.

```
220 FOR num% = 1 TO LEN(Strng$)
230   tcol% = txtcol% + 3 * (num% - 1)
240   LOCATE trow%, tcol%: Letter$ = MID$(Strng$, num%, 1)
```

The **MID\$** function returns a substring of a string. In this case, the string is Strng\$. The substring returned starts at the position num%. The number just in front of the right parenthesis tells how many characters to return.

The value of num%, in the FOR . . . NEXT loop, starts at one and increases each time from your string starting from the left-most character and assigns it to the variable Letter\$.

For example: Strng\$ = "Word"

When num% = 1,	W o r d	Letter\$ = W
When num% = 2,	W o r d	Letter\$ = o
When num% = 3,	W o r d	Letter\$ = r
When num% = 4,	W o r d	Letter\$ = d



So LEN(Strng\$) finds the length of the string you enter. This length is the upper limit of a FOR . . . NEXT loop that uses MID\$(Strng\$, num%, 1) to pick out one letter of the string each time through the loop.

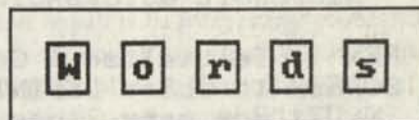
When a character is "picked off," it is printed. If the character is a space, no square is drawn. If the character is not a space, a subroutine is called to draw a rectangle around the character.

```

250 PRINT Letter$
260 IF Letter$ <> " " THEN GOSUB 1010
270 NEXT num%

1000 REM ** SUBROUTINE: Square the Letters **
1010 pcol% = 8 * (tcol% - 1)
1020 LINE (pcol% - 3, prow% - 3)-(pcol% + 10, prow% + 10), , B
1030 RETURN

```



We sneaked in another new BASIC keyword in the subroutine. The rectangles could be drawn by setting many points with the PSET statement. However, it is quicker to draw a straight line made up of many pixels with a **LINE** statement. All you need for the LINE statement are the two end points.

LINE (PixelColumn1, PixelRow1)-(PixelColumn2, PixelRow2)

starting point

ending point

We even went a little farther and used the **B option** (Box) with the LINE statement in line 1020 of the subroutine.

```
1020 LINE (pcol% - 3, prow% - 3)-(pcol% + 10, prow% + 10), , B
```

The B option at the end of the LINE statement tells the computer to draw a box using the two specified points as the opposite corners. The B option saves drawing four separate lines to form the box. It is all done in one giant step.

There are many ways you could enhance the Square Around Letters program. We used an INPUT statement to enter the test string. If you used LINE INPUT, you could also use punctuation characters in your string. You could also add features to provide a line feed and carriage return when you reach the end of a line. You would probably want to add two line feeds to provide enough space between lines to draw the rectangles without interfering with adjacent lines. Then you could write a whole screenful of text.



Here is a QuickBASIC program that goes a little beyond the Squares Around Letters program. This program draws a rectangle around words—even several lines of words. I'm sure you GW-BASIC folks can translate it into your own language:

```

REM ** General Rectangle/Text Demonstration **
' Browsing BASIC #4 8/24/88
' Microsoft QuickBASIC File: FNCYWORD.002

REM ** Initialize & Get Data **
SCREEN 1: CLS : DEFINT A-Z
INPUT "How many lines of text "; LineNum
INPUT "Row number for first line "; trow
INPUT "Column number for first line "; tcol
REDIM Array(1 TO LineNum) AS STRING
Lng = 0
FOR num = 1 TO UBOUND(Array$)
  PRINT "Enter line number"; num
  LINE INPUT Array$(num)
  temp = LEN(Array$(num))
  IF temp > Lng THEN
    Lng = temp
  END IF
NEXT num
CLS

REM ** Print Array & Box **
FOR lyne = 1 TO UBOUND(Array$)
  LOCATE trow + lyne - 1, tcol: PRINT Array$(lyne)
NEXT lyne
pcol = (tcol - 2) * 8: prw1 = (trow - 2) * 8
pco2 = (tcol + Lng) * 8: prw2 = (trow + LineNum) * 8
LINE (pcol, prw1)-(pco2, prw2), , B

REM ** Wait for Keypress & End **
a$ = INPUT$(1): CLS : END

```

**Wrap a rectangle
around me, please.**

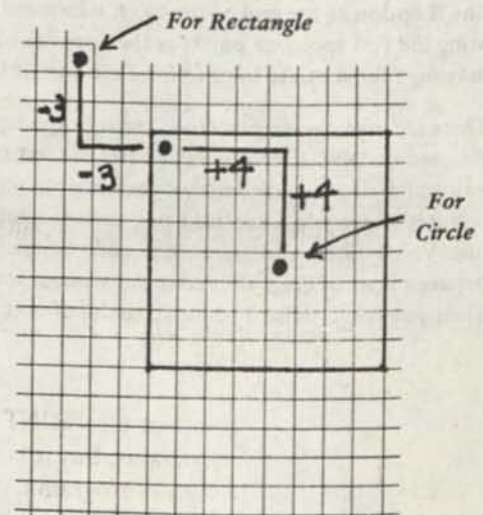
Put your thinking caps on for a minute. If you can draw rectangles around a text character, there is no reason you couldn't draw a circle around a character. To do this, you would have to reconsider the text/pixel relationship we have been using. To draw a circle around a text character, its center should be near the center of the text character. For a circle whose radius is 10, the following scheme would work:

*For Upper-Left Corner
of a Rectangle*

pcol% - 3
prw% - 3

*For Center
of a Circle*

pcol% + 4
prw% + 4



We'll continue this subject next time, showing how to use text with graphs to provide titles, scales, and labels. If you have graphic topics you would like to have discussed, drop us a line at *The BASIC Teacher*.



Computer Kid, USA

THE MOST powerful way to use a computer is to learn a general-purpose programming language and apply it to interesting problems. **QuickBASIC** is the best language for learning and teaching this high-level skill. Fifth or sixth grade is a good place to start. Serendipity! QuickBASIC has a very capable built-in word processor. Kids can learn the low-level skill of word processing as a by-product of learning the high-level skill of programming. And, since QuickBASIC works in a **Windows** environment, another by-product is learning how to use windows, pull-down menus, and other things useful on real-life computers, the 25 million IBM compatibles. Yet another bonus, another by-product of learning QuickBASIC: the tools of file management. Unfortunately schools don't teach real-life languages on real-life computers, so we have to do it outside of school with projects like **Computer Kid, USA** in our home town of Sebastopol, California. You can do it in your town. For more information:

Computer Kid, USA, PO Box 1635, Sebastopol, CA 95473, USA.

DO . . . LOOP

The DO . . . LOOP is a fundamental **control structure** in QuickBASIC. Go now to QB Control and enter the following tiny program in the View Window. Two versions of the program are shown, one for the Tandy 1000 series of computers, another for most other IBM PC compatibles.

If you are using an IBM PC or compatible (except Tandy 1000), enter the following program:

```
DO
  PRINT "Hold down CTRL and press BREAK"
LOOP
```

If you are using one of the Tandy 1000 computers, enter the following program:

```
DO
  PRINT "Hold down CTRL and press HOLD"
LOOP
```

It is not necessary to indent the PRINT statement as shown in the two programs, but it is very good practice to do so. It is good programming style. Programs written in good programming style are easy to read and understand.

Run the Program

Use the Run Menu to run the program. Zip . . . quickly the screen fills up with the message in the PRINT statement. Look at the bottom line on the screen. The bottom copy of the message seems to blink. Actually a new copy of the message is being printed many, many times a second. The new message printed at the bottom of the screen "pushes" the other copies up one line . . . this is called **scrolling**. The top copy is "pushed off the screen." This happens so fast, though, that only superheroes with ultra-fast eyes can see it happen.

The screen looks like this:

```
Hold down CTRL and press BREAK
Hold down CTRL and press BREAK
Hold down CTRL and press BREAK
Hold down CTRL and press BREAK
.
.
.
Hold down CTRL and press BREAK
Hold down CTRL and press BREAK
```

The bottom line is a
little blurry.

STOP the Program

Stop the program. For most IBM compatibles, hold down the CTRL key and press the BREAK key. For a Tandy 1000, hold down the CTRL key and press the HOLD key. You will be rewarded with the screen shown below:

```
File Edit View Search Run Debug Calls
<Untitled>
DO
PRINT "Hold down CTRL and press BREAK"
LOOP
```

This is highlighted

0.0

When you interrupt a DO . . . LOOP by using CTRL BREAK or CTRL HOLD, you are returned to QB Control. The instruction being executed when you interrupted will be highlighted. On our computer, it appears in a bright white, whiter than the rest of the text. In this short program, it is likely that LOOP will be highlighted. You will also see the cursor at the first letter of the highlighted word.

This is a nice feature of QuickBASIC . . . it tells you what it was doing when you interrupted.

The program in the View Window is a very simple example of a DO . . . LOOP. The instruction between DO and LOOP is repeated until you interrupt.

PRINT TIMES\$

Another DO . . . LOOP is shown in the following tiny program, which repeatedly prints the time until you interrupt:

```
DO
PRINT TIMES$
LOOP
```

Enter and run this program, then watch the screen. You will notice the seconds changing, a second at a time, of course. Watch for a while and see the minutes change. If you watch long enough, the hours will change.

PRINT TIMER

QuickBASIC has a keyword called TIMER that keeps track of the number of seconds since midnight. Enter and run the following tiny program to see TIMER in action . . . lots of action because TIMER shows hundredths of seconds.

```
DO
PRINT TIMER
LOOP
```

Freeze the screen. You can freeze the screen, stop the scrolling and look at the now quiescent number, then unfreeze the screen and let the seconds fly by.

To freeze the screen:

Tandy 1000: Press the HOLD key.

IBM PC: Hold down CTRL and press S.

To unfreeze the screen:

Tandy 1000: Press the HOLD key.

IBM PC: Hold down CTRL and press S.

PRINT DATES\$

The following DO . . . LOOP will print the date repeatedly on the screen.

```
DO
PRINT DATES$
LOOP
```

Of course, the date changes rather slowly, so you must have great patience to notice anything happening.



PRINT "Mariko" in Many COLORS

Enter and run all three of the following programs. All three programs cause Mariko's name to be printed in randomly selected colors from 16 available colors, including black, white, and very white.

```
DO
  COLOR INT(16*RND)
  PRINT "Mariko"
LOOP
```

```
DO
  COLOR INT(16*RND)
  PRINT "Mariko",
LOOP
```

```
DO
  COLOR INT(16*RND)
  PRINT "Mariko";
LOOP
```



The three programs are the same except for the comma (,) at the end of the PRINT instruction in the second program and the (;) at the end of the PRINT instruction in the third program. The first program has neither a comma nor a semicolon. Run all three programs to see the effect of the comma and semicolon.

```
COLOR INT(16*RND)
```

picks a random color from 0 to 15.

Color Number	Color	Color Number	Color
0	black	8	gray
1	blue	9	light blue
2	green	10	light green
3	cyan	11	light cyan
4	red	12	light red
5	magenta	13	light magenta
6	brown	14	yellow
7	white	15	bright white

RND? INT?

RND is a random number between 0 and 1, but is never exactly 0 nor 1. Could be, for example: 0.3760942. RND is greater than zero, but less than one.

$$0 < \text{RND} < 1$$

$16 * \text{RND}$ is a random number between 0 and 16, but is never exactly 0 or 16. If RND happens to be 0.3760942, then $16 * \text{RND}$ will be 6.017507. Yes, $16 * \text{RND}$ is greater than zero and less than 16.

$$0 < 16 * \text{RND} < 16$$

$\text{INT}(16 * \text{RND})$ is an integer from 0 to 15. It can be 0 or 1 or 2 or 3... and so on, up to 15. $\text{INT}(16 * \text{RND})$ is an integer greater than or equal to zero, but less than or equal to 15.

$$0 \leq \text{INT}(16 * \text{RND}) \leq 15$$

Be sure to look up COLOR, INT, and RND in the book, *BASIC Language Reference* that comes in the box with QuickBASIC.

Make It Blink

You can use color numbers 16 to 31 to make printed information blink. Use 16 for blinking black which, of course, you can't see on the black screen. Use 17 for blinking blue, 18 for blinking green, and so on. To see *only* blinking colors, run this program:

```
DO
  COLOR INT(16*RND) + 16
  PRINT "Mariko ";
LOOP
```

To see both blinking and non-blinking colors, use this program:

```
DO
  COLOR INT(32*RND)
  PRINT "Mariko ";
LOOP
```


SOUND Advice

You can use the keyword **BEEP** to make the computer go beep. Every beep sounds the same . . . the same **frequency** (pitch) and the same **duration** (length of time). You can make less monotonous sounds by using the keyword **SOUND**, which lets you control the frequency and duration of the sound. The following **SOUND** instruction causes the computer to play the note musicians call Middle C for about one second.

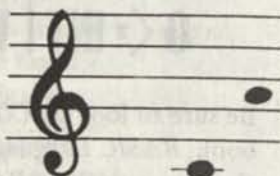
SOUND 262, 18

frequency

duration

If you would like to hear Middle C for about one second, go to the Immediate Window and enter the above **SOUND** instruction as an immediate command. Then enter the following tiny program, which plays two notes alternately until you interrupt by pressing **CTRL BREAK** or **CTRL HOLD**.

```
DO
  SOUND 262, 9
  SOUND 440, 9
LOOP
```



The instruction: **SOUND 262, 9**

tells the computer to sound a tone of frequency

262 Hertz (cycles per second) for 9 clock ticks, about half a second. This tone is Middle C.

The instruction: **SOUND 440, 9**

tells the computer to sound a tone of frequency 440 Hz (Hertz, cycles per second) for 9 clock ticks, about half a second. This tone is A, above Middle C.

Go ahead, make some music. The frequency can be any number from 37 to 32767. If you can hear those high notes, you must be a dolphin. The duration may be any number from 0 to 65535. Here is a table of frequency numbers for some well-known musical notes.

Note	Frequency	Note	Frequency
C	130.81	C	523.25
D	146.83	D	587.33
E	164.81	E	659.26
F	174.61	F	698.46
G	196.00	G	783.99
A	220.00	A	880.00
B	246.94	B	987.77
C	261.63	C	1046.50
D	293.66	D	1174.70
E	329.63	E	1318.50
F	349.23	F	1396.90
G	392.00	G	1568.00
A	440.00	A	1760.00
B	493.88	B	1975.50



QB 4.0 Work Disk

STILL AVAILABLE!

LAST ISSUE *The BASIC Teacher* announced that it has received authorization from Microsoft to make available to teachers and educators a limited quantity of QuickBASIC 4.0 Work Disks at a special low price. The Work Disks, which contain the two most-important QuickBASIC files, QB.EXE and QB.HLP, are still available.

Requests should be made on school letterheads. The disks may not be duplicated and may be used only by the person making request. Recipients' names and addresses will be made available to Microsoft.

Educators, this is the least-expensive way for you to get a copy of this exciting new computer language that is revolutionizing the way people use computers. Take advantage now while supplies last in this special offer.

MS-QBWD QuickBASIC
Work Disk \$12.00

Using QuickBASIC

By Don Inman and Bob Albrecht

(Osborne/McGraw-Hill, 436pp)

Here's an excellent programming guide to Microsoft's newest version of QuickBASIC by the authors of *The BASIC Teacher*. The book approaches QuickBASIC's programming environment in three stages so beginning and experienced BASIC programmers can find the appropriate level of instruction.

**OMH-881274 Using
QuickBASIC. . . \$19.95**

DOS Made Easy

By Herbert Schildt

(Osborne/McGraw-Hill, 385 pp)

Previous computer experience is not necessary to understand this concise, well-organized introduction that's filled with short applications and exercises. The book walks you thru all the basics, beginning with an overview of a computer system's inner components and a step-by-step account of how to run DOS for the first time.

OMH-881194 DOS Made
Easy **\$18.95**

Mathematics, Magic and Mystery

By Martin Gardner

115 diversions, magical tricks arising from mathematical principles.

DOV-20335 paperbound 176pp \$3.50

Mathematical Puzzles of Sam Loyd

Selected and edited by Martin Gardner

One of the very few great innovators of puzzles, Sam Loyd (1841-1911)

invented thousands of the most valuable, ingenious, and popular puzzles ever originated.

DOV-20498 paperbound 167pp \$3.50

*More Mathematical Puzzles of
Sam Loyd*

Selected and edited by Martin Gardner

For more than 50 years, Sam Loyd's ingenious posers appeared in innumerable newspapers and magazines.

DOV-20709 paperbound 177pp \$3.50

Order Form

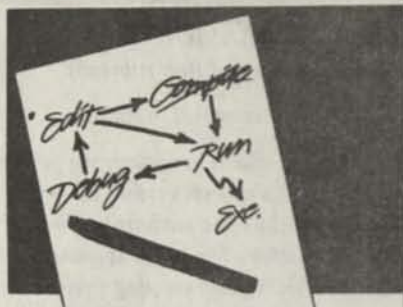
Send check or money order.
Foreign customers send
international money order
available at post offices and
banks.

Name _____
Address _____
City/State _____ Zip _____
Phone () _____

[illegible]

PC Magazine
Technical
Excellence
Award
Winner

Microsoft QuickBASIC 4.0



Microsoft

A powerful, full-featured programming language, QuickBASIC 4.0 takes BASIC into an entirely new dimension by adding source-level debugging, huge arrays, unlimited string space, support for Hercules graphics, and a wealth of other important features. The most impressive new feature of all is the threaded p-code interpreter. QuickBASIC 4.0 uses an incremental compiler that converts each line of source code as it is entered. What makes this system impressive is its ability to stop a program's execution, examine variables and make changes to the source code, and then resume execution. Further, QuickBASIC programs can now call routines written in any of the other Microsoft languages, and vice versa.

"Even the most cynical 'structure' fanatics and BASIC bashers must now agree that BASIC is a serious development language . . . BASIC has indeed come of age."

—Ethan Winter
PC Magazine

MS-11407 QuickBASIC 4.0 . . . \$95.00

ADVANCED COLOR GRAPHICS AND ANIMATION FOR THE IBM PC

By Don Inman and Kurt Inman

(Hayden, 248pp)

ACHIEVE SOPHISTICATED screen effects with this example-oriented text that shows you how to use the extended graphics capabilities of IBM BASICA for programming displays.

The ShareWare Book Using PC-Write, PC-File, PC-Talk By Emil Flock, et al

(Osborne/McGraw-Hill, 688pp)

Covers the most popular "free" programs: PC-Write, a word processor; PC-File, a database manager; and PC-Talk, a telecommunications program. These programs are available thru user groups or bulletin-board services in return for a nominal registration fee. The book has all the details on how you can obtain these program disks.

OMH-881251 The Shareware
Book. \$14.95

QuickBASIC: The Complete Reference

By Steven Nameroff

(Osborne/McGraw-Hill, 700pp)

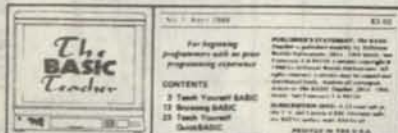
Publication date pushed back to November '88. Please do not order this book until further notice.

Whether you're creating a business chart or an artistic animation, you'll learn the techniques that give your work professional results.

HAY-121 Advanced Color
Graphics \$18.95

Back Issues Available

TBT-1 Issue No. 1 \$3.00
TBT-2 Issue No. 2 \$3.00
TBT-3 Issue No. 3 \$3.00



Different Worlds Publications

2814 - 19th Street
San Francisco, CA 94110

Address Correction Requested

BULK RATE
U.S. Postage
PAID
San Francisco, CA
Permit No. 11798



No. 5, December 1988

\$3.00

*For beginning
programmers with no prior
programming experience*

CONTENTS

- 3 Teach Yourself BASIC
- 9 Browsing BASIC
- 17 Teach Yourself
QuickBASIC

PUBLISHER'S STATEMENT: *The BASIC Teacher* is published monthly by Different Worlds Publications, 2814 - 19th Street, San Francisco, CA 94110. Contents copyright 1988 by Different Worlds Publications. All rights reserved. Contents may be copied and distributed freely. Address all correspondences to *The BASIC Teacher*, 2814 - 19th Street, San Francisco, CA 94110.

SUBSCRIPTION INFO: A 12-issue sub in the U.S. and Canada is \$36. Overseas subs are \$42 by surface mail, \$54 by air.

PRINTED IN THE U.S.A.

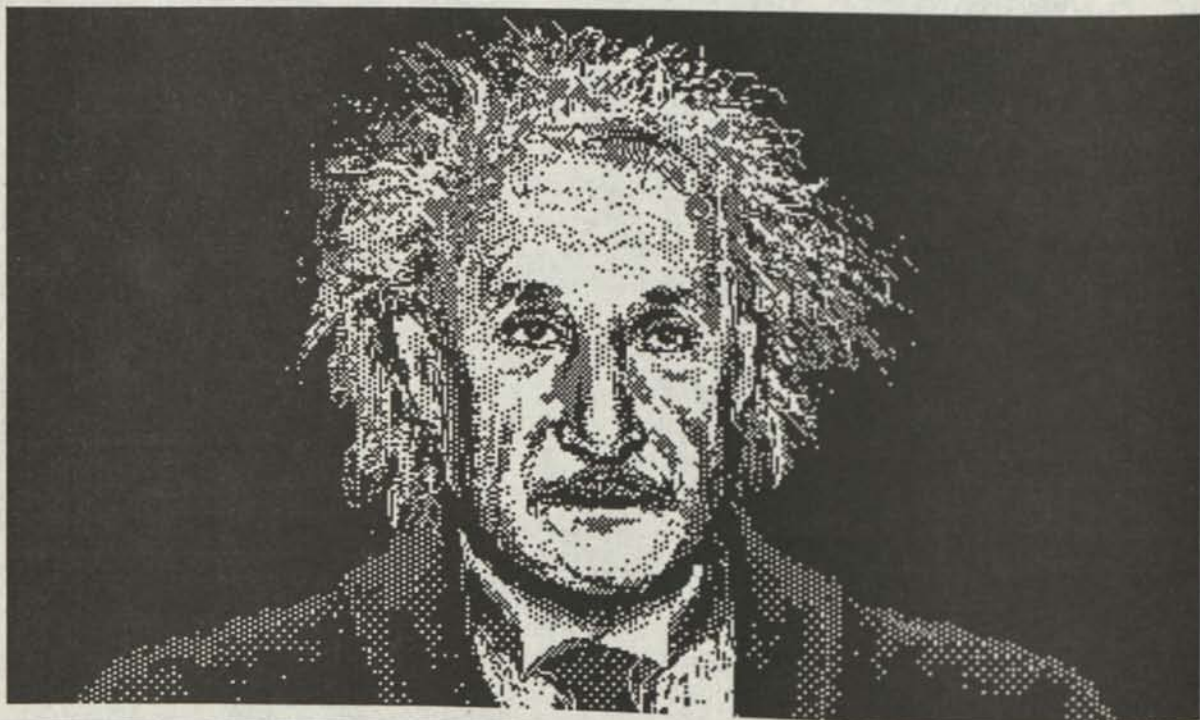


The Math Crisis

"UNLESS WE increase our own supply of mathematicians, we are going to find by the late 1990s that we will have to import foreigners to train our next generation of scientists." So declares Dr. Edward A. Connors of the University of Massachusetts at Amherst, who chairs a joint committee on educational policy for the American Mathematical Society and the Mathematical Association of America.

In a recent conversation, Dr. Connors made the following points:

- Fewer and fewer U.S. citizens are interested in advanced mathematics.
- Last year marked a 20-year low in the number of graduate math degrees earned by Americans.
- Between July 1986 and June 1987, fewer than 400 U.S. citizens were awarded doctorates in math. During that period, 51% of the math doctorates awarded in the U.S. were earned by foreign citizens studying here.
- The federal government should offer incentives to students to pursue math as a career—i.e., forgive its "educational loans" to those who major in math.
- The decline in the number and quality of math students in this country can be traced to poorly trained elementary-school teachers. Many of them simply do not know the subject or how to teach it.
- Many of the teachers do not realize that if they fail to interest a student in math by the sixth grade, they've lost the student forever.
- Parents should consistently encourage their children, particularly girls, to consider math as a career possibility. The traditional belief that females have a "math gene" missing in their heritage structure is so much nonsense.



Copyright 1988 T/Maker Company. From ClickArt Personal Graphics.



INTRODUCTION

BASIC HAS a small **vocabulary** and a simple **syntax** (grammar). We have already discussed some of the special words that Microsoft BASIC understands. They are called **keywords** or **reserved words**.

Here are the keywords introduced and described previously:

BEEP	GOTO	NEW	RANDOMIZE	TIME\$
CLS	INT	OFF	RND	TIMER
COLOR	KEY	ON	RUN	WIDTH
DATE\$	LIST	PRINT	SOUND	

Last time we described **numeric variable** and **values of numeric variables**. A numeric variable can be a single letter or a combination of letters and digits up to 40 characters long. The first character must be a letter. We will always show keywords (BEEP, CLS, etc.) in all upper-case letters. We will usually show variables in lower-case letters or a MiXtuRe of upper and lower-case letters.

FOR...NEXT Loops

YOU CAN use a FOR ... NEXT loop to assign a **sequence of values** to a numeric variable. The program shown below uses a FOR ... NEXT loop to print the numbers 1, 2, and so on, up to 7. The FOR ... NEXT loop is in lines 20, 30, and 40.

```
10 CLS
20 FOR number = 1 TO 7
30   PRINT number
40 NEXT number
```

Enter and run this program. You should see the numbers 1 to 7 on the screen.



A FOR...NEXT Loop

- BEGINS WITH a FOR statement;
- ends with a NEXT statement;
- usually has one or more statements between FOR and NEXT.

A numeric variable must follow the word FOR:

```
20 FOR number = 1 TO 7
    ← numeric variable
```

The same numeric variable follows the word NEXT:

```
40 NEXT number
    ← numeric variable
```

This numeric variable can be used in statements between FOR and NEXT:

```
30   PRINT number
    ← numeric variable
```


A Sequence of Values

A FOR statement defines a sequence of values for the numeric variable that follows the word FOR.

The statement: `FOR number = 1 TO 7`
defines the following sequence of values for the variable **number**:

1, 2, 3, 4, 5, 6, 7

When you ran the program on the preceding page, you saw these numbers printed on the screen by the PRINT statement in line 30.

In the following program, the FOR statement defines a sequence of numbers from 0 to 5 as the value of **number**.

```
10 CLS
20 FOR number = 0 TO 5
30 PRINT number;
40 NEXT number
```

Note the semicolon (;) at the end of line 30. This causes the numbers to be printed close together on the same line:

```
0 1 2 3 4 5
Ok
—
```

You can use any numeric variable after the word FOR. Also use the same variable after the word NEXT.

In the following program, the FOR statement defines a sequence of numbers from 10 to 13 as the values of the variable **k**.

```
10 CLS
20 FOR k = 10 TO 13
30 PRINT k;
40 NEXT k
```

Here is a run:

```
10 11 12 13
Ok
—
```

A Colorful FOR...NEXT Loop

THE "NORMAL" screen colors are whitish letters on a black screen. You can use the COLOR statement to tell the computer to print in any of 16 colors, including black (COLOR 0) and the normal white (COLOR 7). The following program prints one line in each of the 15 colors from 1 to 15 and tells you the color number in which it was printed:

```
10 CLS
20 FOR kolor = 1 TO 15
30 COLOR kolor
40 PRINT "This is color number"; kolor
50 NEXT kolor
60 COLOR 7
```

Unfortunately we can't show you the beautiful colors here. When you run the program, you will see the following in 15 different colors:

```
This is color number 1
This is color number 2
This is color number 3
This is color number 4
This is color number 5
This is color number 6
This is color number 7
This is color number 8
This is color number 9
This is color number 10
This is color number 11
This is color number 12
This is color number 13
This is color number 14
This is color number 15
Ok
—
```

Color me wonderful!



Line 60 returns the screen to its normal foreground color (COLOR 7). This is done after the FOR...NEXT loop has been completed.

If you would like to see blinking colors, change line 20 to the following:

```
20 FOR kolor = 17 TO 31
```

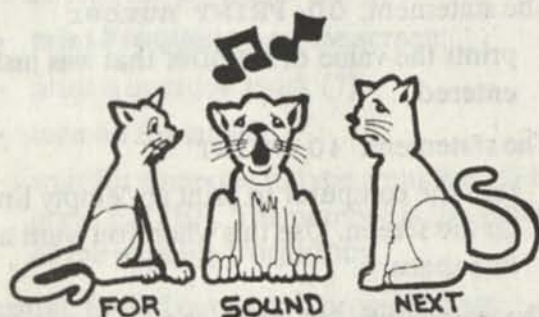
COLOR is a BASIC keyword and appears in upper-case letters; **kolor** is a numeric variable and appears in lower-case letters. We couldn't use **color** as a variable because COLOR is a keyword.

SOUND Effects

HAVE YOU ever wondered how they make all those strange sounds in arcade games? Try the following program:

```
10 CLS
20 FOR frequency = 100 TO 300
30   SOUND frequency, 0.125
40 NEXT frequency
```

Run this program. You will hear a sound that quickly rises from 100 Hertz (cycles per second) to 300 Hertz.



The FOR ... NEXT loop causes a sequence of very short sounds. Each sound is of duration 0.125 ticks. A tick, as you may recall from last time, is about 1/18 of a second. You hear 201 very short sounds with frequencies of 100 Hertz, 101 Hertz, 102 Hertz, and so on, up to 300 Hertz. This all happens in a little over a second.

With the above program, you hear one whoop, quickly rising in pitch. Now make a small change and get a program that goes whoop, whoop, whoop, ...

```
10 CLS
20 FOR frequency = 100 TO 300
30   SOUND frequency, 0.125
40 NEXT frequency
50 GOTO 20
```

When you have heard enough whoops, hold down CTRL and press BREAK to stop all the whooping.

Counting Backwards

ALL THE FOR ... NEXT loops we have used so far define **increasing sequences** of numbers:

- 1, 2, 3, 4, 5, 6, 7
- 0, 1, 2, 3, 4, 5
- 10, 11, 12, 13
- 100, 101, 102, ..., 300

You can tell the computer to count backwards by using a STEP -1 clause in the FOR statement, as shown in the following program:

```
10 CLS
20 FOR frequency = 300 TO 100 STEP -1
30   SOUND frequency, 0.125
40 NEXT frequency
50 GOTO 20
```

Run this program to hear a familiar arcade sound. This time you hear a falling pitch. The sound goes quickly from 300 Hertz to 100 Hertz in steps of -1.

- 300, 299, 298, ..., 100

Now put both programs together into a single program that makes a sound sort of like a siren. The sound goes up, down, up, down, etc:

```
10 CLS
20 FOR frequency = 523 TO 1046
30   SOUND frequency, 0.125
40 NEXT frequency
50 FOR frequency = 1046 TO 523 STEP -1
60   SOUND frequency, 0.125
70 NEXT frequency
80 GOTO 20
```



INPUT Makes It Easier

THE INPUT statement is useful for putting numbers into number boxes. It tells the computer to print a question mark (?) and wait for you to enter a number as a value of a numeric variable. Here is a short program with an INPUT statement:

```
10 CLS
20 INPUT number
30 PRINT number
40 PRINT
50 GOTO 20
```

Run the program. You see a question mark and the blinking cursor:

```
?_
```

The computer wants something. It wants what the INPUT statement tells it to want. It wants a number, a value for the variable **number**.

Type the number 7 and press ENTER:

```
? 7
7
? _
```

It prints the number you enter, then asks for another number, a new value of the variable **number**. Go ahead, enter more numbers. Press ENTER after each number. Each time, the computer prints back your number, then asks for another number:

```
? 7
7
? 123
123
? -20
-20
? qwerty
?Redo from start
? _
```

You see, qwerty is not a number.

Oh.



20 INPUT number

TELLS THE computer to:

- print a question mark;
- turn on the cursor;
- wait for someone to enter a value of the variable **number**.

The computer is very patient. It will wait and wait and wait . . . until you type a number and press ENTER. Then it continues with the next line of the program.

The statement: 30 PRINT number

prints the value of **number** that was just entered.

The statement: 40 PRINT

tells the computer to print an "empty line" on the screen. Use this when you want a line space.

The statement: 50 GOTO 20

sends the computer back to the INPUT statement in line 20, where it will again wait patiently for someone to enter a number.

Follow the Arrows

```
10 CLS
↓
20 INPUT number ←
↓
30 PRINT number
↓
40 PRINT
↓
50 GOTO 20
```

Around and around and around. Hold down CTRL and BREAK to break out of this loop.

INPUT " " "

THE INPUT statement tells the computer to print a question mark, turn on the cursor, and wait for someone to enter something. Wouldn't it be nice if, instead of a cryptic question mark, the computer would tell you what it wants? Easy. The following program has an "enhanced" INPUT statement:

```
10 CLS
20 INPUT "Frequency"; frequency
30 SOUND frequency, 18
40 PRINT
50 GOTO 20
```

Line 20 tells the computer to:

- print **Frequency** on the screen;
- print a question mark (?);
- turn on the cursor;
- wait for someone to type a number and press ENTER. The number is the value of the variable **frequency**.

As usual, to find out what a program does, enter it into the computer and run it. It begins like this:

```
Frequency? _
```

Type a number and press ENTER. You will hear a sound of that frequency for about one second ... if the frequency is within your hearing range. Try some of these:

```
Frequency? 262
```

```
Frequency? 1000
```

```
Frequency? 37
```

```
Frequency? 32767
```

```
Frequency? _
```



A slightly different program lets you enter both the frequency and duration of the sound:

```
10 CLS
20 INPUT "Frequency"; frequency
30 INPUT "Duration "; duration
40 SOUND frequency, duration
50 PRINT
60 GOTO 20
```

You can use this program to hear low sounds, high sounds, short sounds, long sounds, and in-between sounds:

```
Frequency? 1000
Duration ? 36
```

```
Frequency? 440
Duration ? 1
```

```
Frequency? 440
Duration ? 0.5
```

```
Frequency? 440
Duration ? 0.125
```

```
Frequency? 1000
Duration ? 0.1
```



Sound Effects

USE THE following program to find sound effects you like:

```
10 CLS
20 INPUT "First frequency"; first
30 INPUT "Last frequency "; last
40 INPUT "In steps of "; incr
50 INPUT "Duration "; duration
60 FOR frequency = first TO last STEP incr
70 SOUND frequency, duration
80 NEXT frequency
90 PRINT
100 GOTO 20
```

For example, try these:

```
First frequency? 100
Last frequency ? 1000
In steps of ? 100
Duration ? 0.25
```

```
First frequency? 1000
Last frequency ? 100
In steps of ? -100
Duration ? 0.25
```



Function Keys

WHEN YOU first enter BASIC, you are in **SCREEN 0** in **80 column text mode**. The screen is in a text mode with **25 rows** (lines) and 80 character positions in each row... excellent for writing programs. The bottom row of the screen is the **key line**. The left part of the key line looks like this:

1LIST 2RUN 3LOAD" 4SAVE

The key line tells you what happens when you press a **function key**. On our keyboard, these keys are in a row across the top of the keyboard and are labeled F1, F2, F3, and so on. On other computers, they are at the left side of the keyboard. You can save time by using the function keys.

If you want to list a program on the screen, you can press the F1 function key. The computer prints LIST on the screen. Press ENTER to list the program.

- To list a program:

Press F1 and then press ENTER.

You can press F2 to run a program. You don't even have to press ENTER. The computer will run the program as soon as you press F2.



CAUTION! Before you press a function key, make sure the cursor is on a line by itself. Otherwise you might see a syntax-error message. Use the arrow keys to put the cursor on an otherwise empty line before you press a function key.

We'll tell you about other function keys later, when we need to use them. In the meantime, you can read about them in the *BASIC Reference Manual* that came with your computer.

Shortcuts

ANYTHING THAT reduces the amount of typing also reduces the likelihood of making a typing mistake. For example, you can use the function keys to more quickly LIST or RUN a program.

Several BASIC keywords can be typed by holding down the ALT key and pressing another key. Here is a list of shortcuts for some of the keywords we have used:

Keyword	Shortcut
COLOR	ALT C
FOR	ALT F
GOTO	ALT G
INPUT	ALT I
NEXT	ALT N
PRINT	ALT P
RUN	ALT R
WIDTH	ALT W

To type PRINT, hold down the ALT key and press the P key.



This is a short list of shortcuts. Others are possible. Go ahead and try ALT plus any other key and see what happens.

Practice using the shortcuts to enter and run this program:

```
10 COLOR INT(16*RND)
    ↖ ALT C
20 PRINT "Mariko ";
    ↖ ALTP
30 GOTO 10
    ↖ ALT G
```

To list the program:

- Press F1, then press ENTER.

To run the program:

- Press F2.



By Don Inman

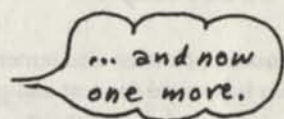
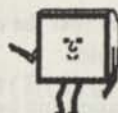
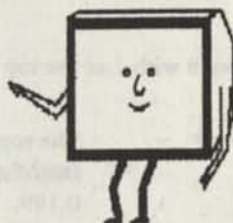
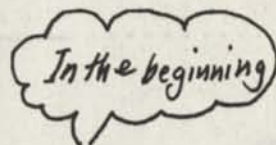
THE FOLLOWING topics were covered in previous installments of Browsing BASIC:

- No. 1: The **KEY** command and **ALT** key shortcuts.
- No. 2: The **VIEW PRINT** statement and pull-down windows.
- No. 3: The **COPY CON** command to write a **DRIVER.SYS** file and disk labels.
- No. 4: Test and Graphic Positions Relationships.

We started in Browsing BASIC No. 1 with a small list of keywords that we assumed you knew how to use. The list grew a little in issue 2. Our BASIC keyword list was static for one issue as we took a BASIC siesta to browse a bit of DOS. We then returned to BASIC as Browsing BASIC No. 4 added a few more keywords.

This month we will add one more:

CIRCLE



TEXT AND GRAPHIC POSITION RELATIONSHIPS

IN LAST month's Browsing BASIC, we discovered the following relationships between the numbering schemes for text and graphics screen positions.

Text positions are set by the **LOCATE** statement with the row specified first, followed by the column:

LOCATE row, column *row first, then column*

Unfortunately when BASIC locates graphic pixels, it reverses the order of the row and column values used in positioning text characters. This difference in order is seen when comparing a **LOCATE** (text character) statement with a **PSET** (graphics pixel) statement:

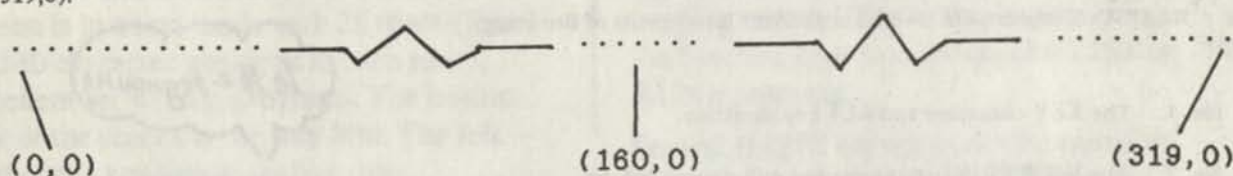
PSET (column, row) *column first, then row*

Another difference between the **LOCATE** statement and graphics location statements such as **PSET** is the fact that parentheses are used to enclose the column and row values in a graphics statement.

We also discovered that each text character occupies the same space as an 8 x 8 array of pixels, and the number of a pixel position in graphics does not correspond to the same number used for a text position.

The text size used in graphics mode **SCREEN 1** uses a maximum of 40 characters per line. The top text line of the screen is specified as row 1. The left-most character position in a line is specified as column 1. Thus the top line consists of positions 1,1, thru 1,40.

Since eight graphics pixels occupy the same width as one text character, there are 8 x 40, or 320, columns of pixels in each graphics row. The top graphics row is specified as row 0 (zero). The left-most pixel position in a row is specified as column 0 (zero). The top row of graphics pixels consists of positions (0,0) thru (319,0).



Rows in text begin with 1 at the top and run thru 25 (positions 1,1 thru 25,1).

0, 0 → . The top row of pixels of SCREEN 1 is numbered 0 (zero) and the left-most pixels run from 0,0 thru 0,199.

0, 199 → . Notice that the origin for numbering text and graphics units is also different: (1,1) for text and (0,0) for graphics.

Of course, you must also remember that the order of specifying row and column values is reversed for text and graphics.

We found that there is a direct relationship between text positions and graphics positions which can be used to calculate corresponding positions. They are:

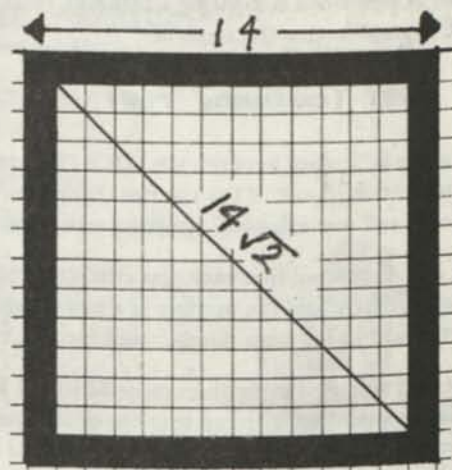
1. PixelColumn = 8 * (TextColumn - 1)
2. TextColumn = PixelColumn / 8 + 1
3. PixelRow = 8 * (TextRow - 1)
4. TextRow = PixelRow / 8 + 1

We used these relationships in a program, Squares Around Letters, to place a graphics rectangle around double-spaced text letters.

We concluded last month's episode with a few clues on drawing circles around text characters. Since a circle is drawn around its center, we had to change the reference point from:

Upper Left Corner of a Rectangle	$\begin{cases} pcol\% - 3 \\ prow\% - 3 \end{cases}$	Center of a Circle	$\begin{cases} pcol\% + 4 \\ prow\% + 4 \end{cases}$
-------------------------------------	--	-----------------------	--

The radius used to draw a circle around a text character should be approximately one-half the diagonal of the rectangle used in last month's discussion. The sides of that rectangle was 14 pixels long. Therefore, the diagonal is 14 times the square root of 2 as shown at the right. Therefore the radius of a corresponding circle is seven times the square root of 2, or approximately 10.



Program 5-1, CIRCLES AROUND LETTERS, replaces the LINE statement with box (B) option with the following **CIRCLE** statement:

CIRCLE ($\underbrace{\text{pcol\%} + 4, \text{prow\%} + 4}_{\text{center of circle}}, 10_{\text{radius}}$)

Here is the program.

```

1 REM ** Circles Around Letters **
2 ' Browsing BASIC #5 6/1/88
3 ' Microsoft GW-BASIC File: FNCYWORD.002

100 REM ** Set Graphics Screen and Get Information **
110 SCREEN 1: CLS: KEY OFF
120 INPUT "Text starting row : "; trow%
130 INPUT "Text starting column : "; txtcol%
140 PRINT: INPUT "Your word(s): "; Strng$

200 REM ** Clear Screen and Print Words **
210 CLS: prow% = 8 * (trow% - 1)
220 FOR num% = 1 TO LEN(Strng$)
230   tcol% = txtcol% + 3 * (num% - 1)
240   LOCATE prow%, tcol%: Letter$ = MID$(Strng$, num%, 1)
250   PRINT Letter$
260   IF Letter$ <> " " THEN GOSUB 1010
270 NEXT num%

300 REM ** Wait for Keypress, Restore Screens **
310 ky$ = INPUT$(1)
320 CLS: SCREEN 0: CLS: KEY ON: WIDTH 80
330 END

1000 REM ** SUBROUTINE: Circle the Letters **
1010 pcol% = 8 * (tcol% - 1)
1020 CIRCLE (pcol% + 4, kprow% + 4), 10
1030 RETURN

```

Run the Circles Around Letters program and enter your text. When we did, we saw these circled letters. The circles are not exactly round. Pixels are rectangular in shape, and it is impossible to draw small, round-looking circles with rectangular units.

R o u n d W o r d s

The **CIRCLE** statement can also produce ellipses (ovals) by adding an optional aspect ratio parameter (asp) to the statement:

CIRCLE (column, row), radius, color, start, end, asp

The default color used to draw the circle is color number 3 of the palette being used. The start and end options allow you to draw a portion of a circle (an arc), by specifying where to start and where to end the drawing. If you want to include aspect ratio, and not use the color, start, and end options, you must provide the necessary commas so the computer knows which value is the aspect ratio.

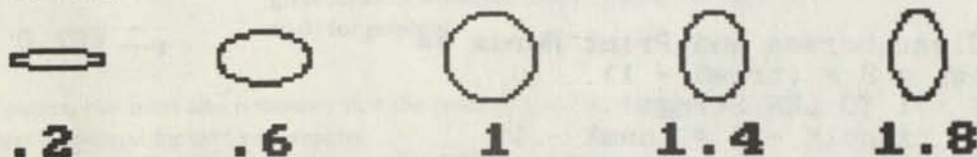
Example:

```
CIRCLE (pcol% + 4, prow% + 4), 10, , , , 1.1
```

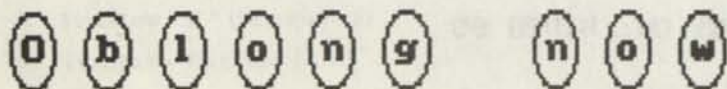
center
place holders for
color, start, and end
aspect ratio
(height to width)

The aspect ratio is a ratio of height and width. In general, values larger than 1 give ellipses that are higher than wide. Values less than 1 give ellipses that are wider than high.

Examples:

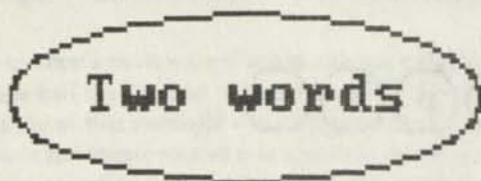


Add the comma placeholders and some value for aspect ratio in line 1020. Then run the revised Circles Around Letters program. You can also experiment with the value used for radius to ensure the letters are enclosed in the way you want them to be. Here is the result of using an aspect ratio of 1.4 with a radius of 10.




If you use a very small value for the aspect ratio, you may have to provide more space between letters.

You might want to write a program that places one ellipse around one or more complete words. Instead of centering the ellipse at a single letter, you will want to center the ellipse at the **middle** of the word or words. If the word has an odd number of letters, this would be at the center of the middle letter.




What happens when a word has an even number of letters? The ellipse should then be centered **between** two letters.


If the left-most column of the first letter in the word is used as a reference point, careful consideration leads to the following relationship:

One letter word 


center +4 pixels to right

Two letter word 

center +8 pixels to right

Three letter word 

center +12 pixels to right

Four letter word 

center +16 pixels to right

Therefore a slight modification is needed for our usual method of locating the pixel column from a text column. The relationship needed to center an ellipse about one or more letters is:

$$pcol\% = 8 * (txtcol\% - 1) + 4 * long\%$$

where: pcol% is the column for centering the ellipse
txtcol% is the left-most text column of the word
long% is the number of letters in the word

The following program, Ellipse Around Words, draws ellipses around words. You can enclose single words as shown at the right or a group of words as shown below the program.

One more input is requested at line 230, the aspect ratio. In most cases, the aspect ratio should be smaller than one. The radius is automatically adjusted to the length of the word, or words.

```

1 REM ** Ellipse Around Words **
2 ' Browsing BASIC #3 6/1/88
3 ' Microsoft GW-BASIC File: FNCYWORD.003

100 REM ** Set Graphics Screen **
110 SCREEN 1: CLS: KEY OFF

200 REM ** Get Information **
210 INPUT "Text starting row : "; trow%
220 INPUT "Text starting column : "; txtcol%
230 INPUT "Aspect ratio : "; asp!
240 PRINT
250 INPUT "Your Word(s): "; Strng$
260 long% = LEN(Strng$)

```

One

word

at

a

time



CONTINUED ...


```

300 REM ** Clear Screen and Print Words **
310 CLS: prow% = 8 * (trow% - 1)
320 FOR num% = 1 TO long%
330   tcol% = txtcol% + num% - 1
340   LOCATE trow%, tcol%: Letter$ = MID$(Strng$, num%, 1)
350   PRINT Letter$
360 NEXT num%
370 GOSUB 1010

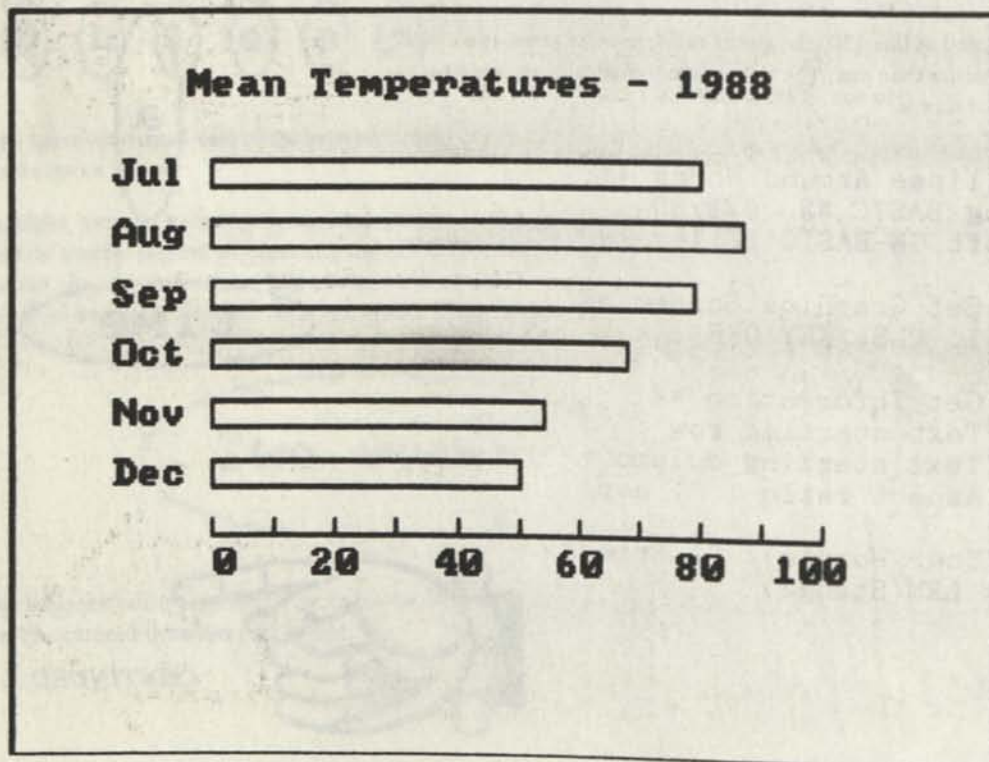
400 REM ** Wait for Keystroke Restore Screen and End Program **
410 KY$ = INPUT$(1)
420 CLS: SCREEN 0: CLS: KEY ON: WIDTH 80
430 END

1000 REM ** Subroutine: Draw Ellipse **
1010 pcol% = 8 * (txtcol% - 1) + 4 * long%
1020 CIRCLE (pcol%, prow% + 4), long% * 4 + 16,,, asp!
1030 RETURN

```

A lot at once

Now that we've got the basics out of the way, let's tackle a more practical application that mixes text and graphics. Here is the output of a program that places a title, labels, and a scale on a graph.



The title is placed at the top of the graph, labels are placed for each bar along the side, and a scale is located at the bottom of the graph. The text and graphics are printed and drawn from the top of the graph downward. The title is printed first. Each label and its associated bar is drawn in a FOR ... NEXT loop using the graphics-to-text relationships we have been discussing.

```

310 pcol% = 8 * 8: num% = 2
320 LOCATE 3, 8: PRINT Text$(1)           'Title
330 FOR trow% = 6 TO 16 STEP 2
340   LOCATE trow%, 5: PRINT Text$(num%)   'Label
350   prow% = 8 * (trow% - 1)
360   LINE (pcol%, prow%)-(pcol% + 2 * Temp%(num%), prow% + 7), , B
370   num% = num% + 1
380 NEXT trow%

```

The scale is placed below the bars using mixed text and graphics.

```

410 LOCATE 19, 9: PRINT "0   20   40   60   80  100"
420 LINE (64, 140)-(264, 140)
430 FOR tick% = 64 TO 264 STEP 20
440   LINE (tick%, 136)-(tick%, 139)
450 NEXT tick%

```

The following program produces the Mean Temperatures graph. Enter it then run it to verify the results shown.

```

1 REM ** Bar Graph **
2 ' Browsing BASIC 4/2/88
3 ' Microsoft GW-BASIC File: BARGRAPH.001

100 REM ** Define Screen **
110 SCREEN 1: CLS : KEY OFF

200 REM ** Define Text Strings **
210 Text$(1) = "Mean Temperatures - 1988"
220 FOR num% = 2 TO 7
230   READ Text$(num%), Temp%(num%)
240 NEXT num%
250 DATA Jul,80, Aug,87, Sep,79, Oct,68, Nov,54, Dec,50

300 REM ** Place Text, Draw Bars and Wait for Keypress **
310 pcol% = 8 * 8: num% = 2
320 LOCATE 3, 8: PRINT Text$(1)
330 FOR trow% = 6 TO 16 STEP 2
340   LOCATE trow%, 5: PRINT Text$(num%)
350   prow% = 8 * (trow% - 1)
360   LINE (pcol%, prow%)-(pcol% + 2 * Temp%(num%), prow% + 7), , B
370   num% = num% + 1
380 NEXT trow%

```



MORE


```

400 REM ** Draw Scale **
410 LOCATE 19, 9: PRINT "0  20  40  60  80  100"
420 LINE (64, 140)-(264, 140)
430 FOR tick% = 64 TO 264 STEP 20
440   LINE (tick%, 136)-(tick%, 139)
460 NEXT tick%

```

```

500 REM ** Restore the Screen & END **
510 CLS: SCREEN 0: CLS : KEY ON: WIDTH 80
520 END

```



This program draws a specific bar graph. The program would have to be rewritten for a different set of data. Can you write a more general program that would let you enter the data and automatically print the title, scale, and labels? The placement of the bars would depend upon how many entries you have. Their size would vary with the data entered. If you come up with such a program, send it to *The BASIC Teacher*.

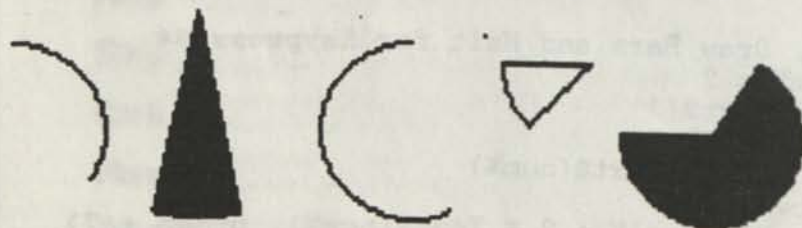
CIRCLES

WE USED the CIRCLE statement in the Ellipse Around Words program to specify the center, radius, and aspect ratio of the ellipse. If you look at that statement, you will see a bunch of commas between the radius and the aspect ratio.

CIRCLE (pcol%, prow% + 4), long% * 4 + 16 , , , , asp!

center ratio aspect ratio

The commas hold a place for optional circle specifications: color, start point, and end point. The color option allows you to choose the drawing color from a four-color palette in SCREEN mode 1. The start-point and end-point options allow you to draw an arc instead of a complete circle. The start point, stated in radians, tells where to start drawing the arc. The end point, also stated in radians, tells where to stop drawing the arc. Here are some examples:



More on circles in future issues of *The BASIC Teacher*.

QuickBASIC 4.5



QUICKBASIC CONTINUES to get better and better, especially for beginners. Microsoft recently announced QuickBASIC 4.5, which includes a hypertext-based help facility and other features to make it even easier to learn and use by first-time users. QB 4.5 includes *QB Express*, an on-line tutorial to help you get started, and the hypertext-based *QB Adviser* to help you keep learning.

We have just finished writing a beginner's book on the new QB 4.5. It's called *QuickBASIC Made Easy* and will be published by Osborne/McGraw-Hill in very early 1989. We know QB 4.5 is great for kids 'cause our 4th-grade to 8th-grade Computer Kids are learning it just fine. QuickBASIC is by far the best computer language we have used in 24 years of teaching kids and teachers how to use, program, and enjoy computers.

Computer Kid, USA, PO Box 1635, Sebastopol, CA 95473, USA

Characters

YOU CAN print many different characters on the screen. You can see some of these characters on the keyboard.

- upper-case letters: A B C D
- lower-case letters: a b c d
- digits: 1 2 3 4
- punctuation: . , ; :
- special characters: @ # \$ *

There are also computer characters you don't see on the keyboard. Some are shown below:

- card characters: ♥ ♦ ♣ ♠
- Greek letters: α β ϵ π Σ
- math symbols: $\sqrt{}$ \pm \leq \equiv \approx
- graphics characters: L ⊥ † ‡ ¶

ASCII

EVERY COMPUTER character has an **ASCII** code. An ASCII code is an integer in the range of 0 to 255. ASCII means American Standard Code for Information Interchange. Here are some examples:

- The ASCII code for **A** is 65
- The ASCII code for **B** is 66
- The ASCII code for **a** is 97
- The ASCII code for **b** is 98
- The ASCII code for * is 42
- The ASCII code for ♥ is 3

You have probably guessed that the ASCII code for C is 67. For the upper-case letters A to Z, the ASCII codes are 65 to 90. For the lower-case letters a to z, the ASCII codes are 97 to 122. Digits also have ASCII codes—the codes for the digits 0 to 9 are 48 to 57.

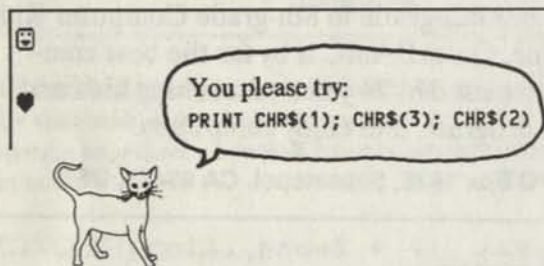
CHR\$

YOU CAN use the CHR\$ function to print any ASCII character on the screen. CHR\$ is a **string function**. Its value can be any single character. A string function always ends in a dollar sign.

To see CHR\$ at work, go to QB Control and enter the following program in the View Window:

```
CLS
PRINT CHR$(1)
PRINT
PRINT CHR$(3)
```

Run this program to see characters 1 and 3.



You can see that CHR\$(1) is a tiny face and CHR\$(3) is a heart. Go ahead and try some more. ASCII numbers less than 32 are used for special control purposes . . . so funny things may happen. For example, you can make the computer beep by trying to print CHR\$(7).

- CHR\$(32) is a space, so an invisible space is printed.
- CHR\$(36) is a dollar sign (\$).
- CHR\$(42) is an asterisk (*).
- CHR\$(57) is the digit 9.
- CHR\$(65) thru CHR\$(90) are the upper-case letters, A thru Z.
- CHR\$(97) thru CHR\$(122) are the lower-case letters, a thru z.
- CHR\$(128) thru CHR\$(255) are foreign alphabets, graphics characters, math symbols, and other stuff.

ASCII Soup

ENTER AND run the following program to see lots of character in 15 gorgeous colors.

```
CLS
RANDOMIZE TIMER
DO
  COLOR INT(15 * RND) + 1
  ascii = INT(223 * RND) + 32
  PRINT CHR$(ascii);
LOOP
```

The program quickly fills the screen with randomly-selected characters in randomly-selected colors. Since some characters from 0 to 31 can cause funny things to happen, this program selects only from characters 32 to 255, as follows:

```
ascii = INT(223 * RND) + 32
```

This statement computes a random integer from 32 to 255 and assigns it as the value of the **variable** called **ascii**.

$$\underbrace{\text{ascii}}_{\text{variable}} = \underbrace{\text{INT}(223 * \text{RND}) + 32}_{\text{integer from 32 to 255}}$$

REMEMBER: INT(223 * RND) is a random integer from 0 to 222. Add 32 to get a random integer from 32 to 255.

The statement: COLOR INT(15 * RND) + 1
picks a random color from 1 to 15.

The statement: PRINT CHR\$(ascii)
prints the random character.

The statement: RANDOMIZE TIMER
uses the value of TIMER (seconds since midnight) to start the RND function at a different place for each possible value of TIMER.

To stop the program:

Tandy 1000: CTRL HOLD
IBM PC: CTRL BREAK

INPUT ascii

USE THE following program to see individual ASCII characters. To see a character, you type its ASCII code (0 to 255) and press ENTER.

```
CLS
DO
  INPUT ascii
  PRINT CHR$(ascii)
  PRINT
LOOP
```

Run the program. It begins like this:

?_

You see a question mark and the blinking cursor. The computer wants something. It wants you to INPUT a value of the variable **ascii**. Do it. Type any number from 0 to 255 and press the ENTER key. Try some of these:

? 1

☐

To stop the program:

? 3

Tandy 1000: CTRL HOLD

♥

IBM PC: CTRL BREAK

? 42

*

? 65

A

? 97

a

? 219

■

? 227

π

? -

—

Your turn. Type a number in the range 0 to 255 and press ENTER. What happens if you enter a number outside this range? Try it and find out.

The statement: **INPUT ascii**

tells the computer to:

- print a question mark;
- turn on the cursor;
- wait for someone to enter a number.

When you type a number and press ENTER, the computer puts your number into the number box called **ascii**. Your number is now the **value** of the **numeric variable** called **ascii**.

After accepting your number as the value of **ascii**, the computer goes on to the next statement and prints the ASCII character you requested.

The statement: **PRINT CHR\$(ascii)**

tells the computer to print the character whose ASCII number is the value of the numeric variable called **ascii**.



INPUT " " "

THE INPUT statement tells the computer to print a question mark, turn on the cursor, and wait patiently for someone to enter something. Wouldn't it be nice if, instead of a cryptic question mark, the computer would tell you what it wants? Easy. The following program has an "enhanced" INPUT statement.

```
CLS
DO
  INPUT "ASCII number, please"; ascii
  PRINT CHR$(ascii)
  PRINT
LOOP
```

The INPUT statement tells the computer to:

- print **ASCII number, please** on the screen;
- print a question mark;
- wait for someone to type a number and press ENTER. The number is the value of the numeric variable **ascii**.

As usual, to find out what a program does, enter it into the computer and run it. It begins like this:

```
ASCII number, please? _
```

Try some numbers:

```
ASCII number, please? 1
```



```
ASCII number, please? 3
```



```
ASCII number, please? 90
```



```
ASCII number, please? _
```



ASC

QUICKBASIC HAS a function called ASC that is just the opposite of CHR\$. ASC is a **numeric function** that gives the ASCII number of a character. Run the following program to see ASC in action:

```
CLS
PRINT ASC("A")
PRINT
PRINT ASC("a")
PRINT
PRINT ASC("*")
```

Run this program to see the ASCII codes (numbers) for the characters A, a, and *.

65 ← code for A

97 ← code for a

42 ← code for *

The statement: PRINT ASC("A")

tells the computer to print the ASCII code (number) for the character enclosed in quotation marks in parentheses following the word ASC:

ASC("A") is 65

ASC("a") is 97

ASC("*") is 42

If you put two or more characters in quotes, the ASC function will give you the ASCII code of the first character:

ASC("abc") is 97

ASC("123") is 49

Yes, digits also have ASCII codes!

PRINT ASC(" ")

Put your character here



Fancy PRINT

INSTEAD OF printing the character all by itself on a line, let's have it "say something." Use the arrow keys to put the cursor on the C in CHR\$, as shown below:

```
CLS
DO
  INPUT "ASCII number, please"; ascii
  PRINT CHR$(ascii)
  PRINT
LOOP
```

← cursor

Now type stuff so that the program looks like the one below:

```
CLS
DO
  INPUT "ASCII number, please"; ascii
  PRINT "That's my number! "; CHR$(ascii)
  PRINT
LOOP
```

← cursor

As you typed, the cursor moved to the right. CHR\$(ascii) also moved to the right. The cursor is still on the C in CHR\$.

Don't press ENTER. Use the down-arrow key to move the cursor off the line. The program is ready to run.

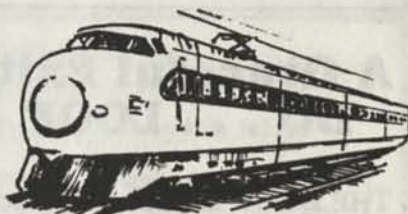
However if you did press ENTER, the program might now look like this:

```
CLS
DO
  INPUT "ASCII number, please"; ascii
  PRINT "That's my number! ";
  CHR$(ascii)
  PRINT
LOOP
```

If that happened, press the BACKSPACE key to rejoin CHR\$(ascii) with its PRINT statement.



Run It



WHEN WE ran the program, here is what we saw:

```
ASCII number, please? 1
That's my number! ☺
```

```
ASCII number, please? 3
That's my number! ♥
```

```
ASCII number, please? 65
That's my number! A
```

```
ASCII number, please? _
```

The statement:

```
PRINT "That's my number! "; CHR$(ascii)
```

tells the computer to:

- print **That's my number!**
- then print the value of CHR\$(ascii)

Note that **That's my number!** plus a space is enclosed in quotation marks:

```
    quotation marks
    ↙               ↘
  "That's my number! "
    ↑
  space
```

The information contained in quotation marks is called a **string**.

```
    "That's my number! "
    └──────────────────┘
                string
```

Note that the quotation marks are not printed. Only the string, which is enclosed in quotation marks, is printed.

Between the string (enclosed in quotation marks, of course) and CHR\$(ascii), you see a semicolon (;). If you forget to type it, QuickBASIC will put it in when you move the cursor off the line. Also try a comma (,) here.

A Graceful Exit from DO ... LOOP

THE DO ... LOOPs shown so far must be interrupted by using CTRL BREAK for the IBM PC or CTRL HOLD for the Tandy 1000. Below is yet another version of our ASCII character program. If you enter zero (0) as the ASCII number, it quits.

```
CLS
DO
  INPUT "ASCII number (0 to quit)"; ascii
  IF ascii = 0 THEN EXIT DO
  PRINT "That's my number! "; CHR$(ascii)
  PRINT
LOOP
```

Here is a sample run. We entered zero (0) to quit:

```
ASCII number (0 to quit)? 97
That's my number! a

ASCII number (0 to quit)? 122
That's my number! z

ASCII number (0 to quit)? 0
quit
Press any key to continue
```

You can also quit by pressing ENTER without typing anything. This enters zero as the value of **ascii** in the INPUT statement.



IF ... THEN

THE STATEMENT:

IF **ascii** = 0 THEN EXIT DO
tells the computer to compare the value of **ascii** with zero (0). If the value of **ascii** is zero, the computer leaves (exits) the DO ... LOOP.

If the value of **ascii** is not zero, the computer does not exit the DO ... LOOP. Instead it just goes on to the next line and prints the string in quotation marks and the value of **CHR\$(ascii)**.

```
IF ascii = 0 THEN EXIT DO
   If this is true then do this.
IF ascii = 0 THEN EXIT DO
   If this is false then don't do this.
```



Remember

ASCII CODES are numbers in the range 0 to 255. Most, but not all, ASCII codes represent characters you can print on the screen.

New keywords this time,
in order of appearance:

```
CHR$
RANDOMIZE
INPUT
ASC
IF...THEN
EXIT
```



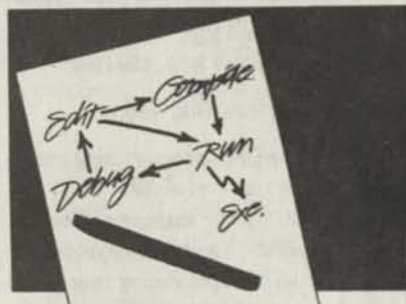
We also talked about a numeric variable called **ascii** and strings in INPUT and PRINT statements. Read about all of this in the book *BASIC Language Reference* that comes in the QuickBASIC package.

Different Worlds
2814 - 19th Street
San Francisco CA 94110

PC Magazine
Technical
Excellence
Award
Winner

Microsoft QuickBASIC 4.0

Microsoft



A powerful, full-featured programming language, QuickBASIC 4.0 takes BASIC into an entirely new dimension by adding source-level debugging, huge arrays, unlimited string space, support for Hercules graphics, and a wealth of other important features. The most impressive new feature of all is the threaded p-code interpreter. QuickBASIC 4.0 uses an incremental compiler that converts each line of source code as it is entered. What makes this system impressive is its ability to stop a program's execution, examine variables and make changes to the source code, and then resume execution. Further, QuickBASIC programs can now call routines written in any of the other Microsoft languages, and vice versa.

"Even the most cynical 'structure' fanatics and BASIC bashers must now agree that BASIC is a serious development language . . . BASIC has indeed come of age."

—Ethan Winter
PC Magazine

MS-11407 QuickBASIC 4.0 . . . \$95.00

ADVANCED COLOR GRAPHICS AND ANIMATION FOR THE IBM PC

By Don Inman and Kurt Inman

(Hayden, 248pp)

ACHIEVE SOPHISTICATED screen effects with this example-oriented text that shows you how to use the extended graphics capabilities of IBM BASICA for programming displays.

The ShareWare Book Using PC-Write, PC-File, PC-Talk

By Emil Flock, et al

(Osborne/McGraw-Hill, 688pp)

Covers the most popular "free" programs: PC-Write, a word processor; PC-File, a database manager; and PC-Talk, a telecommunications program. These programs are available thru user groups or bulletin-board services in return for a nominal registration fee. The book has all the details on how you can obtain these program disks.

OMH-881251 The Shareware Book . . . \$14.95

QuickBASIC: The Complete Reference

By Steven Nameroff

(Osborne/McGraw-Hill, 700pp)

Publication date pushed back to February '89. Please do not order this book until further notice.

Whether you're creating a business chart or an artistic animation, you'll learn the techniques that give your work professional results.

HAY-121 Advanced Color Graphics . . . \$18.95

Back Issues Available

TBT-1 Issue No. 1 . . . \$3.00
 TBT-2 Issue No. 2 . . . \$3.00
 TBT-3 Issue No. 3 . . . \$3.00
 TBT-4 Issue No. 4 . . . \$3.00



Different Worlds Publications

2814 - 19th Street
 San Francisco, CA 94110

Address Correction Requested

BULK RATE
 U.S. Postage
 PAID
 San Francisco, CA
 Permit No. 11798

MIACHEL K ERICKSON # 14
 BOX 250
 MONTE RIO CA 95462-0250



No. 6, February 1989

\$3.00

*For beginning
programmers with no prior
programming experience*

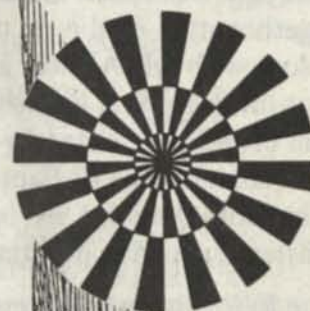
CONTENTS

- 3 Teach Yourself BASIC
- 13 Browsing BASIC
- 23 Teach Yourself
QuickBASIC

PUBLISHER'S STATEMENT: *The BASIC Teacher* is published monthly by Different Worlds Publications, 2814 - 19th Street, San Francisco, CA 94110. Contents copyright © 1989 by Different Worlds Publications. All rights reserved. Contents may be copied and distributed freely. Address all correspondences to *The BASIC Teacher*, 2814 - 19th Street, San Francisco, CA 94110.

SUBSCRIPTION INFO: A 12-issue sub in the U.S. and Canada is \$36. Overseas subs are \$42 by surface mail, \$54 by air.

PRINTED IN THE U.S.A.



Microsoft's New QuickBASIC 4.5

WITH QB4.5, Microsoft takes another major leap forward in language technology. As Microsoft QuickBASIC 4.0 brought together the latest threaded interpreter together with the most advanced BASIC programming language, Microsoft QuickBASIC 4.5 brings together state-of-the-art on-line help technology with QB Advisor as well as a flexible user interface that grows with the sophistication of the user, a step-by-step printed tutorial, and the QB Express on-line tutorial. Microsoft QuickBASIC is the easiest tool to master for solving programming problems.

The following are the key new QB4.5 features and their associated benefits:

- QuickBASIC now gives you the most important information in the easiest and most accessible way—at your fingertips, with QB Advisor.
 - » Complete hypertext electronic reference lets the user access info quickly and easily when he gets into trouble.
 - » Cut and paste examples to your programming window to execute the examples.
 - » New context-sensitive help on errors (syntax and runtime)—dialog boxes and menus is always there when the user needs it.
- New easy-to-use interface with Easy Menus lets the beginner type in and run programs with the minimum of menu choices. The more seasoned user may elect to use Full Menus to get the full power of QB.
- Superior integrated debugging lets you see exactly what the program is doing.
 - » Instant WATCH command available at all times for any variable or expression.



» Variable Help provides type and scoping information about procedures, variables, and data structures used in a program without searching through the source code.

- QB Express on-line tutorial program gets you started in the QuickBASIC environment quickly and painlessly.
- Step-by-step printed tutorial lets the beginner learn how to program quickly by working through a complete functional application (an electronic cardfile).
- Improved setup program makes setting up QuickBASIC a snap even on a floppy-based system.
- Intermediate values of complex expressions are now kept on the math co-processor chip, thus enhancing speed and removing any chance of propagating errors in rounding.
- ON UEVENT, UEVENT ON/OFF/STOP, and SLEEP statements added for industrial and instrumentation-control applications where interrupt processing is used.
- Multi-module error handling saves code size by allowing for a common error handler in a multi-module program.

As you can see, QB4.5 addresses the “ease-of-mastery” or how quickly the user becomes productive in solving programming problems as well as having all of the power for which QB4.0 was famous. QB4.5 reduces the learning hurdles to programming for PC power users who have run out of steam with their application products. QB will become even more entrenched as the “Swiss Army Knife” of programming tools.



Introduction



BASIC HAS a small **vocabulary** and a simple **syntax** (grammar). We have already discussed some of the special words that Microsoft BASIC understands. They are called **keywords** or **reserved words**.

Here are the keywords introduced and described previously:

BEEP	GOTO	NEW	RANDOMIZE	TIMER
CLS	INPUT	NEXT	RND	WIDTH
COLOR	INT	OFF	RUN	
DATE\$	KEY	ON	SOUND	
FOR	LIST	PRINT	TIME\$	

We always show keywords in all upper-case letters. We show variables in lower-case or a MiXtuRe of upper- and lower-case letters. We think this makes programs easier to read and understand. However, when you LIST a program, unfortunately, variables will probably appear in all upper-case letters.

Strings Are Handy

A **STRING** is a bunch of characters, one after another in, well, a string. All strung together as one entity called a string.

A string can be:

- a name: **Mariko**
- a telephone number: **707-555-1212**
- a message: **Trust your psychic tailwind**
- gibberish: **123Bz#m%&**



A string might be enclosed in quotation marks:

"Take a dragon to lunch"

 This string is enclosed in quotation marks.

The quotation marks enclose the string, but are not part of the string.

String Variables

A **STRING** can be the **value** of a **string variable**. Think of string variable as a labeled box that can hold, or store, a string.

FirstName\$ **George**

PhoneNumber\$ **707-555-1212**

The value of the string variable FirstName\$ is the string: **George**.

The value of the string variable PhoneNumber\$ is the string: **707-555-1212**.

A string variable is a name followed by a dollar sign (\$). It may consist of letters, numbers, a point (.), up to 40 characters if you like long names. The first character must be a letter.

Ok: city\$ word\$ abc.123\$

Oops: City word 123.abc

Numeric Variables, String Variables

CITY AND WORD are legal **numeric variables**, handy for storing numbers to use in calculations. However, a string variable must end in a dollar sign.

Numeric variables:

k x number price TIMER

String variables:

k\$ x\$ word\$ DATE\$ TIME\$

DATE\$, TIME\$, and TIMER are also BASIC keywords, so we show them in all upper-case letters. However, other keywords in our list cannot be used as variables, although they can be used as part of a variable name.

- You can't use COLOR as a variable,
- but ColorSet is Ok.

Don't worry about it. If you slip up, BASIC will remind you with an error message.

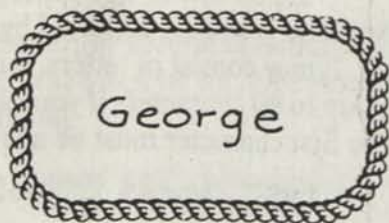
Here is one way to tell the computer to stuff a string in a string box. That is, here is one way to tell the computer to assign a string as the value of a string variable. Tell the computer to put the string **George** into a string box called **FirstName\$**.

Type:

```
FirstName$ = "George"
```

and press ENTER.

```
FirstName$ = "George"
Ok
-
```



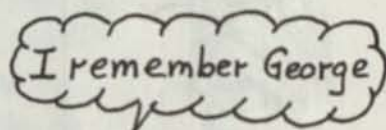
Print the Value of a String Variable

NOW FIND out what is in string box **FirstName\$**. Tell the computer to print the value of the string variable **FirstName\$**.

Type:

```
PRINT FirstName$
and press ENTER.
```

```
FirstName$ = "George"
Ok
PRINT FirstName$
George
Ok
```



REMEMBER how to assign a string value to a string variable:

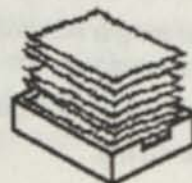
```
FirstName$ = "George"
```

string variable = string value in quotation marks

REMEMBER how to print the value of a string variable:

```
PRINT FirstName$
```

↑ ↖
PRINT keyword string variable



Practice

PRACTICE ASSIGNING a value to a string variable and then printing the value.

You type:

```
JediKnight$ = "Obi-wan Kenobi"
```

You type: PRINT JediKnight\$

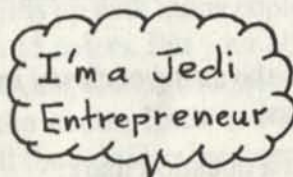
It prints: Obi-wan Kenobi

You type:

```
JediNovice$ = "Luke Skywalker"
```

You type: PRINT JediNovice\$

It prints: Luke Skywalker



If you didn't see any error messages, try these:

You type:

```
JediKnight$ = Obi-wan Kenobi
```

It prints: Type mismatch

(quotation marks missing)

You type:

```
JediNovice = "Luke Skywalker"
```

It prints: Type mismatch

(dollar sign missing)

You type:

```
Jedi Novice$ = "Luke Skywalker"
```

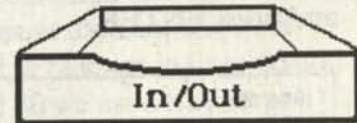
It prints: Syntax error

(space in variable name)

Use INPUT to Stuff String Boxes

YOU CAN use the INPUT statement to put strings into string boxes. INPUT causes the computer to print a question mark, then wait for someone to enter a value for a variable. Try this tiny program:

```
10 CLS
20 INPUT Strng$
30 PRINT Strng$
40 PRINT
50 GOTO 20
```



Strng\$ is a string variable. We didn't use String\$ because it is a BASIC keyword (STRING\$), which we haven't discussed yet, but will sometime.

Enter the program and run it. The program begins by clearing the screen (CLS). Then it prints a question mark, blinks the cursor, and waits for you to enter a value for the string variable Strng\$.

```
? _
```

The computer prints a question mark and waits. It is waiting for a string value to stuff into the string box Strng\$. The computer is very patient. If you don't cooperate by typing a string and pressing the ENTER key, it will wait, and wait, and wait.

So cooperate with your ever-patient computer. Type a string and press ENTER. George did it like this:

```
? George
George
```

```
? _
```

George typed his name and pressed ENTER. His name became the value of Strng\$ (line 20 of the program) and then was printed (line 30). The computer then printed a line space (thanks to line 40) and, as ordered by line 50, went back to line 20. It is now waiting for a new value of Strng\$.

? _

WHEN YOU see the question mark and the blinking cursor, type a string and press the ENTER key. The string you enter will replace **George** as the value of the string variable **Strng\$**.

Type: New string
and press ENTER.

```
? George
George

? New string
New string

? _
```



Note that the strings are not enclosed in quotation marks. That's Ok, unless you want to enter a string that contains a comma.

```
? Firedrake, George
? Redo from start
? _
```

If a string contains a comma, it must be enclosed in quotation marks, as shown below:

```
? "Firedrake, George"
Firedrake, George

? _
```

REMEMBER: If a string contains a comma, enclose it in quotation marks when you enter it in response to an INPUT statement.

Try a few more, then stop the computer. To stop the computer, hold down CTRL and press BREAK:

Hold down CTRL & press BREAK

```
? 
Break in 20
Ok
_
```

The computer tells you that a break occurred while it was doing line 20—and everything is Ok.

INPUT " "

INPUT PUTS a question mark on the screen, turns on the blinking cursor, then waits for someone to enter something. Wouldn't it be nice if, instead of just a cryptic question mark, the computer would tell you what it wanted? Easy! The following program has an "enhanced" INPUT statement in line 20:

```
10 CLS
20 INPUT "String, Please"; Strng$
30 PRINT Strng$
40 PRINT
50 GOTO 20
```

Line 20 has a **prompt string** enclosed in quotation marks. Line 20 tells the computer to:

- print the string enclosed in quotation marks;
- print a question mark;
- wait for someone to enter a value for the string variable **Strng\$**.

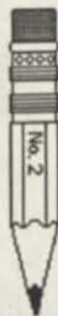
Run it. It begins like this:

```
String, please? _
```

Enter a string and press enter:

```
String, please? "Skywalker, Luke"
Skywalker, Luke

String, please? _
```



Go ahead. Type the string of your choice and press enter.

When you tire of entering strings, stop the computer with CTRL + BREAK.

A Shortcut for INPUT

You can use ALT + I as a shortcut for typing INPUT. Hold down the ALT key and press the I key. The computer will print INPUT and a space. Then you type the rest of the INPUT statement.

Your Message?

HERE IS a program that lets you enter a message, then prints it again and again and again—until you use CTRL + BREAK to stop the computer:

```
10 CLS
20 INPUT "Your message"; msg$
30 COLOR INT(15 * RND) + 1
40 PRINT msg$
50 GOTO 30
```

Run the program. It begins like this:

Your message? _

Type any message and press ENTER. Quickly the screen fills up with many copies of your message, in 15 colors. But your obedient computer continues printing the message at the bottom of the screen. Each new message "pushes" all the previous messages up a line—this is called **scrolling**. The message on the top line is "pushed off" the screen.

Stop the computer:

- Hold down CTRL and press BREAK.

The computer stops with many copies of your message on-screen, in many colors. At the bottom of the screen, you see the following, or something similar:

Break in 30
Ok
_

Oops—when you stop the computer, it can end in any of 15 colors, probably not the usual light gray (or is it dark white?). If you want to get back to the standard color for text,

Clear the screen (CTRL + L).

Type:

COLOR 7
and press ENTER.



Put Your Name Here, There, Everywhere

NOW TRY the following program. It puts your name (or whatever you enter) in random places all over the screen. It also selects random colors, including black (invisible) and blinking colors. The secret to this program is the LOCATE statement in line 70. LOCATE positions the cursor at the row and col (column) selected at random in lines 40 and 50. In lines 30 and 80 we used Naym\$ as a string variable because NAME is a BASIC keyword.

```
10 CLS
20 RANDOMIZE TIMER
30 INPUT "Your name"; Naym$
40 row = INT(24 * RND) + 1
50 col = INT(80 * RND) + 1
60 COLOR INT(32 * RND)
70 LOCATE row, col
80 PRINT Naym$;
90 GOTO 40
```



Go ahead—run the program. It begins like this:

Your name? _

Type your name and press ENTER. You will quickly see it here, there, anywhere on the screen in both blinking and non-blinking colors. Use CTRL + BREAK to stop the computer, then use a COLOR 7 statement to return to the normal screen color.

Lines 10, 20, and 30 are done only once. Then lines 40 thru 90 are repeated in a "loop" until you stop the computer. Like this:

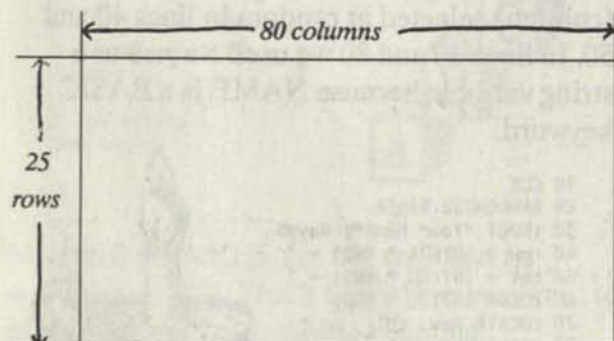
```
RUN
↓
10 CLS
20 RANDOMIZE TIMER
30 INPUT "Your name"; Naym$
↓
40 row = INT(24 * RND) + 1
50 col = INT(80 * RND) + 1
60 COLOR INT(32 * RND)
70 LOCATE row, col
80 PRINT Naym$;
90 GOTO 40
```



SCREEN 0

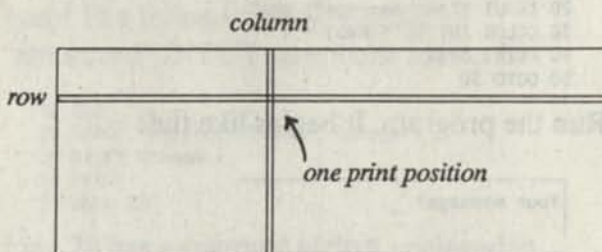
WHEN YOU first enter GW-BASIC, you are in **SCREEN 0** in **80-column text mode**. The screen is set up in a text mode having **25 rows** (horizontal lines) and **80 columns** (character positions across a row or line).

The bottom row of the screen is the key line. You can turn it off with **KEY OFF**, and on with **KEY ON**.



Rows & Columns

ROWS ARE numbered from 1 (top of screen) to 25 (bottom of screen). Print positions across the screen (**columns**) are numbered from 1 (left edge of screen) to 80 (right edge of screen).



Use the **LOCATE** statement to print stuff anywhere on the screen:

LOCATE row, col
 ↑ ↑
 1 to 25 1 to 80

A SCREEN MAP:

Here is a map of the left half of the screen. Use it to practice finding things on the screen.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	A									HERE															
2																									
3																									
4																									
5																									
6																									
7																									
8																									
9																									
10																									
11																									
12																									
13																									
14																									
15																									
16																									
17																									
18																									
19																									
20																									
21																									
22																									
23																									
24																									
25																									

ROW is in row 10, **COLUMN** is in column 30.

A is in row 1, column 1. A lonely star (*) is in row 19, column 20.

HERE is in row 1, columns 10, 11, 12, and 13.

THERE is in row 5, columns 16 to 20.

Row 25 is the **key line**. To turn it off, type **KEY OFF** and press **ENTER**.

Where is **Z**? _____

Where is the word **NOWHERE**? _____

Z is in row 24, column 40.

NOWHERE is in column 12, rows 12, 11, 10, 9, 8, 7, and 6. We suspect that, at first, you thought **NOWHERE** was, well, nowhere.

A Little Bit More About DOS

WE DIGRESS from BASIC in this issue to talk a little bit more about your Disk Operating System (DOS). DOS is the master program that coordinates the flow of information from your computer to your disks and from your disks to your computer. Every computer that uses disks must have a disk operating system. Despite its importance, DOS is quite often the most ignored part of a computer system. DOS is necessary, but it is usually used only for the most basic things—like formatting, and copying disks and disk files. Of course, DOS is also necessary to load the software that you use.

In Browsing BASIC No. 3, we discussed how to label disks and how to use DOS to write a short **CONFIG.SYS** file to the DOS disk. We began with a short list of DOS commands that we assumed you knew how to use:

DIR DISKCOPY COPY FORMAT

We added the following terms:

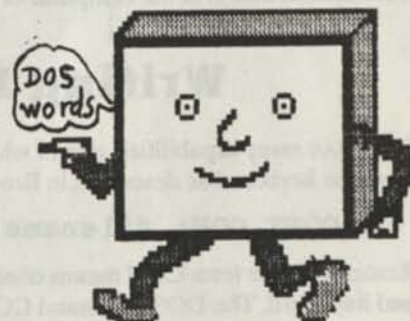
COPY CON DRIVER.SYS LABEL
DEVICE FORMAT /V

In this issue, we will use COPY CON to turn DOS into a note pad so that we can write short text files. As you read through the article, you should follow the procedures described. The notes that we save will contain information about using DOS.

DOS features discussed here are for MS-DOS, version 3.2 and 3.3. If a feature does not work on an earlier version, this fact will be stated in the discussion. If you have a version of DOS that is compatible with ours, use the text described. Otherwise, use notes that will apply to your version of DOS.

The MS-DOS version 3.20 used here is Tandy version 03.20.21. As you browse the following discussion, remember that there may be slight differences between this version and the version of MS-DOS that you are using.

Microsoft MS-DOS Version 3.20
(C) Copyright Microsoft Corp 1981, 1986
Tandy Version 03.20.21
Licensed to Tandy Corp.
All rights reserved.



There are two distinct types of DOS commands:

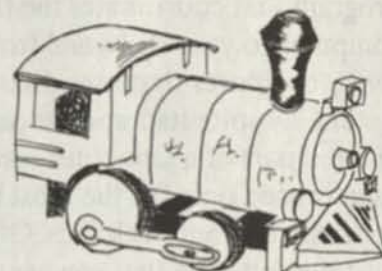
1. **Resident commands:** If you have loaded DOS into memory, resident commands can be performed whether or not your DOS system disk is in the computer. These commands are placed in memory when DOS is loaded.

Examples: COPY CLS DATE DIR TIME TYPE

2. **Transient commands:** These commands are really small, special-purpose programs that are on the DOS system disk but are not placed in memory when DOS is loaded. They are "brought in" from the system disk when you enter the command. Therefore you must have a DOS system disk in the computer when the commands are used.

Examples: BACKUP DISKCOPY FORMAT
GRAPHICS LABEL

The DOS commands used in this article are resident commands. Therefore everything discussed can be done after you have loaded DOS, whether your DOS system disk is in the computer or not.



Writing DOS Files

DOS HAS many capabilities, one of which is to write files to itself directly from the keyboard as described in Browsing BASIC No. 3.

A>COPY CON: filename opens a file for text input

Remember, the term CON means console, a combination of your monitor and keyboard. The DOS command COPY CON means copy (from the console) the file that follows (*filename*).

A DOS file is created in ASCII format by typing text at the DOS prompt and pressing the ENTER key at the end of each line of text. The file is closed by typing CTRL Z and pressing ENTER. The file is sent to the default disk drive (A in the above example) or to the disk drive specified in the filename. The file entries are echoed (displayed) on the screen as the file is created.

DOS: The Lazy Man's Word Processor

FORGETFUL PEOPLE often write notes to themselves in an effort to organize their work and play. I frequently write notes of things to be done in the future. A sequential data file is quite handy for note taking. These files are usually created from a word processor.

Being lazy by nature, my notes are quite often never made. The thought of loading a word processor (I do not have a hard disk), typing in the notes, and saving them to disk seems like too much trouble. So the note taking is usually put off until a later time. Usually, they are forgotten. Notes should be recorded quickly while still fresh in memory.



You could load **EDLIN**, a word processor file that is on the DOS disk. However, you found out in Browsing BASIC No. 3 that DOS could create an ASCII text file directly from the keyboard. Why not use the same method to record notes quickly? As long as the notes are short, a fancy word processor isn't really needed. From time to time, you can merge these files into a larger file using your favorite word processor. Or, you may use the **COPY** command as shown in Note Taking Session #2 to combine files.

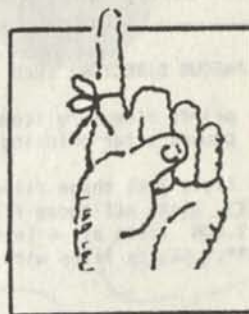
Note Taking Session #1

SINCE LEARNING more about DOS has been a long postponed project, I decided to use DOS to make notes about how to use DOS.

My first session with DOS note taking consisted of entering a few short lines of disk directory information. Since I usually save my files in disk drive B, I switched the default drive to B and entered the following DOS command.

```
A>B:
```

```
B>COPY CON: Note1.txt_
```



COPY is a resident command. **Note1.txt** is the name of the DOS ASCII text file to be created.

After entering the command, I typed in a title and several short statements about directory commands. Remember, I am creating a short note pad file without a word processor. Each line is entered in the file as a record when I press the **ENTER** key. If a typing mistake is made, I must correct it before pressing the **ENTER** key. Otherwise the record has already been entered, mistake and all. The backspace key can be used to back up and correct mistakes in a record as long as the **ENTER** key has not been pressed.

After the last entry, I pressed **CTRL Z** which marks the end of the file. The records were then sent to the file on the disk in drive B. Here is a screen print of the records as they were echoed to the screen when entered:

```
A>B:
```

```
B>COPY CON: Note1.txt
```

```
MISCELLANEOUS DIRECTORY INFO
```

```
DIR /W prints directory items across the screen.
DIR /P pauses after printing each page of items.
```

```
DIR A* lists just those files beginning with letter A.
DIR *.TXT lists all files that have the TXT extension.
DIR ????*.COM lists all 4 letter or less files with COM extension.
DIR ??S?*. lists files with S as the third character in its name.
```

```
^Z
```

```
1 File(s) copied
```

```
B>_
```


Checking The File

TO WRAP up my first DOS note taking session, I decided to check the file to see what it looked like. There are several ways to do this. Some of them will be discussed in the future. Since we have been using the COPY command, I used it to check the file with this command:

```
B>COPY Note1.txt LPT1:
```

This tells the computer to copy the file (**Note1.txt**) to my printer (LPT1:). Here is how the file was printed:

MISCELLANEOUS DIRECTORY INFO

DIR /W prints directory items across the screen.
DIR /P pauses after printing each page of items.

DIR A* lists just those files beginning with letter A.
DIR *.TXT lists all those files that have the TXT extension.
DIR ????.COM lists all 4 letter or less files with COM extension.
DIR ??S*. * lists files with S as the third character in its name.

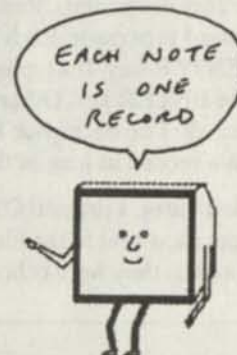
When the computer finished printing the file the following message was displayed on the screen:

```
B>COPY Note1.txt LPT1:
      1 File(s) copied
B>_
```

Each note in the **Note1.txt** file is saved as a single record in the file. Each blank line space in the file is also saved as a record. The file contains eleven records:

1. A blank line at the beginning.
2. The title **MISCELLANEOUS DIRECTORY INFO**.
3. A blank separating the title from the notes.
- 4-5. Notes on **DIR /W** and **DIR /P**.
6. A blank line.
- 7-10. **Wild card** directory notes.
11. A blank line.

If you haven't used DOS to create files before, stop now and experiment with this procedure. You may want to add more disk directory notes to the file. . . . Or, you may want to store information on an entirely different subject.



Note Taking Session #2

THIS SESSION includes two sections of related DOS features. Several ways are given to use the COPY command. The first section contains some uses that will be familiar to you. Some were used in Note Taking Session #1. The second section tells how to use the COPY command to combine two or more files.

Section 1 - COPY Files

Two important commands are listed first. They show how to "backup" files. Text and data files are often modified. You can make a secondary copy of a working file on the same disk. Then you can make modifications to the original file while a copy of the unmodified file is still available as a backup.

Example: A>COPY TCTQB1.DOC TCTQB1.BAK

I used the above command to copy a text file for a teacher's guide for our QuickBASIC book, *Using QuickBASIC*. The command copies the file TCTQB1.DOC to the same disk (in drive A) with a new name (different extension).

You can also make a backup copy of a file to a different disk. Then put the second disk away in a safe place. This would provide protection for inadvertent damage to the original or a loss of the file.

Example: A>COPY PHONENUM.DAT B:

The PHONENUM.DAT file is copied from the disk in drive A to the disk in drive B. The name is unchanged.

Additional uses of the COPY command are listed in Section 1, including copying data from a disk file to a device other than a disk drive.



Section 2 - Combine Files

The two items in this section tell how to combine multiple files.

A>COPY Note1.txt + Note2.txt

A>COPY *.dat ALL.txt

The first command combines two text files, **Note1.txt** and **Note2.txt**. We will use this command in Note Taking Session #3. The second command combines all files that have the .dat extension into one file named **All.dat**.

Disk drive B was used once again to copy the files. The name of the file copied to the console is **Note2.txt**. Session 2 opened with the following command:

A>B:

B>COPY CON: Note2.txt

A new text file is created from the keyboard. When finished, it will be copied to the disk in drive B.



After entering the command, the file was entered in the same way as **Note1.txt** in Note Taking Session #1. Line spaces were again inserted before and after the title, as well as between sections of the file. The ENTER key is pressed to create the line spaces. ENTER is also pressed at the end of each line of text. Here is how the screen looked just before the file was completed:

```
B>COPY CON: Note2.txt
```

COPY FILES

```
A> COPY TCTQB1.TXT TCTQB1.BAK makes a secondary copy of file.
A> COPY PHONES.TXT copies file from Drive A to Drive B.
A> COPY *.COM B: copies all COM files from Drive A to Drive B.
A> COPY *.* B: copies all files from Drive A to Drive B.

A> COPY NOTE1.TXT CON: copies NOTE1.TXT file to screen
A> COPY NOTE1.TXT LPT1: copies NOTE1.TXT file to printer.
A> COPY CON: NOTE2.TXT opens NOTE2.TXT file for text. Type in
text pressing ENTER after each line. End
with CTRL Z. Press ENTER to return to DOS.
```

COMBINE FILES

```
A> COPY NOTE1.TXT + NOTE2.TXT NOTEF.TXT combines NOTE1.TXT with
NOTE2.TXT to make NOTEF.TXT
A> COPY *.TXT ALL.TXT combines all .TXT files into one .TXT file.
```



Once again the file is closed by pressing CTRL Z, and the file is copied to the disk in drive B.

Stop now and make a copy of this file or a file of your choice. Copy it to the same disk that you used in Note Taking Session #1. In the next note taking session, we'll combine the two files.

Note Taking Session #3

WE WILL combine our two text files, **Note1.txt** and **Note 2.txt**, into one file named **NoteF.txt** by entering the following command:

```
B>COPY Note1.txt + Note2.txt NoteF.txt
```

combine these two files

copy result to this file

Before using the command, be sure both files, **Note1.txt** and **Note2.txt**, are on a disk in Drive B. Our directory, taken before combining the files, is shown below:

Directory of B:\

COMMAND	COM	23612	7-21-87	3:00p
NOTE1	TXT	391	6-05-88	3:02p
NOTE2	TXT	804	6-05-88	3:06p
3 File(s)		291840 bytes free		

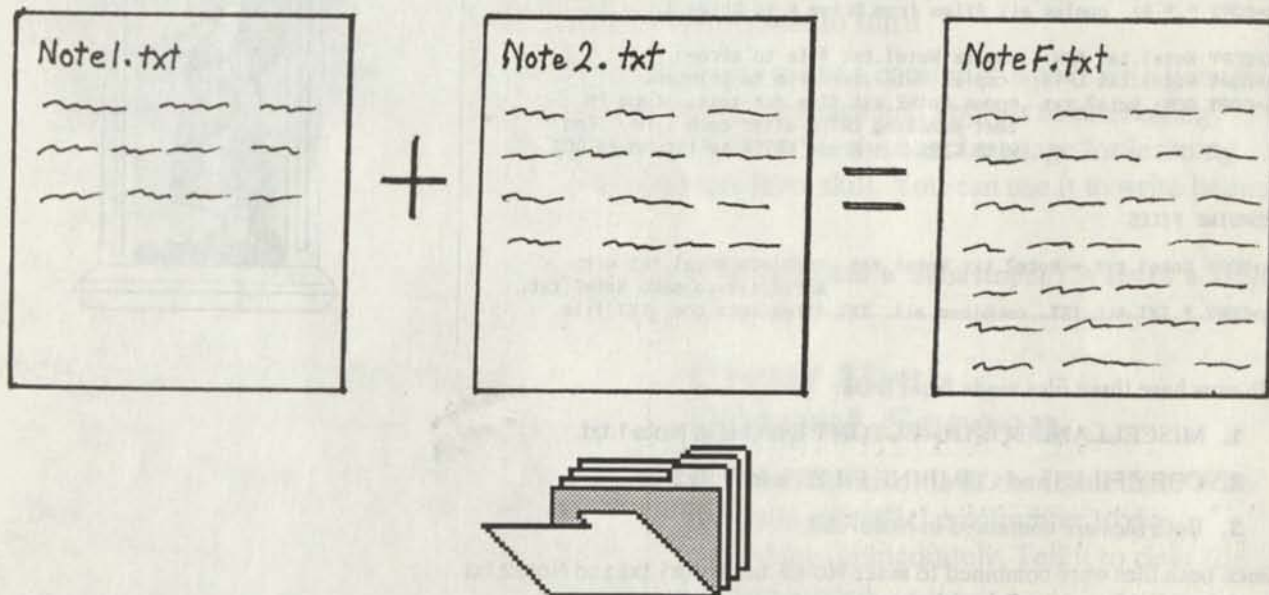


After entering the COPY command to combine the files, the red light on Drive B flickered and the disk whirled as the files were combined. Then this message appeared on the screen:

```
B>COPY Note1.txt + Note2.txt NoteF.txt
NOTE1.TXT
NOTE2.TXT
      1 File(s) copied

B>_
```

The files that were combined are listed directly below the command. Note that DOS tells you that one file was copied. This indicates that the two original files were combined. Of course, you will probably want to confirm that the **NoteF.txt** file has been created.



Checking The File

AFTER THE files were combined, a directory of the disk in Drive B was taken. The result is shown below:

Directory of B:\

COMMAND	COM	23612	7-21-87	3:00p
NOTE1	TXT	391	6-05-88	3:02p
NOTE2	TXT	804	6-05-88	3:06p
NOTEF	TXT	1196	6-05-88	3:17p
4 File(s)		289792 bytes free		

The **NoteF.txt** file does appear on the directory. As a final check, I decided I should send a copy to the printer with the command:

```
B>COPY NoteF.txt LPT1:
```


When the command was entered, the red light on Drive B flickered, the disk whirred, and the following text appeared line by line (or record by record from the disk's viewpoint) on my printer:

MISCELLANEOUS DIRECTORY INFO

DIR /W prints directory items across the screen.
DIR /P pauses after printing each page of items.

DIR A* lists just those files beginning with letter A.
DIR *.TXT lists all files that have the TXT extension.
DIR ????*.COM lists all 4-letter or less files with COM extension.
DIR ??S?*. lists files with S as the third character in its name.

COPY FILES

A>COPY TCTQB1.TXT TCTQB1.BAK makes a secondary copy of file.
A>COPY PHONENUM.TXT B: copies file from Drive A to Drive B.
A>COPY *.COM B: copies all COM files from Drive A to Drive B.
A>COPY *.* B: copies all files from Drive A to Drive B.

A>COPY Note1.txt CON: copies Note1.txt file to screen.
A>COPY Note1.txt LPT1: copies Note1.txt file to printer.
A>COPY CON: Note2.txt opens Note2.txt file for text. Type in text pressing ENTER after each line. End with CTRL Z. Press ENTER to return to DOS.

COMBINE FILES

A>COPY Note1.txt + Note2.txt NoteF.txt combines Note1.txt with Note2.txt to make NoteF.txt.
A>COPY *.TXT ALL.TXT combines all .TXT files into one .TXT file.

We now have three files made from DOS:

1. MISCELLANEOUS DIRECTORY INFO is in **Note1.txt**.
2. COPY FILES and COMBINE FILES is in **Note2.txt**.
3. Both files are contained in **NoteF.txt**.

Since both files were combined to make **NoteF.txt**, **Note1.txt** and **Note2.txt** can be erased from the disk with these two commands:

B>ERASE Note1.txt
B>ERASE Note2.txt



From time to time, Browsing BASIC will include tips, suggestions, and uses of MS-DOS. If you have particular DOS features that you would like discussed, send suggestions to **The BASIC Teacher**, 2814 - 19th Street, San Francisco, CA 94110. If there is enough demand, we might start a "Browsing DOS" section.





QuickBASIC 4.5

FOR 24 years we have been writing beginners' books about BASIC. More than 20 books so far and continuing. For 24 years we have been teaching kids how to use BASIC as a general-purpose problem-solving tool. We began with the very first BASIC, Dartmouth BASIC, and grew along with BASIC as it became better and better, according to the needs of people. QuickBASIC 4.5 is by far the best BASIC, in a class by itself—best for beginners, best for experts. If you want to learn BASIC, QB 4.5 is the place to start!

The most powerful way to use a computer is to learn a general-purpose programming language and apply it to interesting problems. QuickBASIC 4.5 is the best language for learning and teaching this high-level skill. You can use it to write beautiful software.

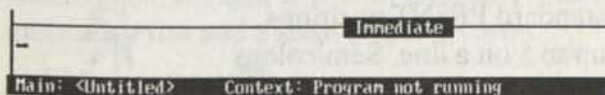
ComputerKid USA • PO Box 1635 • Sebastopol, CA 95473 • USA

Number Crunching

IF YOU misplace your \$10 solar-powered calculator, relax . . . you can use your computer as a calculator. You can use the **Immediate Window** to tell the computer to do arithmetic and print the results in the Output Screen. Use +, -, *, and / to specify arithmetic operations, as follows:

Operation	Example
Addition	3 + 4
Subtraction	3 - 4
Multiplication	3 * 4
Division	3 / 4

Go now to QB Control. Press the F6 function key to position the cursor in the Immediate Window, as shown here:



The cursor is in the Immediate Window.

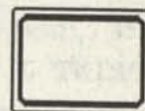
Clear the Output Screen

WHEN THE cursor is in the Immediate Window, you can tell the computer to do something—immediately. Tell it to clear the Output Screen.

Type:

CLS

and press ENTER.



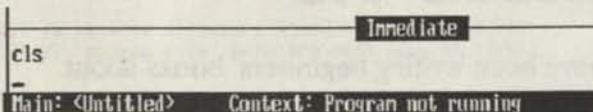
The Output Screen is now clear of any distracting information. It contains only the message "Press any key to continue," as shown below:

Press any key to continue

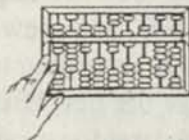
The Output Screen is clear except for the bottom line.

Return to QB Control

PRESS A key to return to QB Control. Your last command (CLS) and the cursor are still in the Immediate Window, as shown here.



Do Some Arithmetic



NOW TELL the computer to add two numbers, then subtract a number from another number, then multiply two numbers, then divide a number by another number. After each operation, you will see the result in the Output Screen. Press any key to return to the Immediate Window.

Type:

`PRINT 3 + 4` (addition)

and press ENTER.

Type:

`PRINT 3 - 4` (subtraction)

and press ENTER.

Type:

`PRINT 3 * 4` (multiplication)

and press ENTER.

Type:

`PRINT 3 / 4` (division)

and press ENTER.

After doing all four of the above, the top of the Output Screen should look like this:

```
7
-1
12
.75
```



Numeric Expressions

TO TELL the computer to do arithmetic and print the result, you can use a PRINT statement consisting of the keyword PRINT followed by a **numerical expression**.

`PRINT 3 + 4`

numerical expression

The computer evaluates the numerical expression (does the arithmetic), then prints the result, a single number.

Notice how the numbers are printed, as shown below:

```
7
-1
12
.75
```



A negative number is printed with a minus sign (-) followed by the digits of the number. A positive number (or zero) is printed as a space followed by the number.

You can put two or more numerical expressions in a PRINT statement. If you do, use commas or semicolons (;) to separate them.

You type:

`PRINT 3 + 4, 3 - 4`

It prints:

```
7          -1
```

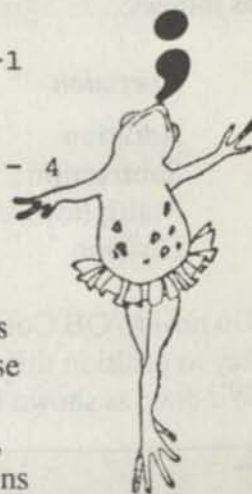
You type:

`PRINT 3 + 4; 3 - 4`

It prints:

```
7 -1
```

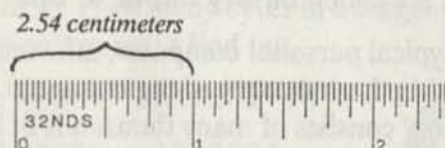
Commas between numbers in a PRINT statement cause answers to be printed in standard PRINT positions, up to 5 on a line. Semicolons cause numbers to be printed close together.



Mariko is 57 inches tall. How tall is she in centimeters? Hmmm... we seem to recall that one inch equals 2.54 centimeters.

You type: `PRINT 57 * 2.54`

It prints: 144.75



Easy! Just multiply the number of inches by 2.54 and print the result. But suppose you know the number of centimeters and want to compute the number of inches?

An ancient ruler named Zalabar measured 100 centimeters from the tip of his nose to the end of his outstretched finger.

How long is that in inches?

You type: `PRINT 100 / 2.54`

It prints: 39.37008

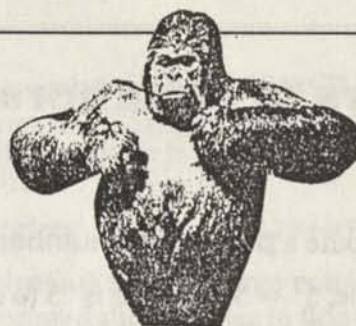
Call it 39.37. Does that sound familiar? Perhaps you recall that 100 centimeters is equal to one meter, and one meter is equal to 39.37 inches, a little more than one yard.

People usually give their height in feet and inches. If you ask Mariko how tall she is, she will probably tell you she is 4 feet, 9 inches tall. Given feet and inches, it's easy to write a `PRINT` instruction to compute height in centimeters:

You type: `PRINT 4 * 12 + 9`

It prints: 57

In evaluating the numerical expression $4 * 12 + 9$, the computer first does the multiplication ($4 * 12$), then does the addition ($+ 9$). In BASIC, the rules for doing arithmetic are very similar to the rules we use in "everyday" math. Remember, though, to use an asterisk (*) for multiplication and a slash (/) for division.



Before he reached his full stature, King Kong was once 37' 8" tall. How tall was he in centimeters?

You type: `PRINT (37 * 12 + 8) * 2.54`

It prints: 1148.08

We cleverly sneaked in the use of parentheses (). The rules for using parenthesis are very similar to the rules you learned in elementary school math classes. Your computer does the arithmetic inside parentheses first, then does the rest.

REMEMBER: 1 inch = 2.54 centimeters
1 meter = 39.37 inches

Recently, we took a trip in our ever-faithful car, Henrietta Honda. At the beginning of the trip, Henrietta had 19,832 miles on her odometer. At the end of the trip, her odometer read 20,219. We filled her tank at the beginning and again at the end. She burned 9.3 gallons of gas.

You type: `PRINT (20219 - 19832) / 9.3`

It prints: 41.6129

Well, let's call it about 41.6 miles to the gallon.

Most people on Earth use the metric system. Someday, we who live in the USA will also go metric. Instead of miles, we will use kilometers.

1 kilometer = 0.621371 mile
1 mile = 1.609344 kilometers

How many kilometers did we travel on that trip with Henrietta?

You type: `PRINT (20219 - 19832) * 1.609344`

It prints: 622.8161


Powers of Numbers

QUICKBASIC CAN do yet another arithmetic operation.

It can compute a **power of a number**.

For example, $5^2 = 5 \times 5 = 25$ is "5 to the second power" or "5 squared."

To compute a power of a number, use the ^ symbol.

To type ^ hold down SHIFT and press 

You type: PRINT 5^2

It prints: 25

Next, compute 5 to the third power, also called 5 cubed:

You type: PRINT 5^3

It prints: 125

Of course, you can also use multiplication (*) to compute a power of a number:

You type: PRINT 5 * 5

It prints: 25

You type: PRINT 5 * 5 * 5

It prints: 125

Here are some more examples of powers of numbers:

You type: PRINT 2^5

It prints: 32

You type: PRINT 10^6

It prints: 1000000

You type: PRINT 10^9

It prints: 1E+09

Oops!
What is 1E+09?
It's a **floating point number**.
Read on...



The Mysterious



COMPUTERS USE a very simple code, called **binary**, to represent information. Binary is very simple; it uses only two symbols, 0 and 1. The symbols 0 and 1 are called **binary digits**, or **bits**.

In a typical personal computer, information is stored in the **memory** of the computer. The memory consists of many thousands of bits organized as bunches of bits in **memory locations**.

One memory location can hold eight bits of information. A bunch of eight bits is called a **byte**. So... one memory location can hold eight bits, or one byte. The memory of a typical personal computer has many thousands of memory locations.

- One memory location can store 8 bits.
- A group of eight bits is called a byte.
- So, a memory location can store one byte.
- A computer memory has many thousands of locations. So the memory can store many thousands of bytes.

Perhaps you have heard about the mysterious K. People say a computer has 128K or 256K or 512K—or more—bytes of memory.

- 1K bytes equals 2^{10} bytes equals 1024 bytes.

Use the computer to change 1K bytes or 256K bytes or 512K bytes to ordinary numbers.

You type: PRINT 2^10

It prints: 1024

You type: PRINT 256 * 2^10

It prints: 262144

You type: PRINT 512 * 2^10

It prints: 524288

Computer memories are getting bigger. Your computer might have a **megabyte** or two or more. The term *mega* is borrowed from the metric system. It means one million. However in referring to the size of computer memories, it means 2^{20} (2 to the 20th power, or the 20th power of 2). How many bytes in a megabyte? Use the computer to find out.

You type: `PRINT 2^20`

It prints: 1048576



So, one megabyte is really 1048576 bytes. How many bytes in a 20-megabyte hard disk?

You type: `PRINT 20 * 2^20`

It prints: 2.097152E+07

20 megabytes is a rather large number. QuickBASIC printed this number in **floating point notation**, which is quite similar to **scientific notation** used in math and science books.

Floating point notation: 2.097152E+07

Scientific notation: 2.097152×10^7

Say it like this:

Two point zero nine seven one five two times ten to the seventh power.

That's a Lot of Bread!

PERHAPS YOU have heard the ancient story about the wise person who did a great service for a king. The king asked her what reward would be appropriate. Her request was simple. She asked only for grains of wheat, computed as follows: On the first square of a chessboard, one grain of wheat. On the second square, two grains of wheat. On the third square, four grains of wheat. And so on, doubling at each new square.

On square number n , there are to be 2^{n-1} grains. Let's find out how many grains on square 16:

You type: `PRINT 2^15`

It prints: 32768



Inexorably, the grains pile up.
How many on square 64?

You type: `PRINT 2^63`

It prints: 9.223372E+18

Yup, that's a lot of wheat, more wheat than existed in all the kingdoms everywhere. The king realized that he had been duped.

The king was: a) chagrined b) overjoyed
c) amused d) befuddled
e) angry f) livid
g) _____ (your choice)



Please pick one of the above and write the end of the story.

Postage & Handling: Add \$2.00 for first item and \$1.00 for each additional item. To shipping points outside U.S., add \$3.00 for first item and \$1.00 for each additional item. Shipped via surface mail.

QuickBASIC Made Easy

by Bob Albrecht, Wenden Wiegand,
and Dean Brown
(Osborne/McGraw-Hill, 350pp)

Learn how to program with Microsoft's QuickBASIC using Osborne/McGraw-Hill's popular "Made Easy" format. For beginning programmers or those experienced in other languages, *QuickBASIC Made Easy* is a step-by-step introduction to reading and writing programs with versions of QuickBASIC through version 4.5. You'll find information on creating files, building a toolkit, editing, and debugging. *QuickBASIC Made Easy* will keep you up-to-date with the latest changes in Microsoft's outstanding compiler.

OMH-881421 QuickBASIC Made Easy \$19.95

Advanced QuickBASIC

by Don Inman, Bob Albrecht, and
Arne Jamtgaard
(Osborne/McGraw-Hill, 500pp)

As an experienced programmer you'll learn how to write professional-level programs using the advanced techniques found in the 4.5 version of QuickBASIC. *Advanced QuickBASIC*, written by three highly regarded professional programmers, provides you with the tools to produce sophisticated programs. Graphics, construction kits, and data bases are some of the topics discussed. Find information on Quick libraries, macros, keyed, sequential, and unsequential files and more. *Advanced QuickBASIC* is packed with applications, examples, and models that will soon have you programming like an expert.

OMH-881361 Advanced QuickBASIC \$21.95

Using QuickBASIC

By Don Inman and Bob Albrecht
(Osborne/McGraw-Hill, 436pp)

Here's an excellent programming guide to Microsoft's newest version of QuickBASIC by the authors of *The BASIC Teacher*. The book approaches QuickBASIC's programming environment in three stages so beginning and experienced BASIC programmers can find the appropriate level of instruction.

OMH-881274 Using QuickBASIC \$19.95

QB4.5 Work Disk

LAST CHANCE!

Final chance for teachers and educators to get the exclusive QuickBASIC 4.0 Work Disk from Microsoft, available only thru *The BASIC Teacher*. The Work Disk contains QB.EXE and QB.HLP, the two most-important QuickBASIC files.

Requests must be made on school letterheads. The disk must not be duplicated and may be used only by the person making the request.

MS-QBWD QuickBASIC Work Disk \$12.00

DOS Made Easy

By Herbert Schildt
(Osborne/McGraw-Hill, 385pp)

Previous computer experience is not necessary to understand this concise, well-organized introduction that's filled with short applications and exercises. The book walks you thru all the basics, beginning with an overview of a computer system's inner components and a step-by-step account of how to run DOS for the first time.

OMH-881194 DOS Made Easy \$18.95

The Shareware Book Using PC-Write, PC-File, PC-Talk

By Emil Flock, et al
(Osborne/McGraw-Hill, 688pp)

Covers the most popular "free" programs: *PC-Write*, a word processor; *PC-File*, a database manager; and *PC-Talk*, a telecommunications program. These programs are available thru user groups or bulletin board services in return for a nominal registration fee. The book has all the details on how you can obtain these program disks.

OMH-881251 The Shareware Book \$14.95

Different Worlds Publications

2814 - 19th Street
San Francisco, CA 94110

Address Correction Requested

BULK RATE
U.S. Postage
PAID
San Francisco, CA
Permit No. 11798

The BASIC Teacher

Back Issues

TBT-1 Issue No. 1	\$3.00
TBT-2 Issue No. 2	\$3.00
TBT-3 Issue No. 3	\$3.00
TBT-4 Issue No. 4	\$3.00
TBT-5 Issue No. 5	\$3.00



MIACHEL K ERICKSON # 14
BOX 250
MONTE RIO CA 95462-0250



No. 7, May 1989

\$3.00

*For beginning
programmers with no prior
programming experience*

CONTENTS

- 3 Teach Yourself BASIC
- 13 Browsing BASIC
- 23 Teach Yourself
QuickBASIC

PUBLISHER'S STATEMENT: *The BASIC Teacher* is published monthly by Different Worlds Publications, 2814 - 19th Street, San Francisco, CA 94110. Contents copyright © 1989 by Different Worlds Publications. All rights reserved. Contents may be copied and distributed freely. Address all correspondences to *The BASIC Teacher*, 2814 - 19th Street, San Francisco, CA 94110.

SUBSCRIPTION INFO: A 12-issue sub in the U.S. and Canada is \$36. Overseas subs are \$42 by surface mail, \$54 by air.

PRINTED IN THE U.S.A.

BASIC



Happy Computing!

Tadashi Ehara, publisher
Bob Albrecht, editor
Don Inman, editor

This issue we present many new resources for your learning pleasure. Use order form on page 23.

Advanced QuickBASIC

by Don Inman, Bob Albrecht, and Arne Jamtgaard
(Osborne/McGraw-Hill, 500pp)

As an experienced programmer you'll learn how to write professional-level programs using the advanced techniques found in the 4.5 version of QuickBASIC. *Advanced QuickBASIC*, written by three highly regarded professional programmers, provides you with the tools to produce sophisticated programs. Graphics, construction kits, and data bases are some of the topics discussed. Find information on Quick libraries, macros, keyed, sequential, and unsequential files and more. *Advanced QuickBASIC* is packed with applications, examples, and models that will soon have you programming like an expert.

OMH-881361 Advanced QuickBASIC \$21.95

Using QuickBASIC 4.5 Second Edition

By Don Inman and Bob Albrecht
(Osborne/McGraw-Hill, 500pp)

Here's an excellent programming guide to Microsoft's newest version of QuickBASIC by the authors of *The BASIC Teacher*. The book approaches QuickBASIC's programming environment in three stages so beginning and experienced BASIC programmers can find the appropriate level of instruction. You'll learn about subprograms, libraries, meta-commands, dynamic debugging, and the important advantage of QuickBASIC's speedy graphics.

OMH-881514 Using QuickBASIC 4.5 \$22.95

Still Available

OMH-881274 Using QuickBASIC 4.0 \$19.95

QuickBASIC Made Easy

by Bob Albrecht, Wenden Wiegand, and Dean Brown
(Osborne/McGraw-Hill, 350pp)

Learn how to program with Microsoft's QuickBASIC using Osborne/McGraw-Hill's popular "Made Easy" format. For beginning programmers or those experienced in other languages, *QuickBASIC Made Easy* is a step-by-step introduction to reading and writing programs with versions of QuickBASIC through version 4.5. You'll find information on creating files, building a toolkit, editing, and debugging. *QuickBASIC Made Easy* will keep you up-to-date with the latest changes in Microsoft's outstanding compiler.

OMH-881421 QuickBASIC Made Easy \$19.95

Microsoft QuickBASIC Version 4.5

Microsoft QuickBASIC 4.5 is a complete BASIC learning system. The new interactive, on-disk tutorial, Microsoft QB Express, quickly and easily introduces you to the Microsoft QuickBASIC environment. A new step-by-step tutorial guides you through an actual application. And numerous example programs help you master BASIC programming.

MS-04366 QuickBASIC 4.5 \$99.00
Now only \$79.00

DOS Made Easy

By Herbert Schildt
(Osborne/McGraw-Hill, 385pp)

Previous computer experience is not necessary to understand this concise, well-organized introduction that's filled with short applications and exercises. The book walks you thru all the basics, beginning with an overview of a computer system's inner components and a step-by-step account of how to run DOS for the first time.

OMH-881194 DOS Made Easy \$18.95



Special Note to Subscribers

Due to circumstances beyond our control, this issue is late. The authors have been *extremely* busy writing several computer books. But they are now finished and we should be back on schedule. Subscribers will get the full number of issues promised. The number next to your name on the mailing label on the back page is the last issue in your sub.

P.S.—The books Bob and Don were writing are *QuickBASIC Made Easy*, *Advanced QuickBASIC*, a new edition of *Using QuickBASIC*, and *GW-BASIC Made Easy*. Except for the *GW-BASIC* book, all are available now and can be ordered thru *The BASIC Teacher*.

QuickBASIC: The Complete Reference

by Steven Nameroff
(Osborne/McGraw-Hill, 700pp)

A comprehensive guide for users of all levels of programming ability from novices to pros. Nameroff has divided the book into sections to help you easily locate the information you need. *QuickBASIC: The Complete Reference* begins with a quick introduction to BASIC programming, followed by a complete command reference section and a discussion of QuickBASIC functions, procedures, files, and graphics. Advanced techniques and applications are grouped together in the last section of the book for the professional QuickBASIC programmer.

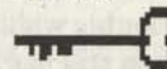
OMH-881362-X QuickBASIC: The Complete Reference \$26.95



Introduction

BASIC HAS a small **vocabulary** and a simple **syntax** (grammar). We have already discussed some of the special words that Microsoft BASIC understands. They are called **keywords** or **reserved words**. Here are the keywords introduced and described previously:

BEEP	GOTO	LOCATE	PRINT	TIMES
CLS	INPUT	NEW	RANDOMIZE	TIMER
COLOR	INT	NEXT	RND	WIDTH
DATE\$	KEY	OFF	RUN	
FOR	LIST	ON	SOUND	



We always show keywords in all upper-case letters and variables in lower-case or a MiXtuRe of upper- and lower-case letters. We think this makes programs easier to read and understand. However, when you LIST a program, unfortunately, variables will probably appear in all upper-case letters.

Happy Birthday, Mom!

HERE IS a tiny program to print a birthday message for mom in big letters near the center of the screen.



```
10 CLS : KEY OFF
20 WIDTH 40
30 msg$ = "Happy Birthday, Mom!"
40 COLOR 5
50 LOCATE 12, 9, 0: PRINT msg$
60 anykey$ = INPUT$(1)
70 WIDTH 80: CLS
```

Enter and run the program. If all goes well, you will see:

Happy Birthday, Mom!

in marvelous magenta near the center of the screen.

Make It Blink

WHILE MOM gazes appreciatively at the screen, bring in the cake. After the party is over, you can press a key and the program will stop with the screen in its normal 80 characters per line mode.

Ok

—

Now change line 40 so that the message blinks the next time you run the program. Use a color number from 17 to 31. To blink in magenta, use color number 21, then run the blinking program.



It Works Like This

HERE IS a line-by-line description of the program:

Line 10 has two statements separated by a colon (:). The first statement (CLS) clears the screen; the second statement (KEY OFF) turns off the key line.

10 CLS : KEY OFF

1st statement colon 2nd statement

Line 20 sets the screen to 40 characters per line instead of the "normal" 80 characters per line. Each character will be double width. The screen still has 25 lines.

20 WIDTH 40

Line 30 assigns the string:

Happy Birthday, Mom!

as the value of the string variable msg\$.

30 msg\$ = "Happy Birthday, Mom!"



Line 40 chooses non-blinking magenta as the color in which text will be printed.

40 COLOR 5

Line 5 has two statements separated by a colon (:). The first statement (LOCATE 12, 9, 0) puts the cursor at line 12, row 9, and makes it invisible (0). The second statement (PRINT msg\$) prints the value of variable msg\$. This value was assigned in line 30. It will be printed in the color selected by line 40.

50 LOCATE 12, 9, 0: PRINT msg\$

Line 60 tells the computer to wait for one key press. Almost any key will do. The string value of the key is assigned as the value of the string variable anykey\$. If you press a key, the computer goes on to line 70.

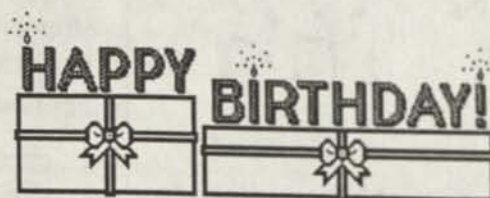
60 anykey\$ = INPUT\$(1)

Line 70 has two statements separated by a colon (:). The first statement returns the screen to 80 characters per line for new text. The second statement (CLS) clears the screen.

70 WIDTH 80: CLS

Use REM and Apostrophes (')

HERE IS an annotated version of the happy birthday program. It uses REM statements and apostrophes (') to begin comments that tell you about the program.



```
1 REM ** Happy Birthday, Mom! **
2 ' The BASIC Teacher #7.  Filename: TYB0701.ASC
```

```
10 CLS : KEY OFF
20 WIDTH 40
30 msg$ = "Happy Birthday, Mom!"
40 COLOR 5
50 LOCATE 12, 9, 0: PRINT msg$
60 anykey$ = INPUT$(1)
70 WIDTH 80: CLS
```

```
'Clear screen & turn off key line
'Set screen width to 40 characters
'Here is the message to be printed
'Magenta
'Print message centered on screen
'Wait for a key press
'Return screen to normal
```


REM

IT IS good practice to put information in a program to tell people about the program. The REM (REMark) statement allows you to do this. Any text that follows REM in a program line is ignored when you run the program. You can use an apostrophe (') as an abbreviation for REM.

From now on, most of our programs will begin with a REM statement in line 1 that has the name of the program.

```
1 REM ** Happy Birthday, Mom! **
```

Line 2 will tell where the program appears and its file name, as stored on our disk.

```
2 ' The BASIC Teacher #7. Filename: TYB0701.BAS
```

Line 2 tells you the program can be found in issue 7 of **The BASIC Teacher**. Its file name is **TYB0701.BAS**, which means "Teach Yourself BASIC #7, program #1."

You can use an apostrophe to put a comment on a line. Everything to the right of the apostrophe is ignored when the computer runs the program.

```
10 CLS : KEY OFF 'Clear screen & turn off key line
```

SAVE TYB0701.BAS

Enter program **TYB0701.BAS**, including all remarks, into the computer's memory. You can save the program to a disk, then load it back into memory whenever you wish.

Save the program to disk drive A. Well, if you prefer drive B, change A to B in what follows.

Type: `SAVE "A:TYB0701"`

and press ENTER.

The computer will add the **file extension** **.BAS** and save the program under the file name **TYB0701.BAS**.

Well, that's the long way to save the program. There is also a short way. You can press the F4 function key to type SAVE and a quotation mark ("). Then type the file name and press ENTER. Do it. Press F4 and you will see:

`SAVE"`

Then type the file name and press ENTER.

Type: `A:TYB0701"`

and press ENTER.

That's it!

LIST Program TYB0701.BAS

UNFORTUNATELY, MANY older versions of BASIC list programs as shown below, with line spacing removed and variables in all upper case letters. That's one reason why we do all our work in QuickBASIC, except when we are writing stuff about other BASICs.

```
1 REM ** Happy Birthday, Mom! **
2 ' The BASIC Teacher #7. Filename: TYB0701.ASC
10 CLS : KEY OFF
20 WIDTH 40
30 MSG$ = "Happy Birthday, Mom!"
40 COLOR 5
50 LOCATE 12, 9, 0: PRINT MSG$
60 ANYKEY$ = INPUT$(1)
70 WIDTH 80: CLS
```

```
'Clear screen & turn off key line
'Set screen width to 40 characters
'Here is the message to be printed
'Magenta
'Print message centered on screen
'Wait for a key press
'Return screen to normal
```



Is It on the Disk?

IS THE program really stored on the disk in drive A (or the disk drive *you* used)? Use the keyword FILES as a direct statement to find out what is on the disk in drive A.

Type: FILES "A:"

and press ENTER.

The computer displays all files on the disk in drive A. We don't know what is on the disk in your drive A, but here is what is on the disk in our drive A:

```
A:\
COMMAND.COM      GWBASIC.EXE      TYB0701.BAS
586973 Bytes free
Ok
-
```

On our disk drive A (A:\) there are three files: **COMMAND.COM** (from MS-DOS), **GWBASIC.EXE**, and the program we just stored, **TYB0701.BAS**.

If you want to list the files on the disk in disk drive B, type the FILES command like this:

FILES "B:"

If the disk contains many files, you may wish to list the names of only the BASIC programs with the .BAS file extension. In this case, type the FILES command like this:

FILES "A:*.BAS"

or

FILES "B:*.BAS"

The asterisk (*) is called a "wild card." It can be used in certain MS-DOS commands as well as in GW-BASIC. Consult your MS-DOS reference manual for more information on the use of the asterisk as a wild card.



The Default Disk Drive

WE USE a Tandy 1000TX with two floppy disk drives, drive A and drive B. We load MS-DOS into drive A, then load GW-BASIC at the MS-DOS command line, as follows:

A>GWBASIC

So, disk drive A is our **default disk drive**. It is automatically used if we do not designate a different drive. If we want to save a program on our default drive (drive A), we can do it without mentioning the drive letter, like this:

SAVE "TYB0701"

The above SAVE command saves program **TYB0701** to the default disk drive with the file name **TYB0701.BAS**.

If we want to see a list of names of files on the default disk drive, we use the FILES command as shown below.

FILES

If we want to see only the names of BASIC programs (.BAS) on the default disk drive, we do it like this:

FILES "*.BAS"

If you have a hard disk perhaps you are working out of it. You see a C> instead of A>. In this case, your hard disk is your default disk drive.



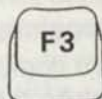
LOAD TYB0701

YOU CAN load a program stored on a disk into the computer's memory. For example, suppose you want to load program **TYB0701.BAS** from the default disk drive. You can do it in two ways, the short way or the long way. First, the long way.

Type: `LOAD "TYB0701"`
and press ENTER.

And now, the short way. Use the F3 function key as a shortcut for typing `LOAD`", as follows:

Press the F3 key. The keyword `LOAD`, a quotation mark, and the blinking cursor appear.



```
LOAD" _
```

Type: `TYB0701"`
and press ENTER.

After using the short way to load the program, the screen should look like this:

```
LOAD"TYB0701"
Ok
_
```

The program is in memory, ready for your use. You can list it, run it, or do whatever else you want to do with it. Note that you do not have to type the file extension (`.BAS`) as part of the file name, altho it is Ok to do so. The space between `LOAD` and the first quotation mark (`"`) is optional. The final quotation mark (after the file name) may be omitted.

If you misspell the file name or try to load a non-existent file, you will see a "File not found" message. That's Ok—just try again.

Saving in ASCII

EARLIER, YOU saved program **TYB0701** to a disk. It is saved in **compressed binary format**, a format peculiar to GW-BASIC. This format is very efficient, using a minimum amount of disk space.

You can also save a program as an ASCII text file. This method is less efficient—the program occupies more space on the disk. However, there is an advantage: the program can be read by a word processor. You can also use the MS-DOS `TYPE` command to type an ASCII program on the screen.

Save program **TYB0701** as an ASCII text file. But give it a new file name so you don't erase the previously stored binary file. Call it **TYB0701A**.

Type: `SAVE"TYB0701A", A`
and press ENTER.

Of course, you can save time by using the F4 function key to type `SAVE`". In the above `SAVE` command, the letter `A` following the comma tells the computer to save the program as an ASCII text file.

```
SAVE"TYB0701A", A
      save in ASCII
```

Use a `FILES` command to verify that the program is on the disk. Its file name should appear as **TYB0701A.BAS**. The file name does not tell you that the program is an ASCII file. This presents no problem to GW-BASIC. You can load the program into memory in the same way you load a program stored in compressed binary format.

You can also save the program as an ASCII text file with a file extension that tells you it is an ASCII file. For example, save the program one more time by using the following `SAVE` command:

Type: `SAVE"TYB0701.ASC", A`

Return to MS-DOS

CLEAR THE screen, then use a SYSTEM command to return to MS-DOS.

Type: SYSTEM
and press ENTER

If disk drive A is your default disk drive, you will see the familiar "A prompt."

A>_

Use the DIR command to display a list of file names on the screen.

Type: DIR
and press ENTER.

Here is what we see:

COMMAND	COM	23612	7-21-87	3:00p
GWBasic	EXE	72240	7-21-87	3:00p
TYB0701	BAS	549	3-11-89	7:49a
TYB0701A	BAS	562	3-11-89	8:23a
TYB0701	ASC	562	3-11-89	8:24a
5 File(s)		583966 bytes free		

The disk now contains our "Happy Birthday, Mom!" program stored three times under three different file names. Since **TYB0701A.BAS** and **TYB0701.ASC** are ASCII files, you can use the MS-DOS TYPE command to display them on the screen. Do this now. Tell the computer to display the **TYB0701.ASC** file.

Type: TYPE TYB0701.ASC
and press ENTER.

Well, we hope you now see the program on the screen.



Practice

LOAD BASIC again and enter this tiny program:

```
1 REM ** Beep, Date, and Time **
2 ' The BASIC Teacher #7.  Filename: TYB0702.BAS
```

```
10 CLS
20 BEEP
30 PRINT DATE$
40 PRINT TIME$
```

Save the program as a compressed binary file.

```
SAVE"TYB0702"
```

Remember, GW-BASIC adds .BAS. Now save the program as an ASCII file.

```
SAVE"TYB0702.ASC", A
```

The program is now on the disk in two ways: (1) as a compressed binary file with file name **TYB0702.BAS**, and (2) as an ASCII file with file name **TYB0702.ASC**.

Load file **TYB0702.BAS** like this:

```
LOAD"TYB0702"
```

Load file **TYB0702.ASC** like this:

```
LOAD"TYB0702.ASC"
```

Of course, also list the program and run it. Then use SYSTEM to exit BASIC and return to MS-DOS. While in MS-DOS, TYPE the program on the screen.

```
A>TYPE TYB0702.ASC
```

What will happen if you try to TYPE the **TYB0702.BAS** file on the screen? Try it and find out.

What would you like us to talk about in "Teach Yourself BASIC"?

REMEMBER: This series is for **beginners**. Write to:

Bob & George
PO Box 1635
Sebastopol, CA 95473

By Don Inman

THE FOLLOWING topics were covered in previous installments of "Browsing BASIC":

- #1: The **KEY** command and ALT-key shortcuts
- #2: The **VIEW PRINT** statement and pull-down windows
- #3: The **COPY CON** command to write a **DRIVER.SYS** file and disk labels
- #4: Text and Graphic Position Relationships
- #5: Enclosing Text in Ellipses
- #6: Using DOS Text Files



In "Browsing BASIC #5" we demonstrated text and graphics position relationships by enclosing text within a graphic shape. We will delve deeper into the uses that may be made of these relationships this month. We will explore the movement of a graphics shape within a block of text. To do this, we must introduce the following graphics keywords and related terms:

GET POINT PUT PSET VAL XOR

GRAPHICS ARRAYS

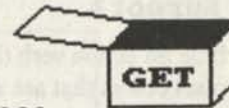
INFORMATION ABOUT a graphic shape can be placed in an array by using GW-BASIC's **GET** graphics statement. The shape can be put back on the screen at any position with a **PUT** statement. The following short program draws a small rectangle in the upper left corner of the screen, stores it in an array with **GET**, and places it in the upper right corner of the screen with **PUT**.

```

1 REM ** GET and PUT A Box **
2 ' Program 7-1 8/8/88
3 ' Microsoft GW-BASIC File: TINYPRO.001

100 REM ** Initialize **
110 SCREEN 1: CLS
120 DIM Kursor%(10)           ' dimension array

200 REM ** GET and PUT Box **
210 LINE (1, 1)-(8, 8), 1, B   ' draw 8 x 8 box
220 GET (1, 1)-(8, 8), Kursor% ' save in array named Kursor
230 Begin! = TIMER            ' start timer
240 WHILE TIMER < Begin! + 2   ' loop for 2 seconds
250 WEND
260 PUT (310, 1), Kursor%      ' put box in new position
270 key$ = INPUT$(1)           ' wait for a keypress
    
```



After the screen is cleared, the array is dimensioned. The size of the array is dependent on the **SCREEN** mode used and the numeric type used for the array. The smaller the array, the quicker it can be stored and retrieved.

SCREEN 1, which is used in this program, needs 2 bits per pixel to describe a pixel. Since integers require less space than other numeric types, we used an integer array for the shape named **Kursor**. The dimension size is calculated as follows:

$$\begin{aligned} \text{bytes} &= 4 + \text{INT}(((\text{col2} - \text{col1} + 1) * 2 + 7) / 8) * ((\text{row2} - \text{row1}) + 1) \\ &\quad \text{difference in first} \quad \text{bits per} \quad \text{difference in first} \\ &\quad \text{and last column} \quad \text{pixel} \quad \text{and last row} \\ &= 4 + \text{INT}(((8 - 1 + 1) * 2 + 7) / 8) * ((8 - 1) + 1) \\ &= 4 + \text{INT}(23 / 8) * 8 = 4 + 2 * 8 = 20 \end{aligned}$$

For an integer array, the number of bytes per element is two. Therefore, the number of array elements needed for the dimension statement is $20 / 2$, or 10.

```
DIM Kursor%(10)
```

The **LINE** statement draws a cyan (**color = 1**) box.

A graphics array is saved as a rectangular area. The **GET** statement specifies the coordinates of opposite corners of the rectangular area to be saved. A comma separates the coordinates from the array name.

```
GET (1, 1)-(8, 8), Kursor%
```

opposite corners array name

We then inserted a two-second time delay so that you could see the original shape first—then the placement of the shape in a new position.

The upper left corner of the rectangular area will be placed at the position specified in the **PUT** statement. This position is followed by a comma and the name under which the array was saved (by **GET**).

```
PUT (310, 1), Kursor%
```

The **PUT** statement can include an action verb that determines the interaction between the shape and the pixel colors that are already on the screen when the **PUT** statement is executed. The action verbs are: **PSET**, **PRESET**, **AND**, **OR**, and **XOR**. The GW-BASIC default action verb is **PSET**. When you do not specify an action verb, **PSET** is used. If you are programming in Quick-BASIC, **XOR** is the default action verb.

The two action verbs of immediate concern are **PSET** and **XOR**.

PSET Transfers the shape point-by-point to the screen using the colors the shape had when it was taken from the screen by a **GET** statement.

XOR Causes the points on the screen to be inverted where a point exists in the shape stored in the array. When a shape is put on the screen a second time at the same position using **XOR**, the original screen is restored. This allows you to move a shape around on the screen without destroying the original screen.

The second small program draws the same box as the first program, saves it in an array, and places the box at a new position with the **PSET** action verb. It then puts it on the screen at this new position with the **XOR** action verb. This last action erases the box, restoring the original screen.

```

1 REM ** Draw Box, PUT It, Erase It **
2 ' Program 7-2 8/10/88
3 ' Microsoft GW-BASIC File: TINYPRO.002

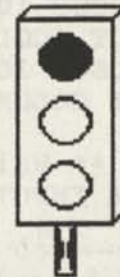
100 REM ** Initialize **
110 SCREEN 1: CLS
120 DIM Kursor%(10)           ' dimension array

200 REM ** Draw Box, PUT It Twice **
210 LINE (1, 1)-(8, 8), 1, B   ' draw 8 x 8 box
220 GET (1, 1)-(8, 8), Kursor% ' save in array named Kursor
230 GOSUB 1010                 ' delay
240 PUT (310, 1), Kursor%, PSET ' display box in new position
250 GOSUB 1010                 ' delay
260 PUT (310, 1), Kursor%, XOR ' erase box

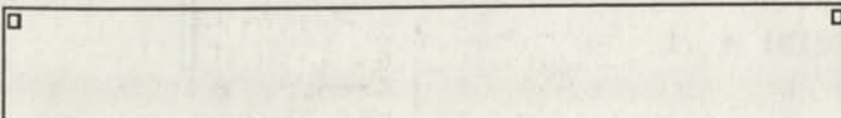
300 REM ** Wait for Keypress and End **
310 ky$ = INPUT$(1)           ' wait for keypress
320 END

1000 REM ** Subroutine: Time Delay **
1010 Begin! = TIMER
1020 WHILE TIMER < Begin! + 2
1030 WEND
1040 RETURN

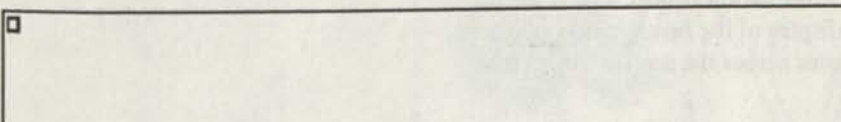
```



The box is drawn first in the upper left corner of the screen. The first **PUT** statement then puts the same box with its upper left corner at position 310, 1 with the **PSET** statement. The top of the screen shows:



The second **PUT** statement (with action verb **XOR**) then erases the box in the upper right corner of the screen.



MOVING THE BOX

THIS TECHNIQUE of using a second **PUT** statement at the same position is used to animate graphic shapes. The box can be "moved" across the screen by successively placing the box at a new location with a **PUT** statement and erasing it with a second **PUT** statement.

Program 7-3, Moving Box, uses this technique to move the box across the screen:

```

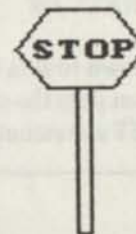
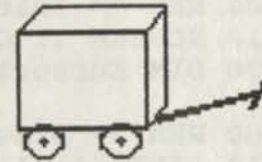
1 REM ** Moving Box **
2 ' Program 7-3 8/15/88
3 ' Microsoft GW-BASIC File: TINYPRO.003

100 REM ** Initialize **
110 SCREEN 1: CLS
120 DIM KURS0R%(10)           ' dimension array
130 LINE (1, 1)-(8, 8), 1, B   ' draw 8 x 8 box
140 GET (1, 1)-(8, 8), KURS0R% ' save in array named Kursor
150 GOSUB 1010

200 REM ** Move the Box **
210 FOR COLUMN% = 1 TO 310 STEP 10
220   PUT (COLUMN%, 10), KURS0R%, XOR ' turn on
230   GOSUB 1010
240   PUT (COLUMN%, 10), KURS0R%, XOR ' turn off
250   GOSUB 1010
260 NEXT COLUMN%

300 REM ** Wait for Keypress, Then End **
310 A$ = INPUT$(1)
320 END

1000 REM ** Delay Subroutine **
1010 BEGIN! = TIMER
1020 WHILE TIMER < BEGIN! + .1
1030 WEND
1040 RETURN
  
```



Enter and run Program 7-3. Each time through the **FOR** . . . **NEXT** loop, the box is **PUT** on the screen at row 10 and the current column value. After a short time delay (shortened to one-tenth of a second), the box is **PUT** again at the same position. This erases the box. After another short time delay the **column%** is incremented by 10 for the display of the box at a new position. The result is that the box apparently moves across the screen from left to right.

If you want to move the box across the screen in smaller steps, you can change the **STEP** value in the **FOR** statement. You may also want to remove the time delay caused by the **GOSUB** in line 250. This would place the box at a new position immediately after it is erased from the previous position. You could also add to the program by moving the box down a line each time so that it moves from left to right, down a line and right to left, down a line and left to right, etc.

CHANGING TEXT COLORS

WHEN YOU are working in the **SCREEN 1** graphics mode, text is printed in the default color value 3 within the palette selected in a **COLOR** statement.

COLOR 0, 1

background palette 1 (cyan, magenta, white)

If no **COLOR** statement has been executed, the default palette of 1 is used with a black background. Therefore, the text would be printed as white (color value 3) on black.

At times, you may want to highlight a character, a word, or a group of words in a block of text by changing colors. You can use **PUT** and **GET** statements to move a cursor (such as the box of Program 7-3) along a line of text. Then you can change the color of the character by using **PSET** to set the color of each pixel under the cursor. Program 7-4 allows you to do this.

A line of text is printed, and the cursor positioned over the first character in the line.

1. Use the left and right arrow keys to move the cursor from character to character in the line.
2. Press a number (1, 2, or 3) to change the color of the character under the cursor.

1 = cyan 2 = magenta 3 = white

3. Press the **ENTER** key when colors have been set as desired. This removes the cursor from the screen so that you can examine the result.
4. Press any key to end the program.

Here is some colored text from a sample run of Program 7-4.

Move the cursor or change color.




```

1 REM ** Move Cursor and Change Text Colors **
2 ' Program 7-4 8/20/88
3 ' Microsoft GW-BASIC File: TINYPRO.004

```

```

100 REM ** Initialize **
110 SCREEN 1: CLS
120 DIM Kursor%(10)           ' dimension array
130 LINE (1, 1)-(8, 8), 1, B   ' draw 8 x 8 box
140 GET (1, 1)-(8, 8), Kursor  ' save in array named Kursor
150 CLS: LOCATE 2,2
160 PRINT "Move the cursor or change color.";

```

```

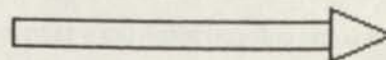
200 REM ** Put Cursor and Make Changes **
210 column% = 2: row% = 2
220 col% = column% * 8 - 8: lyne% = row% * 8 - 8
230 PUT (col%, lyne%), Kursor%  ' Kursor on
240 WHILE ky$ <> CHR$(13)
250   ky$ = ""
260   WHILE ky$ = ""
270     ky$ = INKEY$
280   WEND
290   IF ky$ > "0" AND ky$ <= "3" THEN GOSUB 3010
300   IF ky$ <> CHR$(0)+"M" AND ky$ <> CHR$(0)+"K" AND
      ky$ <> CHR$(13) THEN GOTO 250
310   PUT (col%, lyne%), Kursor%, XOR           'Kursor off
320   IF ky$ = CHR$(0) + "M" THEN GOSUB 1010
330   IF ky$ = CHR$(0) + "K" THEN GOSUB 2010
340 WEND
350 ky$ = INPUT$(1)
360 END

```

```

1000 REM ** Subroutine: Move Right **
1010 col% = col% + 8
1020 IF col% > 256 THEN col% = 8
1030 PUT (col%, lyne%), Kursor%, XOR
1040 RETURN

```

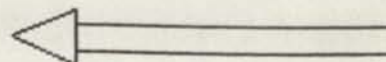


' turn on

```

2000 REM ** Subroutine: Move Left **
2010 col% = col% - 8
2020 IF col% < 8 THEN col% = 256
2030 PUT (col%, lyne%), Kursor%, XOR
2040 RETURN

```



' turn on

```

3000 REM ** Subroutine: Color **
3010 PUT (col%, lyne%), Kursor%, XOR
3020 kolor% = VAL(ky$)
3030 FOR pnt% = col% TO col% + 8
3040   FOR tier% = lyne% TO lyne% + 7
3050     IF POINT(pnt%, tier%) <> 0 THEN PSET(pnt%, tier%), kolor%
3060   NEXT tier%
3070 NEXT pnt%
3080 PUT (col%, lyne%), Kursor%, XOR
3090 RETURN

```

' turn off



' turn on

The left and right arrow keys allow you to move the cursor along the line of text. The arrow keys use two ASCII codes (called Extended codes) for recognition:

Right arrow: Null code (CHR\$(0)) + ASCII code for letter M

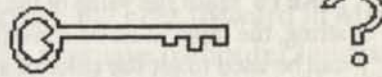
Left arrow: Null code (CHR\$(0)) + ASCII code for letter K

The **INKEY\$** function is used to look for a press of the left or right arrow keys in Program 7-4.

```

250 ky% = ""
260 WHILE ky$ = ""
270   ky$ = INKEY$
280 WEND
.
.
.
320 IF ky$ = CHR$(0) + "M" THEN GOSUB 1010
330 IF ky$ = CHR$(0) + "K" THEN GOSUB 2010

```



If the right arrow key is pressed, control is passed to the Move Right subroutine:

```

1010 col% = col% + 8           ' move right 8 pixels
1020 IF col% > 256 THEN col% = 8 ' if end, go to beginning
1030 PUT(col%, lyne%), Cursor%, XOR ' turn on
1040 RETURN

```

If the left arrow key is pressed, control is passed to the Move Left subroutine:

```

2010 col% = col% - 8           ' move left 8 pixels
2020 IF col% < 8 THEN col% = 256 ' if beginning, go to end
2030 PUT(col%, lyne%), Cursor%, XOR ' turn on
2040 RETURN

```

Provision is made in the Move subroutines to "wrap around" when reaching either end of the line.

The change color option is provided in another subroutine which is accessed by pressing one of the number keys: 1, 2, or 3.

```

290 IF ky$ > "0" AND ky$ <= "3" THEN GOSUB 3010
.
.
3010 PUT (col%, lyne%), Cursor%, XOR ' turn off
3020 kolor% = VAL(ky$)
3030 FOR pnt% = col% TO col% + 8 ' all columns of area
3040   FOR tier% = lyne% TO lyne% + 7 ' all rows of area
3050     IF POINT(poynt%, tier%) <> 0 THEN PSET(pnt%, tier%), kolor%
3060   NEXT tier%
3070 NEXT pnt%
3080 PUT (col%, lyne%), Cursor%, XOR ' turn on
3090 RETURN

```


This subroutine first turns off the cursor so that its colors will not interfere with the area being examined. The subroutine looks at each pixel within the selected character block. If the color is different from the background color (zero), the pixel is **PSET** to the color whose number has been pressed (1, 2, or 3). When all the pixels have been set, the cursor is put back on the screen. Control is then returned to the main program.

Two functions that we have not discussed are used in the Color subroutine.

VAL This function returns the numeric value of a string. Since **INKEY\$** reads the value of the specified color number as a string, the input must be converted to numeric format before it can be used to set the color of a point.

```
3020 kolor% = VAL(ky$)
```

|
numeric value of ky\$ assigned to kolor%

POINT This function can either read the color number of a pixel from the screen, or it can return a pixel's coordinates. In this program, we want to read the color number of a pixel from the screen.

```
IF POINT(pnt%, tier%) <> 0 THEN PSET(pnt%, tier%), kolor%
```

False if point is black
(background color) - leave it alone.

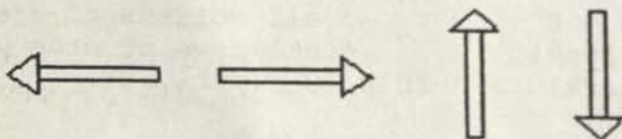
Set point to kolor%
if point is not black.

NEXT TIME

WE WILL continue this discussion next time as we extend the program so that more than one line of text can be scanned. Then you will be able to move the cursor anywhere on the screen and change colors wherever you desire.

In the meantime, study the Move Right and Move Left subroutines. We will have to use a similar technique to move up and down the screen. We will also want to modify these two subroutines so that moving right at the end of a line wraps around to the beginning of the line below. Similarly, moving left at the beginning of a line should wrap around to the end of the line above.

We will put Move Up and Move Down blocks in separate subroutines. In addition, we'll show how QuickBASIC can perform the same tasks much more efficiently.





QuickBASIC Interpreter

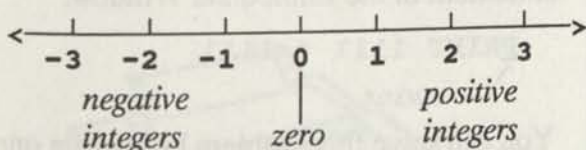
YES, ANOTHER big step forward in Microsoft's quest for the ultimate computer language. The **QuickBASIC Interpreter: Academic Edition** is here. It is even better for beginners than **QB 4.5**, and much less expensive. However, it is available only to the academic world—you must provide "educational identification" at time of purchase (it says on the package). **QBI** is a disk set only—no paper documentation. However, the disks contain an on-line tutorial and a hypertext-based help system, including a complete reference guide.

If you are an interested educator, contact:

Microsoft Corporation
PO Box 97017
Redmond, WA 98073-9717

Integers

INTEGERS ARE the counting numbers (1, 2, 3, and so on), the opposites, or negatives, of the counting numbers (-1, -2, -3, and so on), and the number zero (0). The counting numbers (1, 2, 3, ...) are also called positive integers. The opposites, or negatives, of the counting numbers (-1, -2, -3, and so on) are also called negative integers.



Integers go on forever, in both the positive and negative directions.

QuickBASIC Integers

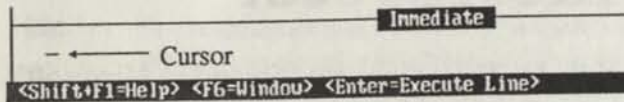
QUICKBASIC CAN handle a finite set of integers. It has two ways of representing integers for computational purposes. These are called **short integers** and **long integers**.

- A short integer is an integer in the range -32,768 to 32,767. It occupies only two bytes of the computer's memory.
- A long integer is an integer in the range -2,147,483,648 to 2,147,483,647. It occupies four bytes of the computer's memory.

Type an integer *without* using commas and *without* a decimal point. QuickBASIC will recognize, say, 32000 as a short integer and, for example, 40000 as a long integer.

A Number Pattern

TO LEARN more about short and long integers, explore a number pattern. Press the F6 function key to put the cursor in the Immediate Window, like this:



First, clear the Output screen:

Type: CLS

and press ENTER.

You will see an empty Output screen, except for the message "Press a key to continue." Press a key to return to the Immediate Window. Now do a little number-crunching with short integers. You type direct PRINT statements in the Immediate Window. It (the computer) prints answers in the Output screen.

You type: PRINT 11 * 11

It prints: 121

You type: PRINT 111 * 111

It prints: 12321

In both of these examples, a short integer is multiplied by a short integer. The result is also a short integer. But what happens if the result is not in the range (-32768 to 32767) for short integers? Well, let's find out.

You type: PRINT 1111 * 1111

Oops! Instead of seeing the answer, you see the Overflow dialog box.

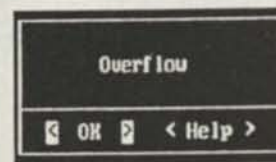


Overflow

THE OVERFLOW dialog box looks like this:



Oh, sorry. That's not it. It really looks more like this:



QuickBASIC is telling you that the result of the multiplication would be too large to be represented as a short integer. It would "overflow" the space (two bytes) allowed for short integers. Here are some interesting things you can do:

- Press the TAB key to move the highlight in the Overflow dialog box to **Help**, then press ENTER and explore QuickBASIC's wonderful on-line help system.
- Remove the Overflow dialog box and read on to learn how to fix the problem. When **OK** is highlighted, you can press ENTER to remove the box. Or, you can press the ESC key to remove the dialog box.

When you remove the Overflow dialog box, you will see the cursor blinking on the PRINT statement in the Immediate Window.

PRINT 1111 * 1111

— cursor

You can solve this problem by making one of the numbers into a long integer. Then QuickBASIC will treat the entire problem as a long integer problem and produce a long integer answer.

Use & for Big Integers

YOU CAN convert a short integer into a long integer by adding an ampersand (&) to the right end. This tells QuickBASIC to regard the number as a long integer. So move the cursor to the right end of the PRINT statement.

```
PRINT 1111 * 1111_
```

Type an ampersand (&) so it looks like:

```
PRINT 1111 * 1111&_
```

and press ENTER.

It prints: 1234321

Since either number can be changed to a long integer to correct the overflow problem, you can also do the following:

```
You type: PRINT 1111& * 1111
```

It prints: 1234321

Note that the result of this multiplication is too large to be a short integer, but is within the range for a long integer (-2147483648 to 2147483647).

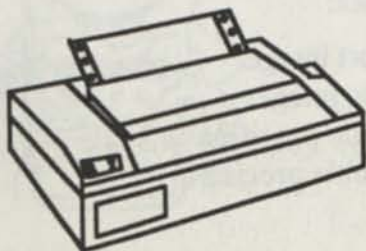
REMEMBER: If you multiply a short integer and a long integer, the result (product) is a long integer. Now continue this number pattern.

```
You type: PRINT 11111 * 11111&
```

It prints: 123454321

Of course, you can also type the PRINT statement like this:

```
PRINT 11111& * 11111
```



What? Overflow Again?

WELL, THE answers keep getting bigger. What will happen if ... ? Let's try the next one and see.

```
You type: PRINT 111111 * 111111&
```

There's that Overflow box again. The answer is too big to be a long integer. What do you think the answer is? Write it here:

The answer: _____

We will soon tell you about another kind of QuickBASIC number that you can use to check the answer you wrote. But first, try these:

```
You type: PRINT 11 * 111
```

It prints: 1221

```
You type: PRINT 11 * 1111
```

It prints: 12221

```
You type: PRINT 11 * 11111
```

The Overflow box pops up. Both 11 and 1111 are recognized by QuickBASIC as short integers, but the result of this multiplication is too big to be a short integer. You know how to fix it.

```
You type: PRINT 11 * 11111&
```

It prints: 122221

Betcha the next one surprises you.

```
You type: PRINT 11 * 111111
```

It prints: 1222221

QuickBASIC recognizes 111111 as a **long integer** and treats the entire problem as a long integer problem!

Single Precision

A **SINGLE precision number** can be an integer or non-integer with up to seven digits. You can designate a number as a single precision number by including a decimal point in the number, by using an explanation point (!) at the right end of the number, or by using the floating point notation described last time. The following numbers are recognized by QuickBASIC as single precision numbers.

```
3.14
1111.    or  1111!
5E+09    or  5E9
```

Try some single precision numbers in the number pattern.

You type: `PRINT 1111 * 1111.`

It prints: 1234321

You type: `PRINT 1111 * 1111!`

It prints: 1234321

Since 1111. and 1111! are single precision numbers, QuickBASIC treated these multiplications as single precision problems and produced a single precision result. Now let's see what happens if the result would require more than seven digits.

You type: `PRINT 11111 * 11111.`

It prints: 1.234543E+08

Of course! QB prints the result as a single precision floating point number. This result is a 7-digit approximation to the true result (123454321), which has 9 digits.

A single precision number can represent a number in the range $-3.37E+38$ to $3.37E+38$. It occupies 4 bytes of memory space.



Double Precision

A **DOUBLE precision number** can have up to 16 digits. QuickBASIC will recognize a number as double precision if it has a decimal point and is too big to be a single precision number. You can also designate a number as double precision by using a number sign (#) at the right end of the number or by using floating point notation with a D instead of an E. The following numbers are recognized as double precision numbers.

```
12345678.
11111#      .06#
5D9         1D23
```

Try some double precision numbers in the number pattern.

You type: `PRINT 11111 * 11111#`

It prints: 123454321

You type: `PRINT 111111 * 111111#`

It prints: 12345654321

If you keep going, you will soon see a double precision floating point number.

You type:

`PRINT 111111111 * 111111111#`

It prints:

1.234567898765432D+16

A double precision floating point number consists of a mantissa and an exponent separated by the letter "D." The mantissa can have up to 16 digits; the exponent is an integer in the range -308 to $+308$. A double precision number occupies 8 bytes of memory space.

REMEMBER: The symbols used to designate number types:

% for short integer
& for long integer
! for single precision
for double precision



Trouble!

PEOPLE USE **decimal numbers** to represent numbers and do arithmetic. A \$7 solar-powered calculator does decimal arithmetic correctly. Let's do some simple arithmetic.

Our local sales tax is 6%. To compute the sales tax, multiply the amount of the sale by .06.

Amount of sale: 100

Sales tax: $100 \times .06 = 6$

Amount of sale: 10000

Sales tax: $10000 \times .06 = 600$

Amount of sale: 1000000

Sales tax: $1000000 \times .06 = 60000$

Now let's do it in QuickBASIC. Enter and run the following program.

```
CLS
TaxRate = .06
PRINT 100 * TaxRate
PRINT 10000 * TaxRate
PRINT 1000000 * TaxRate
```

We ran the program and got the following answers.

```
6
600
59999.99865889549
```

Oops! 6 and 600 are OK, but how about the 3rd answer? It is supposed to be *exactly* 60000. Well, it's only off a little, less than a penny, but that's bothersome.



Ha! I'd never make a mistake like that.

The Binary Blues

ALAS, QUICKBASIC uses **binary numbers** to represent numbers and do arithmetic, thus introducing tiny errors in some numbers. The decimal numbers you type into the computer are converted to binary numbers. That's OK for integers, but commonly used decimal fractions, such as .06, cannot be exactly represented in binary.

QuickBASIC's binary numbers are close enough to the real thing for most purposes, but keep in mind that a tiny error can exist. When in doubt, use double precision numbers, which are more *precise*. That is, the error in converting from decimal to binary is less for double precision numbers than for single precision numbers. Run the following tiny program, which uses double precision numbers (.06# and 1000000#) and a double precision variable (TaxRate#).

```
CLS
TaxRate# = .06#
PRINT 1000000# * TaxRate#
```

We ran it and got:

```
60000
```

Inside the computer, this apparently correct result is still not quite exact, but is real close. Go ahead and try some even larger amounts of sale. Go for a trillion or even more.

REMEMBER: QuickBASIC has no problem with integers and represents them quite well. But non-integers might have tiny errors. To minimize the binary blues, you can use double precision numbers.



Experiment!

YOU CAN learn more about QuickBASIC by experimenting. Try things and see what happens. For example, try this tiny program.

```
CLS
x = 1 / 3
PRINT x
PRINT 10 * x
PRINT 100 * x
PRINT 1000 * x
```

After running the program, change the second line from the top to the following:

```
x# = 1 / 3#
```

Also change x to x# in all the PRINT statements, then run the program again. Do the results look better?

Here is another one. Run this program.

```
CLS
x = .01
PRINT 10 * x
PRINT 100 * x
PRINT 1000 * x
PRINT 10000 * x
PRINT 100000 * x
PRINT 1000000 * x
```

Now change .01 to double precision as .01# and change x to x# everywhere in the program. Run the program again. Better?

Suppose you change x to x#, everywhere, but forget to change .01 to .01#? Try this program.

```
CLS
x# = .01
PRINT 10 * x#
PRINT 100 * x#
PRINT 1000 * x#
PRINT 10000 * x#
PRINT 100000 * x#
PRINT 1000000 * x#
```



Little Errors Can Accumulate

THE ERROR, if any, in a double precision number is quite small. However, if you add a bunch of numbers with tiny errors, they accumulate. The next program assigns the double precision value .1# to the double precision variable x#, then computes the double precision value of sum# by adding x# ten thousand times to the previous value of sum#. This program also prints the value of TIMER before and after it does the work so you can see how long it took.

```
CLS
x# = .1#
sum# = 0
PRINT TIMER
FOR k = 1 TO 10000
    sum# = sum# + x#
NEXT k
PRINT TIMER, sum#
```

Here's a run. You can see that it took about 14 seconds. You can also see the accumulated error on the right end of the value of sum#.

```
22734.65
22748.32      1000.000000000159
```

What might happen if you change x# to a single precision variable in both places where it occurs in the program?

Try it and find out.



TOTAL 1

The Shareware Book
Using PC-Write, PC-File, PC-Talk
By Emil Flock, et al
(Osborne/McGraw-Hill, 688pp)

Covers the most popular "free" programs: PC-Write, a word processor; PC-File, a database manager; and PC-Talk, a telecommunications program. These programs are available thru user groups or bulletin board services in return for a nominal registration fee. The book has all the details on how you can obtain these program disks.

OMH-881251 The Shareware Book \$14.95

Ingenious Mathematical Problems and Methods

by Louis A. Graham
(Dover, 237pp)

Sophisticated material from Graham's *Dial*, applied and pure; stresses solution methods. Logic, number theory, networks, inversions, etc.

DOV-20545-2 Ingenious Mathematical Problems and Methods \$4.95

The Surprise Attack in Mathematical Problems

by Louis A. Graham

Second volume from *Dial*. Difficult, sophisticated, challenging situations, stressing optimal approaches. Applied and pure.

DOV-21846-5 The Surprise Attack in Mathematical Problems \$4.50

Mathematical Brain Benders

by S. Barr
(Dover, 224pp)

"Another collection of devilish problems, everyone original."—Martin Gardner. Over 100 fresh paradoxes, word and number games with wit, humor. Answers.

DOV-24260-9 Mathematical Brain Benders \$4.95

Perplexing Puzzles and Tantalizing Teasers

by Martin Gardner
(Dover, 256pp)

93 riddles, mazes, illusions, tricky questions, word and picture puzzles, other entertainments for youngsters. Many hilarious drawings. Solutions.

DOV-25637-5 Perplexing Puzzles and Tantalizing Teasers \$3.95

Martin Gardner's Mathematical Puzzles

by Martin Gardner
(Dover, 112pp)

Entertaining collection includes stimulating puzzles involving arithmetic, money, speed, plane and solid geometry, topology, more. Solutions.

DOV-25211-6 Martin Gardner's Mathematical Puzzles \$2.95

Mathematical Puzzles of Sam Loyd

edited by Martin Gardner
(Dover, 167pp)

Bizarre, original, whimsical puzzles by America's greatest puzzler. Elementary math.

DOV-20498-7 Mathematical Puzzles of Sam Loyd \$3.95

More Mathematical Puzzles of Sam Loyd

edited by Martin Gardner
(Dover, 177pp)

166 more problems from *Cyclopedia*. Arithmetic, algebra, speed and distance problems, game theory, more.

DOV-20709-9 More Mathematical Puzzles of Sam Loyd \$3.95

Mathematics, Magic and Mystery

by Martin Gardner
(Dover, 176pp)

Math behind card tricks, stage mind reading, coin and match tricks, etc. Plus more than 400 tricks, guaranteed to work.

DOV-20335-2 Mathematics, Magic and Mystery \$3.95

A Short Account of the History of Mathematics

by W.W. Rouse Ball
(Dover, 522pp)

One of the clearest, most authoritative surveys from the Egyptians and Phoenicians thru 19th-century figures such as Grassman, Galois, Riemann.

DOV-20630-0 A Short Account of the History of Mathematics \$9.95

Different Worlds Publications

2814 - 19th Street
San Francisco, CA 94110

Address Correction Requested

BULK RATE
U.S. Postage
PAID
San Francisco, CA
Permit No. 11798



MICHAEL K ERICKSON # 14
COMMUNICATIONS ASSOCIATES
BOX 250
MONTE RIO, CA 95462-0250



No. 8, September 1989

\$3.00

*For beginning
programmers with no prior
programming experience*

CONTENTS

- 3 Teach Yourself BASIC
- 9 Browsing BASIC
- 23 Teach Yourself
QuickBASIC

PUBLISHER'S STATEMENT: *The BASIC Teacher* is published monthly by Different Worlds Publications, 2814 - 19th Street, San Francisco, CA 94110. Contents copyright © 1989 by Different Worlds Publications. All rights reserved. Contents may be copied and distributed freely. Address all correspondences to *The BASIC Teacher*, 2814 - 19th Street, San Francisco, CA 94110.

SUBSCRIPTION INFO: A 12-issue sub in the U.S. and Canada is \$36. Overseas subs are \$42 by surface mail, \$54 by air.

PRINTED IN THE U.S.A.



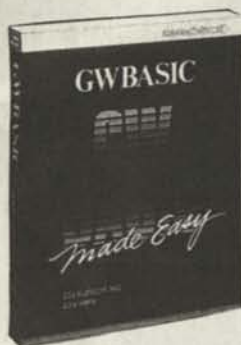
BASIC

HAPPY COMPUTING!

Tadashi Ehara, publisher
Bob Albrecht, editor
Don Inman, editor

BASIC Books

Use Order Form
on page 23.



GW-BASIC Made Easy by Bob Albrecht and Don Inman. If you've always wanted to learn BASIC programming skills on your personal computer, but weren't sure where to start, here's the book you need. Bob Albrecht and Don Inman have written *GW-BASIC Made Easy* for the millions who already have GW-BASIC (or its nearly-identical cousin, BASICA) installed on their computers. *GW-BASIC Made Easy* is designed to teach you how to satisfy your curiosity about programming, while establishing excellent programming fundamentals for your future ventures into QuickBASIC or Turbo BASIC. In particular, Albrecht and Inman emphasize foundation skills needed to develop a "clean" style of programming, through detailed explanations and numerous sample programs. (OMH-881473 \$19.95)



QuickBASIC Made Easy by Bob Albrecht, Wenden Wiegand, and Dean Brown (Osborne/McGraw-Hill, 350pp). Learn how to program with Microsoft's QuickBASIC using Osborne/McGraw-Hill's popular "Made Easy" format. For beginning programmers or those experienced in other languages, *QuickBASIC Made Easy* is a step-by-step introduction to reading and writing programs with versions of QuickBASIC through version 4.5. You'll find information on creating files, building a toolkit, editing, and debugging. *QuickBASIC Made Easy* will keep you up-to-date with the latest changes in Microsoft's outstanding compiler. (OMH-881421 \$19.95)

BASIC Celebrates 25th Anniversary

BASIC, the computer language, turns 25 this year. Dartmouth professors John G. Kemeny and Thomas E. Kurtz developed the original BASIC language as an instructional tool for training novice programmers.

Microsoft, the developers of the most widely-used PC versions of BASIC, claims it has sold half a million copies of QuickBASIC worldwide and 200,000 alone in 1989. It is the best-selling Microsoft language in Europe and among the best sellers in the U.S. and Japan. Microsoft estimates its retail BASIC business will grow by 50% in 1990. BASIC is also the most widely taught language in secondary and



higher institutions of education. Microsoft has targeted the business programming market as a potential market for BASIC expansion.

Addressing BASIC developers in Seattle, Bill Gates, Microsoft chairman and CEO, as well as the codeveloper of the first PC version of BASIC in 1975, reconfirmed his company's commitment to the venerable language, saying it "is a pivotal member of our language family and an important element in our future applications strategy."

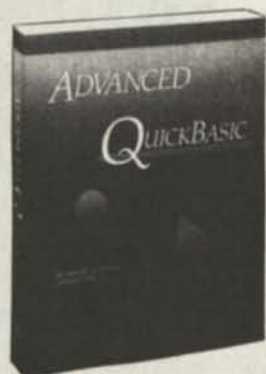
Microsoft has ambitious plans for BASIC, the language upon which the company was founded. More on this next issue.

Using QuickBASIC 4.5 Second Edition by Don Inman and Bob Albrecht (Osborne/McGraw-Hill, 500pp). Here's an excellent programming guide to Microsoft's newest version of QuickBASIC by the authors of *The BASIC Teacher*. The book approaches QuickBASIC's programming environment in three stages so beginning and experienced BASIC programmers can find the appropriate level of instruction. You'll learn about subprograms, libraries, meta-commands, dynamic debugging, and the important advantage of QuickBASIC's speedy graphics. (OMH-881514 \$22.95)

Still Available

Using QuickBASIC 4.0 (OMH-881274 \$19.95)

QuickBASIC: The Complete Reference by Steven Nameroff (Osborne/McGraw-Hill, 700pp). A comprehensive guide for users of all levels of programming ability from novices to pros. Nameroff has divided the book into sections to help you easily locate the information you need. *QuickBASIC: The Complete Reference* begins with a quick introduction to BASIC programming, followed by a complete command reference section and a discussion of QuickBASIC functions, procedures, files, and graphics. Advanced techniques and applications are grouped together in the last section of the book for the professional QuickBASIC programmer. (OMH-881362-X \$26.95)



Advanced QuickBASIC by Don Inman, Bob Albrecht, and Arne Jamtgaard (Osborne/McGraw-Hill, 500pp). As an experienced programmer you'll learn how to write professional-level programs using the advanced techniques found in the 4.5 version of QuickBASIC. *Advanced QuickBASIC*, written by three highly regarded professional programmers, provides you with the tools to produce sophisticated programs. Graphics, construction kits, and data bases are some of the topics discussed. Find information on Quick libraries, macros, keyed, sequential, and unsequential files and more. *Advanced QuickBASIC* is packed with applications, examples, and models that will soon have you programming like an expert. (OMH-881361 \$21.95)



Introduction

BASIC HAS a small **vocabulary** and a simple **syntax** (grammar). We have already discussed some of the special words that Microsoft BASIC understands. They are called **keywords** or **reserved words**. Here are the keywords introduced and described previously:

BEEP	GOTO	LOAD	PRINT	SOUND
CLS	INPUT	LOCATE	RANDOMIZE	SYSTEM
COLOR	INPUT\$	NEW	REM	TIMES
DATE\$	INT	NEXT	RND	TIMER
FILE	KEY	OFF	RUN	WIDTH
FOR	LIST	ON	SAVE	

We always show keywords in all upper-case letters and variables in lower-case or a MiXtuRe of upper- and lower-case letters. We think this makes programs easier to read and understand. However, when you LIST a program, unfortunately, variables will probably appear in all upper-case letters.

Handbook of BASIC

by David L. Schneider

IF YOU don't have a GW-BASIC (or BASICA) reference manual, or can't understand the one you have—get this one. As a reference guide, it's downright excellent, thoroughly covering keywords from A to Z—er, well—from **ABS** to **WRITE#**. At 750+ pages, this book is more than twice the size of the reference manual that comes with GW-BASIC and perhaps seven times more readable. It's fun to open it to a random page and just browse. It even has a long appendix on QuickBASIC and Turbo BASIC. You can order it thru **The BASIC Teacher's** Special Reader Services (see page 23).



David Ahl's BASIC Computer Adventures

JOURNEY THRU time and space; experience travel in other cultures, other times, from distant past to a future perhaps within the lifetime of a young child. These are **educational adventures** in the form of a book with listings in Microsoft BASIC and a disk containing the programs. If you are a teacher or parent with no previous experience in adventure games—loved by millions of kids—this is the way to learn something about them. Book: \$9.95 plus \$1.65 postage. If you have a Tandy 1000, IBM PC, or compatible computer, buy the disk and go a'venturing! Disk: \$20.00 plus \$1.00 postage. Learn some history and, while doing so, learn why kids love adventure games. From: David H. Ahl, 13 Indian Head Road, Morristown, NJ 07960.

Sales Tax Program

OUR LOCAL sales tax is 6%, this year. Unfortunately, it seems to have a propensity to grow. Program **TYB0801**, shown at the bottom of this page, computes the sales tax for a sales amount you enter from the keyboard. It then prints the sales amount, the amount of sales tax, and the total amount of the sale, including tax. Here is a sample run.

Amount of Sale? 1

Amount of sale is 1
Amount of sales tax is .06
Total amount is 1.06

Amount of Sale? 123.45

Amount of sale is 123.45
Amount of sales tax is 7.407
Total amount is 130.857

Amount of Sale? _

Use CTRL + BREAK to stop the program.

A GOTO Loop

PROGRAM **TYB0801** has a **GOTO** loop in lines 210 thru 290. This loop repeats until you interrupt by using CTRL + BREAK (Hold down the CTRL key and press the BREAK key).

To call your attention to the **GOTO** loop, we indented the lines between the top of the loop (line 210) and the bottom of the loop (line 290). We also put the comment 'Top of loop in line 210 and the comment 'Bottom of loop in line 290.

MONTGOMERY WARD & CO.'S CATALOGUE No. 57.

Puzzles



25472 The Ferris Wheel
Puzzle consists of a neatly made box with a glass top. Inside is a perfect miniature Ferris Wheel revolved by a button underneath. The puzzle is to place a passenger (ball) into each vacant car. Can you do it?
Price each.....\$0.20
Price, per doz..... 2.10
Postage, 5c.

```
1 REM ** Sales Tax Program #1 **
2 ' The BASIC Teacher #8.  Filename: TYB0801.BAS
```

```
100 REM ** Set up **
110 CLS : KEY OFF
120 TaxRate = 6 / 100
```

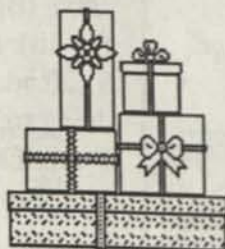
'6 percent

```
200 REM ** GOTO loop to compute and print information **
```

```
210 INPUT "Amount of Sale"; SalesAmount
220 SalesTax = SalesAmount * TaxRate
230 TotalAmount = SalesAmount + SalesTax
240 PRINT
250 PRINT "Amount of sale is "; SalesAmount
260 PRINT "Amount of sales tax is "; SalesTax
270 PRINT "Total amount is "; TotalAmount
280 PRINT : PRINT
290 GOTO 210
```

'Top of loop

'Bottom of loop



TAB

YOU CAN use the TAB function within a PRINT statement to position information where you want it on a line. Program **TYB0802** uses TAB functions in lines 250, 260, and 270 to tell the computer to begin printing at column 30 (print position 30).

TAB(30)

THE FUNCTION: **TAB(30)**

tells the computer to move the cursor to column 30. In line 250, the value of SalesAmount will be printed beginning at column 30.

```

1 REM ** Sales Tax Program #2 (with TAB function) **
2 ' The BASIC Teacher #8.  Filename: TYB0802.BAS

100 REM ** Set up **
110 CLS : KEY OFF
120 TaxRate = 6 / 100                                '6 percent

200 REM ** GOTO loop to compute and print information **

210 INPUT "Amount of sale"; SalesAmount                'Top of loop
220   SalesTax = SalesAmount * TaxRate
230   TotalAmount = SalesAmount + SalesTax
240   PRINT
250   PRINT "Amount of sale is"; TAB(30); SalesAmount
260   PRINT "Amount of sales tax is"; TAB(30); SalesTax
270   PRINT "Total amount is"; TAB(30); TotalAmount
280   PRINT : PRINT
290 GOTO 210                                           'Bottom of loop
  
```



Here is a sample run.

Amount of sale? 1		Print position 30
Amount of sale is	1	
Amount of sales tax is	.06	
Total amount is	1.06	
Amount of sale?	123.45	
Amount of sale is	123.45	
Amount of sales tax is	7.407	
Total amount is	130.857	
Amount of sale? _		

In the sample run, the numbers are printed beginning in column 30.

REMEMBER: A positive number is printed with a leading space. So the first digit of the number (or the decimal point for .06) appears in column 31. Try entering a negative sales amount. The minus sign is printed in position 30, like this:

Amount of sale? -1		Print position 30
Amount of sale is	-1	
Amount of sales tax is	-.06	
Total amount is	-1.06	

PRINT USING

WELL, THE numbers were almost lined up vertically, but not quite. We would like to see them lined up vertically at the decimal point. We would also like to see the amounts rounded to the nearest cent. Here is what we would like to see:

Amount of sale? 1	
Amount of sale is	1.00
Amount of sales tax is	0.06
Total amount is	1.06
Amount of sale?	123.45
Amount of sale is	123.45
Amount of sales tax is	7.41
Total amount is	130.86
Amount of sale? _	

The PRINT USING statement can do this. Look for it in lines 255, 265, and 275 of program TYB0803, shown here:

PRINT USING #####.##

THE FOLLOWING pair of statements prints one line on the screen:

```
PRINT "Amount of sale is "; TAB(30);
PRINT USING "#####.##"; SalesAmount
```

The first line prints the string "Amount of sale is" and then tabs over to column 30. The semicolon at the end of the statement holds the cursor at column 30, so that the information printed by the second statement will begin there.

The second statement prints the value of SalesAmount as specified by the **format string**, "#####.##." This format string tells the computer to print a number with up to 5 digits before the decimal point, then print a decimal point, then print 2 digits after the decimal point. If necessary, the digits after the point are rounded to the nearest hundredth (the nearest cent).

```
1 REM ** Sales Tax Program #3 (with PRINT USING) **
2 ' The BASIC Teacher #8.  Filename: TYB0803.BAS

100 REM ** Set up**
110 CLS : KEY OFF
120 TaxRate = 6 / 100                                '6 percent

200 REM ** GOTO loop to compute and print information **

210 INPUT "Amount of sale"; SalesAmount              'Top of loop
220   SalesTax = SalesAmount * TaxRate
230   TotalAmount = SalesAmount + SalesTax
240   PRINT
250   PRINT "Amount of sale is"; TAB(30);
255   PRINT USING "#####.##"; SalesAmount
260   PRINT "Amount of sales tax is"; TAB(30);
265   PRINT USING "#####.##"; SalesTax
270   PRINT "Total amount is"; TAB(30);
275   PRINT USING "#####.##"; TotalAmount
280   PRINT : PRINT
290 GOTO 210                                          'Bottom of loop
```


Format String

THE FORMAT string: #####.##

reserves space for printing numbers up to 99999.99, as follows:

Format string: #####.##
Largest number: 99999.99

If a number has fewer digits to the left of the decimal points, leading spaces are printed.

Format string: #####.##
Number: 123.45

This format string allows no more than 5 characters to the left of the decimal point. So the most negative number that can be printed is:

Format string: #####.##
Most negative number: -9999.99

What happens if a number is bigger than 99999.99 or less than -9999.99? As usual, find out by experimenting.

Amount of sale? 99999.99

Amount of sale is	99999.99
Amount of sales tax is	6000.00
Total amount is	%105999.99

Amount of sale? -9999.99

Amount of sale is	-9999.99
Amount of sales tax is	-600.00
Total amount is	%-10599.99

Aha! If a number doesn't fit, the computer puts a percent sign (%) in front of it.

Please note that program **TYB0803** uses single-precision variables, which can represent numbers with up to seven decimal digits. The format string used in the program (#####.##) allows up to seven digits to be printed. Quite appropriate, we think.

Variations

INSTEAD OF "#####.##"

try one of these:

"\$#####.##"

"\$\$#####.##"

"\$\$\$#,###.##"

"\$\$\$\$###, .##"

For one of the above format strings, the results are as shown below. We added a "ruler" so you can see where print position 30 is located.

Amount of sale? 12345.67

Amount of sale is	\$12,345.67
Amount of sales tax is	\$740.74
Total amount is	\$13,086.41

1234567890123456789012345678901234567890

We used the same format string in all the PRINT USING statements. Try different format strings in different PRINT USING statements and adjust the TAB function so that decimal points line up.

EXPERIMENT!



A Sales Tax Program for Big Spenders

THE THREE programs shown previously all use **single-precision** numbers and variables. The format string we used (#####.##) is about right for single-precision work. However, it won't do for big spenders. Instead, use program **TYB0804**, which uses **double-precision** numbers and variables.

Program **TYB0804** assigns a format string as the value of the string variable `format$`. This variable is then used in all the **PRINT USING** statements.

```
Format$ = "$$####,###,###,###.##"
```

```
PRINT USING format$; SalesAmount#
PRINT USING format$; SalesTax#
PRINT USING format$; TotalAmount#
```

REMEMBER: You use a number sign (#) to designate a variable as a **double-precision** variable. `SalesAmount#`, `SalesTax#`, and `TotalAmount#` are double-precision variables.

\$\$####,###,###,###.##

THE **FORMAT** string:

\$\$####,###,###,###.##

tells the computer to print a number as follows:

- The double dollar sign (\$\$) causes a dollar sign to be printed to the left of the number.
- The 12 number signs (#) separated by commas allow up to 12 digits of the number to be printed, with commas inserted every three digits. If fewer than 12 digits are printed, spaces are printed instead, in this case to the left of the dollar sign.
- The decimal point causes a decimal point to be printed in the number.
- The two number signs to the right of the decimal point cause two digits to be printed to the right of the decimal point. If necessary, the number being printed is rounded to two places.

Your turn. Try this program with big sales, such as 1 million or 1 billion or more.

```
1 REM ** Sales Tax Program #4 (double precision) **
2 ' The BASIC Teacher #8.  Filename: TYB0804.BAS

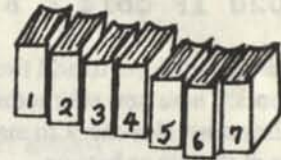
100 REM ** Set up **
110 CLS : KEY OFF
120 TaxRate# = 6 / 100#
130 format$ = "$$####,###,###,###.##"           'Double precision
                                                'Format string

200 REM ** GOTO loop to compute and print information **

210 INPUT "Amount of sale"; SalesAmount#       'Top of loop
220 SalesTax# = SalesAmount# * TaxRate#
230 TotalAmount# = SalesAmount# + SalesTax#
240 PRINT
250 PRINT "Amount of sale is "; TAB(25);
255 PRINT USING format$; SalesAmount#
260 PRINT "Amount of sales tax is"; TAB(25);
265 PRINT USING format$; SalesTax#
270 PRINT "Total amount is "; TAB(25);
275 PRINT USING format$; TotalAmount#
280 PRINT : PRINT
290 GOTO 210                                     'Bottom of loop
```


THE FOLLOWING topics were covered in previous installments of "Browsing BASIC":

- #1: The **KEY** command and ALT-key shortcuts
- #2: The **VIEW PRINT** statement and pull-down windows
- #3: The **COPY CON** command to write a **DRIVER.SYS** file and disk labels
- #4: Text and Graphic position relationships
- #5: Enclosing text in ellipses
- #6: Using DOS text files
- #7: **PUT** and **GET** and coloring text



In "Browsing BASIC #7," we used **GET** and **PUT** statements to move a box from one place to another. You should be aware that the keywords, **GET** and **PUT**, are used in two ways. We have been using the graphics form of **GET** and **PUT**. These keywords can also be used as file input and output statements.

MOVING THE CURSOR LEFT AND RIGHT

WE USED two subroutines in Program 7-4 (last month) to move our cursor shape left and right. Since we only had one line of text, the subroutines (shown here again) were simple.

```
1000 REM ** Subroutine: Move Right **
1010 col% = col% + 8
1020 IF col% > 256 THEN col% = 8
1030 PUT (col%, lyne%), Cursor%, XOR      ' turn on
1040 RETURN
```



```
2000 REM ** Subroutine: Move Left **
2010 col% = col% - 8
2020 IF col% < 8 THEN col% = 256
2030 PUT (col%, lyne%), Cursor%, XOR      ' turn on
2040 RETURN
```



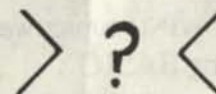
Now we want to modify our Move Cursor and Change Text Colors program so that the cursor can be moved up and down as well as left and right. When the cursor reaches the end of a line, we want it to move down a line as well as moving back to the extreme left column. When the cursor reaches the begin-

ning of a line, we want it to move up a line as well as to the extreme right. This will allow a smooth cursor movement throughout the entire screen.

To accomplish these two feats, we will leave the Move Right and Move Left subroutines largely unchanged. We'll modify only one line in each of the subroutines.

```
1020 IF col% > 312 THEN GOSUB 4010
```

```
2020 IF col% < 8 THEN GOSUB 5010
```



Of course, we'll have to add two new subroutines to make the necessary adjustments. These new subroutines must not only move the cursor to a new line, they must also check to make sure the cursor has not moved off the screen at the top or bottom.

```
4000 REM ** Subroutine: End Adjustment **
4010 IF col% > 312 THEN col% = 8: lyne% = lyne% + 8
4020 IF lyne% > 192 THEN lyne% = 8
4030 RETURN
```

```
5000 REM ** Subroutine: Start Adjustment **
5010 IF col% < 8 THEN col% = 312: lyne% = lyne% - 8
5020 IF lyne% < 8 THEN lyne% = 192
5030 RETURN
```



Notice the repetition of the IF ... THEN statements in the subroutines and the lines that call the subroutines. This was done so that these two subroutines could also be used when the cursor moves off the top or the bottom of the screen. You will see this more clearly later.

MOVING THE CURSOR UP AND DOWN

NOW WE are ready to add up and down movements for the cursor. We first move the three closing lines of the main program (Put Cursor and Make Changes block) to make room for two IF statements that detect the up and down arrow keys.

```
340 IF ky$ = CHR$(0) + "P" THEN GOSUB 6010
350 IF ky$ = CHR$(0) + "H" THEN GOSUB 7010
360 WEND
370 ky$ = INPUT$(1)
380 END
```



We must also modify line 300 to check for illegal key presses. Legal key presses are now right arrow, left arrow, down arrow, up arrow, and a carriage return. Line 300 now becomes monstrously large.

```
300 IF KY$ <> CHR$(0)+"M" AND KY$ <> CHR$(0)+"K" AND KY$ <>
CHR$(13) AND KY$ <> CHR$(0)+"P" AND KY$ <> CHR$(0)+"H" THEN
GOTO 250
```


Next, we must write the Move Down and Move Up subroutines. Not only must we update the cursor position, but a check must be made to see if the cursor has moved off the screen at the top or bottom. If it has, the subroutines of block 4000 or 5000 are used as they were when the cursor moved off the screen to the left or to the right.

The subroutine of block 4000 is used to check and adjust (if necessary) the cursor position for a cursor movement off the screen at the right and at the bottom. The subroutine of block 5000 checks and adjusts cursor movement off the screen at the left and at the top.

Here are the Move Down and Move Up subroutines.

```
6000 REM ** Subroutine: Move Down **
6010 lyne% = lyne% + 8
6020 IF lyne% > 192 THEN GOSUB 4010
6030 PUT (col%, lyne%), Kursor%, XOR
6040 RETURN
```

```
7000 REM ** Subroutine: Move UP **
7010 lyne% = lyne% - 8
7020 IF lyne% < 8 THEN GOSUB 5010
7030 PUT (col%, lyne%), Kursor%, XOR
7040 RETURN
```

off at bottom



off at top



PUTTING IT ALL TOGETHER

ONLY ONE more modification is needed to complete the new program. We must add more text to the screen so that there will be more than one line to scan. Lines 150-190 contain the text.

```
150 CLS: LOCATE 2,2: PRINT "Press an arrow key to move the cursor"
160 PRINT " from letter to letter."
170 LOCATE 5, 2: PRINT "Press a number: 1, 2, or 3 to change"
180 PRINT " the color of the character."
190 LOCATE 8, 2: PRINT "Press the ENTER key to quit."
```

Here is Program 8-1, Scan a Screenfull and Color.

```
1 REM ** Scan a Screenfull and Color **
2 ' Program 8-1 8/28/88
3 ' Microsoft GW-BASIC File: PRO0801.BAS

100 REM ** Initialize **
110 SCREEN 1: CLS: KEY OFF
120 DIM Kursor%(10) ' dimension array
130 LINE (1, 1)-(8, 8), 1, B ' draw 8 x 8 box
140 GET (1, 1)-(8, 8), Kursor% ' save in array named Kursor
150 CLS: LOCATE 2,2: PRINT "Press an arrow key to move the cursor"
160 PRINT " from letter to letter."
170 LOCATE 5, 2: PRINT "Press a number: 1, 2, or 3 to change"
180 PRINT " the color of the character."
190 LOCATE 8, 2: PRINT "Press the ENTER key to quit."
```



```

200 REM ** Put Cursor and Make Changes **
210 column% = 2: row% = 2
220 col% = column% * 8 - 8: lyne% = row% * 8 - 8
230 PUT (col%, lyne%), Cursor%      ' Cursor on
240 WHILE KY$ <> CHR$(13)
250   ky$ = ""
260   WHILE ky$ = ""
270     ky$ = INKEY$
280   WEND
290   IF ky$ > "0" AND ky$ <= "3" THEN GOSUB 3010
300   IF ky$ <> CHR$(0)+"M" AND ky$ <> CHR$(0)+"K" AND ky$ <>
CHR$(13) AND ky$ <> CHR$(0)+"P" AND ky$ <> CHR$(0)+"H" THEN
GOTO 250
310   PUT (col%, lyne%), Cursor%, XOR      'Cursor off
320   IF ky$ = CHR$(0) + "M" THEN GOSUB 1010
330   IF ky$ = CHR$(0) + "K" THEN GOSUB 2010
340   IF ky$ = CHR$(0) + "P" THEN GOSUB 6010
350   IF ky$ = CHR$(0) + "H" THEN GOSUB 7010
360 WEND
370 ky$ = INPUT$(1)
380 END

1000 REM ** Subroutine: Move Right **
1010 col% = col% + 8
1020 IF col% > 312 THEN GOSUB 4010
1030 PUT (col%, lyne%), Cursor%, XOR      ' turn on
1040 RETURN

2000 REM ** Subroutine: Move Left **
2010 col% = col% - 8
2020 IF col% < 8 THEN GOSUB 5010
2030 PUT (col%, lyne%), Cursor%, XOR      ' turn on
2040 RETURN

3000 REM ** Subroutine: Color **
3010 PUT (col%, lyne%), Cursor%, XOR      ' turn off
3020 kolor% = VAL(ky$)
3030 FOR pnt% = col% TO col% + 8
3040   FOR tier% = lyne% TO lyne% + 7
3050     IF POINT(pnt%, tier%) <> 0 THEN PSET(pnt%, tier%), kolor%
3060   NEXT tier%
3070 NEXT pnt%
3080 PUT (col%, lyne%), Cursor%, XOR      ' turn on
3090 RETURN

4000 REM ** Subroutine: End Adjustment **
4010 IF col% > 312 THEN col% = 8: lyne% = lyne% + 8
4020 IF lyne% > 192 THEN lyne% = 8
4030 RETURN

5000 REM ** Subroutine: Start Adjustment **
5010 IF col% < 8 THEN col% = 312: lyne% = lyne% - 8
5020 IF lyne% < 8 THEN lyne% = 192
5030 RETURN

```



```

6000 REM ** Subroutine: Move Down **
6010 lyne% = lyne% + 8
6020 IF lyne% > 192 THEN GOSUB 4010
6030 PUT (col%, lyne%), Cursor%, XOR
6040 RETURN

```

```

7000 REM ** Subroutine: Move UP **
7010 lyne% = lyne% - 8
7020 IF lyne% < 8 THEN GOSUB 5010
7030 PUT (col%, lyne%), Cursor%, XOR
7040 RETURN

```

RUNNING PROGRAM 8-1

WHEN YOU start the program, the text is printed and the cursor is placed over the first character in the first line. The text indicates the actions you can take.

Press an arrow key to move the cursor from letter to letter.

Press a number: 1, 2, or 3 to change the color of the character.

Press the ENTER key to quit.

Move the cursor around in the text to position it where you want. The next figure shows the cursor moved to the first letter of the word **arrow**.

Press an arrow key to move the cursor from letter to letter.

Press a number: 1, 2, or 3 to change the color of the character.

Press the ENTER key to quit.

To color a character, press one of the numbers 1, 2, or 3. Press the right arrow key to move to the next character. Press a number to color it. Repeat this process until an entire word, or group of words, is colored. The next figure shows the word **arrow** changed to color 1 (cyan). The cursor is on the last letter of the word.



Press an arrow key to move the cursor from letter to letter.

Press a number: 1, 2, or 3 to change the color of the character.

Press the ENTER key to quit.

Move on and color other letters, words, or phrases if you want. When you have the text colored just the way you want, press the ENTER key. The cursor is erased, and only the finished text shows. The next figure shows a completed screen with most characters displayed in white. The exceptions are:

<i>cyan</i>	<i>magenta</i>
arrow	cursor
number	color
1	2
ENTER	quit

Press an arrow key to move the cursor from letter to letter.

Press a number: 1, 2, or 3 to change the color of the character.

Press the ENTER key to quit.

QuickBASIC VERSION

THE QUICKBASIC version of Scan a Screenfull and Color checks and updates the cursor and colors text in one large IF ... END IF block instead of using the many subroutines of the GW-BASIC version. The block IF ... END IF structure is not available in GW-BASIC. Another difference is the use of the DO ... LOOP structure in the QuickBASIC program in place of WHILE ... WEND. The QuickBASIC IF ... END IF block uses the following form.


```

IF ky$ = CHR$(0) + "M" THEN
.
.
ELSEIF ky$ = CHR$(0) + "K" THEN
.
.
ELSEIF ky$ = CHR$(0) + "P" THEN
.
.
ELSEIF ky$ = CHR$(0) + "H" THEN
.
.
ELSEIF ky$ > CHR$(0) AND ky$ <= "3" THEN
.
.
ELSE PUT (col%, lyne%), Cursor%, XOR
END IF

```

Each group of three vertical ellipses contains a block of QuickBASIC code equivalent to a subroutine in the GW-BASIC version.

The QuickBASIC version uses subprograms in place of the Start Adjustment and End Adjustment subroutines of the GW-BASIC version. Subroutines can be used in QuickBASIC, but the subprogram procedure (SUB...END SUB) is more versatile.

Here is the QuickBASIC program.

```

DECLARE SUB Adjust2 ()
DECLARE SUB Adjust1 ()

REM ** Scan a Screenfull and Color - QuickBASIC **
' Program 8-2 8/28/88
' Microsoft QuickBASIC File: PRO0802.BAS

REM ** Initialize **
DIM SHARED col AS INTEGER, lyne AS INTEGER
DIM Cursor(1 TO 11) AS INTEGER           'dimension array
SCREEN 1: CLS
LINE (1, 1)-(8, 8), 1, B                 'draw 8 x 8 box
GET (1, 1)-(8, 8), Cursor%               'save in array
CLS : LOCATE 2, 2: PRINT "Press an arrow key to move the cursor"
PRINT " from letter to letter."
LOCATE 5, 2: PRINT "Press a number: 1, 2, or 3 to change"
PRINT " the color of the character."
LOCATE 8, 2: PRINT "Press the ENTER key to quit."

REM ** Put Cursor and Make Changes **
column% = 2: row% = 2
col% = column% * 8 - 8: lyne% = row% * 8 - 8
PUT (col%, lyne%), Cursor%, XOR          'Cursor on
DO
DO
ky$ = INKEY$
LOOP WHILE ky$ = ""
PUT (col%, lyne%), Cursor%, XOR          'Cursor off
IF ky$ = CHR$(0) + "M" THEN

```



```

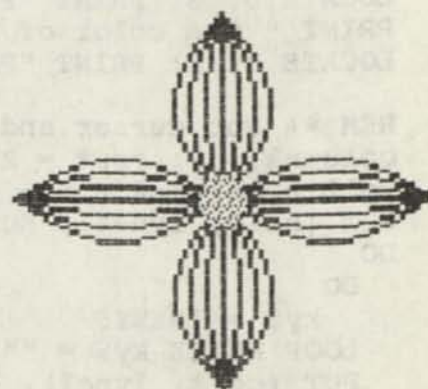
col% = col% + 8
IF col% > 312 THEN CALL Adjust1
PUT (col%, lyne%), Cursor%, XOR
ELSEIF ky$ = CHR$(0) + "K" THEN
col% = col% - 8
IF col% < 8 THEN CALL Adjust2
PUT (col%, lyne%), Cursor%, XOR
ELSEIF ky$ = CHR$(0) + "P" THEN
lyne% = lyne% + 8
IF lyne% > 192 THEN CALL Adjust1
PUT (col%, lyne%), Cursor%, XOR
ELSEIF ky$ = CHR$(0) + "H" THEN
lyne% = lyne% - 8
IF lyne% < 8 THEN CALL Adjust2
PUT (col%, lyne%), Cursor%, XOR
ELSEIF ky$ > "0" AND ky$ <= "3" THEN
kolor% = VAL(ky$)
FOR pnt% = col% TO col% + 8
FOR tier% = lyne% TO lyne% + 7
IF POINT(pnt%, tier%) <> 0 THEN
PSET (pnt%, tier%), kolor%
END IF
NEXT tier%
NEXT pnt%
PUT (col%, lyne%), Cursor%, XOR
ELSE PUT (col%, lyne%), Cursor%, XOR
END IF
LOOP WHILE ky$ <> CHR$(13)
PUT (col%, lyne%), Cursor%, XOR
END

SUB Adjust1
IF col% > 312 THEN
col% = 8
lyne% = lyne% + 8
END IF
IF lyne% > 192 THEN lyne% = 8
END SUB

SUB Adjust2
IF col% < 8 THEN
col% = 312
lyne% = lyne% - 8
END IF
IF lyne% < 8 THEN lyne% = 192
END SUB

```

'move Cursor right
'Cursor on
'move Cursor left
'Cursor on
'move Cursor down
'Cursor on
'move Cursor up
'Cursor on
'color text
'Cursor on
'Cursor on



The QuickBASIC program "paints" characters very fast. You can see the individual points of a character being set in the GW-BASIC program. The QuickBASIC program sets the points so fast that the complete character seems to be instantly placed on the screen.

That's all for now. Happy Screen Coloring!

The Family Computer Club

WE WENT to meetings of several computer clubs and found that they were no place for beginners! The people there (98.3% male—no kids) all mumbled in advanced computerese.

So we are helping start **The Family Computer Club** here in our town, Sebastopol, CA. This club is primarily for beginners, especially families who have, or plan to acquire, a Tandy 1000, IBM PC, or compatible computer.

Kids are very welcome. After all, this is the **Family Computer Club**. We'll use Tandy's **DeskMate** to help beginners begin. We are reviewing hundreds of **shareware** disks to find the best educational and home business programs. And, of course, we'll use **QuickBASIC** for many purposes.



FOR...NEXT Loops

YOU CAN use a FOR . . . NEXT loop to assign a **sequence of values** to a numeric variable. The program shown below uses a FOR . . . NEXT loop to print the numbers 1, 2, 3, and so on, up to 7.

```
CLS
FOR number = 1 TO 7
  PRINT number
NEXT number
```

Enter and run this program. You should see the numbers 1 to 7 on the Output screen.



A FOR...NEXT Loop

- BEGINS WITH a FOR statement;
- ends with a NEXT statement;
- usually has one or more statements between FOR and NEXT.

A numeric variable must follow the word FOR:

```
FOR number = 1 TO 7
    numeric variable
```

The same numeric variable follows the word NEXT:

```
NEXT number
    numeric variable
```

This numeric variable can be used in statements between FOR and NEXT:

```
PRINT number
    numeric variable
```


A Sequence of Values

A FOR statement defines a sequence of values for the numeric variable that follows the word FOR.

- The statement: FOR number = 1 TO 7 defines the following sequence of values for the variable number:

1, 2, 3, 4, 5, 6, 7

When you ran the program on the preceding page, you saw these numbers printed on the screen by the PRINT statement.

In the following program, the FOR statement defines a sequence of numbers from 0 to 5 as the value of number.

```
CLS
FOR number = 0 TO 5
  PRINT number;
NEXT number
```

Note the semicolon (;) at the end of the PRINT statement. This causes the numbers to be printed close together on the same line:

```
0 1 2 3 4 5
```

You can use any numeric variable after the word FOR. Also use the same variable after the word NEXT.

In the following program, the FOR statement defines a sequence of numbers from 10 to 13 as the values of the variable k.

```
CLS
FOR k = 10 TO 13
  PRINT k;
NEXT k
```

Here is a run:

```
10 11 12 13
```

A Colorful FOR...NEXT Loop

THE "NORMAL" screen colors are white letters on a black screen. White is the **foreground** color; black is the **background** color. You can use the COLOR statement to tell the computer to print in any of 16 colors, including black (COLOR 0) and the normal white (COLOR 7). Of course, if you print in black letters on a black background, you won't see anything.

The following program prints one line in each of the 15 colors from 1 to 15 and tells you the color number in which it is printed:

```
CLS
FOR kolor = 1 TO 15
  COLOR kolor
  PRINT "This is color number"; kolor
NEXT kolor
COLOR 7
```

Unfortunately, we can't show you the beautiful colors here. When you run the program, you will see the following in 15 different colors:

```
This is color number 1
This is color number 2
This is color number 3
This is color number 4
This is color number 5
This is color number 6
This is color number 7
This is color number 8
This is color number 9
This is color number 10
This is color number 11
This is color number 12
This is color number 13
This is color number 14
This is color number 15
```

Color me wonderful!



The last line of the program (COLOR 7) returns the screen to its normal foreground color. This is done after the FOR...NEXT loop has been completed.

If you would like to see blinking colors, change line 20 to the following:

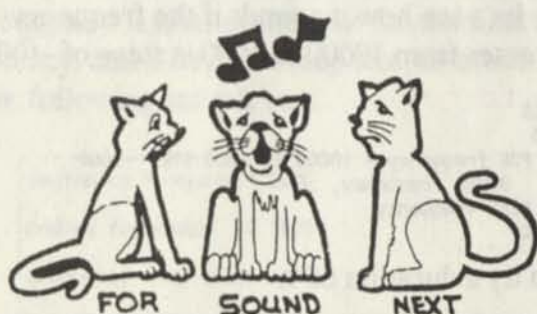
```
FOR kolor = 17 TO 31
```


SOUND Effects

HAVE YOU ever wondered how they make all those strange sounds in arcade games? Try the following program:

```
CLS
FOR frequency = 100 TO 300
  SOUND frequency, .125
NEXT frequency
```

Run this program. You will hear a sound that quickly rises from 100 Hertz (cycles per second) to 300 Hertz.



The FOR ... NEXT loop causes a sequence of very short sounds. Each sound is of duration 0.125 ticks. A tick is about 1/18 of a second. You hear 201 very short sounds with frequencies of 100 Hertz, 101 Hertz, 102 Hertz, and so on, up to 300 Hertz. This all happens in a little over a second.

With the above program, you hear one whoop, quickly rising in pitch. Now make a small change and get a program that goes whoop, whoop, whoop, ...

```
CLS
DO
  FOR frequency = 100 TO 300
    SOUND frequency, .125
  NEXT frequency
LOOP
```

When you have heard enough whoops, hold down CTRL and press BREAK to stop all the whooping.

Counting Backwards

ALL THE FOR ... NEXT loops we have used so far define **increasing sequences** of numbers:

- 1, 2, 3, 4, 5, 6, 7
- 0, 1, 2, 3, 4, 5
- 10, 11, 12, 13
- 100, 101, 102, ..., 300

You can tell the computer to count backwards by using a STEP -1 clause in the FOR statement, as shown in the following program:

```
CLS
DO
  FOR frequency = 300 TO 100 STEP -1
    SOUND frequency, .125
  NEXT frequency
LOOP
```

Run this program to hear a familiar arcade sound. This time you hear a falling pitch. The sound goes quickly from 300 Hertz to 100 Hertz in steps of -1.

- 300, 299, 298, ..., 100

Now put both programs together into a single program that makes a sound sort of like a siren. The sound goes up, down, up, down, etc.:

```
CLS
DO
  FOR frequency = 523 TO 1046
    SOUND frequency, .125
  NEXT frequency
  FOR frequency = 1046 TO 523 STEP -1
    SOUND frequency, .125
  NEXT frequency
LOOP
```



STEP

IF YOU don't use a STEP clause in a FOR statement, the variable following the word FOR will increase by one each time. For example, in the following FOR statement, the value of frequency goes from 523 to 1046 in steps of 1.

```
FOR frequency = 523 TO 1046
```

If you use a STEP clause, then you must specify the step size. Try this variation of the preceding program.

```
CLS
DO
  FOR frequency = 523 TO 1046 STEP 2
    SOUND frequency, .125
  NEXT frequency
  FOR frequency = 1046 TO 523 STEP -2
    SOUND frequency, .125
  NEXT frequency
LOOP
```

In the first FOR ... NEXT loop, the value of frequency goes from 523 to 1046 in steps of 2.

523, 525, 527, ..., 1045

Note that the final frequency (1046) is not actually attained. Only odd numbered frequencies will occur.

In the second FOR ... NEXT loop, the value of frequency goes from 1046 to 523 in steps of -2.

1046, 1044, 1042, ..., 524

The final frequency (523) is not actually attained. Only even numbered frequencies occur.



You can get many different sound effects by trying various combinations of beginning frequency, final frequency, step size, and duration. For example, try this tiny program.

```
CLS
DO
  FOR frequency = 1000 TO 10000 STEP 1000
    SOUND frequency, .5
  NEXT frequency
LOOP
```

The frequency increases from 1000 to 10000 in steps of 1000. Note that the duration is .5 ticks. How would it sound if you change the duration to 1? Try it and find out.

Now let's see how it sounds if the frequency decreases from 10000 to 1000 in steps of -1000.

```
CLS
DO
  FOR frequency = 10000 TO 1000 STEP -1000
    SOUND frequency, .5
  NEXT frequency
LOOP
```

Also try a duration of 1.

Your turn. You pick the beginning frequency, final frequency, and step size.

FOR frequency = _____ TO _____ STEP _____

Also choose the duration for each sound.

SOUND frequency, _____



Sound Effects Experimenter

YOU CAN use Program **TYQB0806**, Sound Effects Experimenter, to experiment with FOR...NEXT loops and find sound effects to your liking. We saved this program under the filename **TYQB0806**, which means "Teach Yourself QuickBASIC (TYQB), The BASIC Teacher #8 (08), the 6th program saved (06)."

Enter and run the program. To get a falling pitch, enter a larger number for the beginning frequency, a smaller number for the final frequency, and a negative step size, as shown in the following sample run.

```
Beginning frequency? 2000
Ending frequency ? 1000
Frequency step size? -100
Duration each sound? .25
Press the ESC key to quit
```

The sound generated by the FOR...NEXT loop repeats until you press the ESC key. This stops the program. If you liked the sound, jot down the numbers on the screen. Run the program again and enter your numbers.

INKEY\$

INKEY\$ is a string function. It scans the keyboard to see if a key has been pressed. If no key has been pressed, the value of INKEY\$ becomes the empty string (""). If a key has been pressed, the value of INKEY\$ becomes a one- or two-character string. You can use this value to "detect" which key or key combination was pressed.

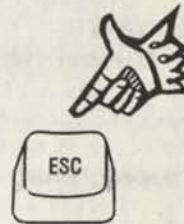
In Program **TYQB0806**, the statement:

```
DO UNTIL INKEY$ = CHR$(27)
```

tells the computer to continue the DO...LOOP UNTIL the value of INKEY\$ is the string character whose ASCII code is 27. This is the ESC key.

Try the following tiny program. It continues until you press the ESC key.

```
DO UNTIL INKEY$ = CHR$(27)
  PRINT "Press ESC to stop me"
LOOP
```



```
REM ** Sound Effects Experimenter #1 **
' The BASIC Teacher #8.  Filename: TYQB0806.BAS
```

```
CLS
```

```
LOCATE 1, 1: INPUT "Beginning frequency"; BeginFreq
LOCATE 3, 1: INPUT "Ending frequency "; FinalFreq
LOCATE 5, 1: INPUT "Frequency step size"; StepSize
LOCATE 7, 1: INPUT "Duration each sound"; duration
LOCATE 9, 1: PRINT "Press the ESC key to quit"
```

```
DO UNTIL INKEY$ = CHR$(27)
  FOR frequency = BeginFreq TO FinalFreq STEP StepSize
    SOUND frequency, duration
  NEXT frequency
LOOP
```

```
END
```



REM

IT IS good practice to put information in a program to tell people about the program. The REM (REMark) statement allows you to do this. Any text that follows REM in a program line is ignored when you run the program. You can use an apostrophe (') as an abbreviation for REM.

From now on, most of our programs will begin with a REM statement that has the name of the program. For example:

```
REM ** Sound Effects Experimenter #1 **
```

The second line will tell where the program appears and its filename, as stored on our disk.

```
' The BASIC Teacher #8. Filename: TYQB0806.BAS
```

This line tells you the program is in issue #8 of **The BASIC Teacher**. Its filename is **TYQB0806.BAS**, which means "Teach Yourself QuickBASIC #8, program #6."



Programs TYQB0801 thru TYQB0805

YOU HAVE seen Program **TYQB0806** as it is saved on our disk. We also saved five other programs, as shown here.

```
REM ** Color Numbers 1 to 16 **
' The BASIC Teacher #8. Filename: TYQB0801.BAS
```

```
CLS
FOR kolor = 1 TO 15
  COLOR kolor
  PRINT "This is color number"; kolor
NEXT kolor
COLOR 7
```

```
REM ** Whoop, Whoop, Whoop, ... **
' The BASIC Teacher #8. Filename: TYQB0802.BAS
```

```
CLS
DO
  FOR frequency = 100 TO 300
    SOUND frequency, .125
  NEXT frequency
LOOP
```

```
REM ** Whoop, Whoop - Decreasing Frequency **
' The BASIC Teacher #8. Filename: TYQB0803.BAS
```

```
CLS
DO
  FOR frequency = 300 TO 100 STEP -1
    SOUND frequency, .125
  NEXT frequency
LOOP
```

TYQB0804.BAS

TYQB0805.BAS

TYQB0801.BAS

TYQB0802.BAS

TYQB0803.BAS

```
REM ** Siren #1 **
```

```
' The BASIC Teacher #8. Filename: TYQB0804.BAS
```

```
CLS
DO
  FOR frequency = 523 TO 1046
    SOUND frequency, .125
  NEXT frequency
LOOP
```

```
REM ** Siren #2 **
```

```
' The BASIC Teacher #8. Filename: TYQB0805.BAS
```

```
CLS
DO
  FOR frequency = 523 TO 1046 STEP 2
    SOUND frequency, .125
  NEXT frequency
  FOR frequency = 1046 TO 523 STEP -2
    SOUND frequency, .125
  NEXT frequency
LOOP
```


For Power Users!

**Microsoft QuickBASIC
Version 4**

Microsoft QuickBASIC 4.5 is a complete BASIC learning system. The new interactive, on-disk tutorial, Microsoft QB Express, quickly and easily introduces you to the Microsoft QuickBASIC environment. A new step-by-step tutorial guides you through an actual application. And numerous example programs help you master BASIC programming.

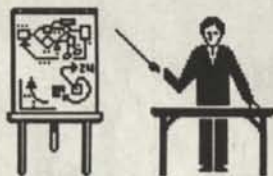
MS-04366 QuickBASIC 4.5

\$99.00

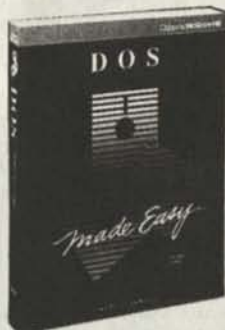
Now only \$79.00

DOS Made Easy by Herbert Schildt (*Osborne/McGraw-Hill, 385pp*). Previous computer experience is not necessary to understand this concise, well-organized introduction that's filled with short applications and exercises. The book walks you thru all the basics, beginning with an overview of a computer system's inner components and a step-by-step account of how to run DOS for the first time. (*OMH-881194 \$18.95*)

The Shareware Book: Using PC-Write, PC-File, PC-Talk by Emil Flock, et al (Osborne/McGraw-Hill, 688pp). Covers the most popular "free" programs: PC-Write, a word processor; PC-File, a database manager; and PC-Talk, a telecommunications program. These programs are available thru user groups or bulletin board services in return for a nominal registration fee. The book has all the details on how you can obtain these program disks. (OMH-881251 \$14.95)



Ideas



The BASIC Teacher
Back Issues Available

TBT-1 Issue No. 1.	\$3.00
TBT-2 Issue No. 2.	\$3.00
TBT-3 Issue No. 3.	\$3.00
TBT-4 Issue No. 4.	\$3.00
TBT-5 Issue No. 5.	\$3.00
TBT-6 Issue No. 6.	\$3.00
TBT-7 Issue No. 7.	\$3.00

Our apologies . . .

THIS ISSUE is even later than usual. For this, profuse apologies.

We are planning to expand *The BASIC Teacher* to 32 pages within the next issue or two. This development, along with our other commitments, are the

causes of the delay. Expect to see issue number 9 in November. We should be back to a regular monthly schedule thereafter.

Subscribers will get the full number of issues promised. The number next to your name on the mailing label on the back page indicates the last issue of your subscription.



**TO ORDER
PRODUCTS
DESCRIBED
IN THIS ISSUE**

Order Form

Send check or money order.
Foreign customers send
international money order
available at post offices and
banks.

Name _____
Address _____
City/State _____ Zip _____
Phone () _____

Different Worlds

2814 - 19th Street
San Francisco CA 94110

[illegible]

NEW BOOKS ON BASIC

HANDBOOK OF BASIC: THIRD EDITION

FOR THE IBM PC, XT, AT, PS/2,
AND COMPATIBLES

David I. Schneider
University of Maryland



Handbook of BASIC: Third Edition by David I. Schneider. "I can recommend [this book] without reservation to everyone, whether you're a beginner, semi-professional, or professional programmer. Anyone who takes the time to look between the covers will find that this book is indeed a very valuable reference."—Richard Aarons, *PC Magazine*. Update your programming library with this third edition of the most thorough and complete reference book on the BASIC language for PCs and compatibles ever put together. Complete and detailed explanations of statements are illuminated by over 600 examples. Each listing has the most popular form of the BASIC command, followed by subtle variations and extensions. Both a ready source and a tutorial, this is the most comprehensive BASIC reference manual available! (BRA-372582-0 \$24.95)

Advanced BASIC for the IBM PC and Compatibles: Tips and Techniques by Larry Joel Goldstein. Now that you're ready to make more sophisticated use BASIC, this book will help you develop your skills right. This book is a thorough, hands-on guide to BASIC's complex commands for creating more powerful programs. As you work through the key principles in the text, you'll literally build a program called the Bar Chart Generator. This practical case study reinforces all the valuable techniques you'll learn, to let you go on to create your own applications programs. (BRA-010307-1 \$19.95)

Hands-On QuickBASIC by Larry Joel Goldstein. You've chosen Microsoft's bestselling compiler to get programs up and running quickly. Now choose *Hands-On QuickBASIC* to get all the information you need to learn and use it quickly and efficiently. Clear, concise tutorials teach you everything you need to know. First-time programmers will find author Larry Joel Goldstein's approach both accessible and complete. Experienced users will benefit from his coverage of advanced topics. *Hands-On QuickBASIC* starts as a tutorial, then becomes a reference that you'll return to over and over again. (BRA-383480-8 \$21.95)

Math History & Fun

Martin Gardner
ENTERTAINING
MATHEMATICAL
PUZZLES



A Short Account of the History of Mathematics by W.W. Rouse Ball (Dover, 522pp). One of the clearest, most authoritative surveys from the Egyptians and Phoenicians thru 19th-century figures such as Grassman, Galois, Riemann. (DOV-20630-0 \$9.95)

Mathematics in the Time of the Pharaohs by Richard Gillings (Dover, 286pp). First book-length study—from simple commercial computations to trigonometric functions used in construction of pyramids. Fascinating, provocative. (DOV-24315-X \$6.95)

Perplexing Puzzles and Tantalizing Teasers by Martin Gardner (Dover, 256pp). 93 riddles, mazes, illusions, tricky questions, word and picture puzzles, other entertainments for youngsters. Many hilarious drawings. Solutions. (DOV-25637-5 \$3.95)

Martin Gardner's Mathematical Puzzles by Martin Gardner (Dover, 112pp). Entertaining collection includes stimulating puzzles involving arithmetic, money, speed, plane and solid geometry, topology, more. Solutions. (DOV-25211-6 \$2.95)

Mathematics, Magic and Mystery by Martin Gardner (Dover, 176pp). Math behind card tricks, stage mind reading, coin and match tricks, etc. Plus more than 400 tricks, guaranteed to work. (DOV-20335-2 \$3.95)

Ingenious Mathematical Problems and Methods by Louis A. Graham (Dover, 237pp). Sophisticated material from Graham's *Dial*, applied and pure; stresses solution methods. Logic, number theory, networks, inversions, etc. (DOV-20545-2 \$4.95)

The Surprise Attack in Mathematical Problems by Louis A. Graham. Second volume from *Dial*. Difficult, sophisticated, challenging situations, stressing optimal approaches. Applied and pure. (DOV-21846-5 \$4.50)

Mathematical Brain Benders by S. Barr (Dover, 224pp). "Another collection of devilish problems, everyone original."—Martin Gardner. Over 100 fresh paradoxes, word and number games with wit, humor. Answers. (DOV-24260-9 \$4.95)

Mathematical Puzzles of Sam Loyd edited by Martin Gardner (Dover, 167pp). Bizarre, original, whimsical puzzles by America's greatest puzzler. Elementary math. (DOV-20498-7 \$3.95)

More Mathematical Puzzles of Sam Loyd edited by Martin Gardner (Dover, 177pp). 166 more problems from *Cyclopedia*. Arithmetic, algebra, speed and distance problems, game theory, more. (DOV-20709-9 \$3.95)

The Master Book of Mathematical Recreations by Fred Schuh (Dover, 430pp). Possibly the finest book work ever prepared on mathematical puzzles, stunts, recreations. (DOV-22134-2 \$6.95)

Test Your Logic by George J. Summers (Dover, 100pp). 50 more truly new puzzles with new turns of thought, new subtleties of inference. (DOV-22877-0 \$2.50)

Codes, Ciphers and Secret Writings by Martin Gardner (Dover, 96pp). Cipher and decipher codes: transportation, polyalphabetic, famous codes, typewriter and telephone codes, much more to challenge minds. (DOV-247661-9 \$2.95)

Different Worlds Publications

2814 - 19th Street
San Francisco, CA 94110

Address Correction Requested

BULK RATE
U.S. Postage
PAID
San Francisco, CA
Permit No. 11798



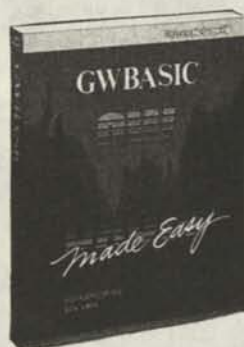
MICHAEL K ERICKSON # 14
COMMUNICATIONS ASSOCIATES
BOX 250
MONTE RIO, CA 95462-0250

HAPPY COMPUTING!

Tadashi Ehara, publisher
Bob Albrecht, editor
Don Inman, editor

BASIC Books

Use Order Form
on page 23.



GW-BASIC Made Easy by Bob Albrecht and Don Inman. If you've always wanted to learn BASIC programming skills on your personal computer, but weren't sure where to start, here's the book you need. Bob Albrecht and Don Inman have written *GW-BASIC Made Easy* for the millions who already have GW-BASIC (or its nearly-identical cousin, BASICA) installed on their computers. *GW-BASIC Made Easy* is designed to teach you how to satisfy your curiosity about programming, while establishing excellent programming fundamentals for your future ventures into QuickBASIC or Turbo BASIC. In particular, Albrecht and Inman emphasize foundation skills needed to develop a "clean" style of programming, through detailed explanations and numerous sample programs. (OMH-881473 \$19.95)



QuickBASIC Made Easy by Bob Albrecht, Wenden Wiegand, and Dean Brown (Osborne/McGraw-Hill, 350pp). Learn how to program with Microsoft's QuickBASIC using Osborne/McGraw-Hill's popular "Made Easy" format. For beginning programmers or those experienced in other languages, *QuickBASIC Made Easy* is a step-by-step introduction to reading and writing programs with versions of QuickBASIC through version 4.5. You'll find information on creating files, building a toolkit, editing, and debugging. *QuickBASIC Made Easy* will keep you up-to-date with the latest changes in Microsoft's outstanding compiler. (OMH-881421 \$19.95)

Your BASIC Backpack

Coming to The BASIC Teacher

"Your BASIC Backpack," the popular column in *The Computer Shopper* is coming to *The BASIC Teacher*. For over two years "Your BASIC Backpack" provided tutorials for beginners who want to learn how to use Microsoft BASIC and QuickBASIC. Starting with the next issue, "Your BASIC Backpack" will now appear in *The BASIC Teacher*, posing problems for you to think about, play with, and solve.

Copies of past issues of "Your BASIC Backpack" are available as follows:

- | | |
|------------------------|------------|
| #1 (10pp) | \$2.00 |
| #2 - 4 (14pp each) | \$2.50 ea. |
| #5 - 29 (16-24pp each) | \$3.00 ea. |

These are full-size (8.5"x11") copies, not the reduced print published in *The Computer Shopper*.



Prices include paper, toner, wear and tear on our trusty Canon copier, out time spent over a hot copy machine, shipping and handling. If you order 5 or more issues, deduct 10% from the total. For 10 or more issues, deduct 20%. The entire 29-issue set is available for \$65. Remember, there is no additional shipping and handling charges for these items.

Using QuickBASIC 4.5 Second Edition by Don Inman and Bob Albrecht (Osborne/McGraw-Hill, 500pp). Here's an excellent programming guide to Microsoft's newest version of QuickBASIC by the authors of *The BASIC Teacher*. The book approaches QuickBASIC's programming environment in three stages so beginning and experienced BASIC programmers can find the appropriate level of instruction. You'll learn about subprograms, libraries, meta-commands, dynamic debugging, and the important advantage of QuickBASIC's speedy graphics. (OMH-881514 \$22.95)

Still Available

Using QuickBASIC 4.0 (OMH-881274 \$19.95)

QuickBASIC: The Complete Reference by Steven Nameroff (Osborne/McGraw-Hill, 700pp). A comprehensive guide for users of all levels of programming ability from novices to pros. Nameroff has divided the book into sections to help you easily locate the information you need. *QuickBASIC: The Complete Reference* begins with a quick introduction to BASIC programming, followed by a complete command reference section and a discussion of QuickBASIC functions, procedures, files, and graphics. Advanced techniques and applications are grouped together in the last section of the book for the professional QuickBASIC programmer. (OMH-881362-X \$26.95)



Advanced QuickBASIC by Don Inman, Bob Albrecht, and Arne Jamtgaard (Osborne/McGraw-Hill, 500pp). As an experienced programmer you'll learn how to write professional-level programs using the advanced techniques found in the 4.5 version of QuickBASIC. *Advanced QuickBASIC*, written by three highly regarded professional programmers, provides you with the tools to produce sophisticated programs. Graphics, construction kits, and data bases are some of the topics discussed. Find information on Quick libraries, macros, keyed, sequential, and unsequential files and more. *Advanced QuickBASIC* is packed with applications, examples, and models that will soon have you programming like an expert. (OMH-881361 \$21.95)



Introduction

BASIC HAS a small **vocabulary** and a simple **syntax** (grammar). We have already discussed some of the special words that Microsoft BASIC understands. They are called **keywords** or **reserved words**. Here are the keywords introduced and described previously:

BEEP	INPUT	NEW	RND	TIMER
CLS	INPUT\$	NEXT	RUN	USING
COLOR	INT	OFF	SAVE	WIDTH
DATE\$	KEY	ON	SOUND	
FILE	LIST	PRINT	SYSTEM	
FOR	LOAD	RANDOMIZE	TAB	
GOTO	LOCATE	REM	TIME\$	

We always show keywords in all upper-case letters and variables in lower-case or a MiXtuRe of upper- and lower-case letters. We think this makes programs easier to read and understand. However, when you LIST a program, unfortunately, variables will probably appear in all upper-case letters.

GW-BASIC Made Easy

By Bob Albrecht & Don Inman

FINALLY, IT'S finished! *GW-BASIC Made Easy* is the last of four big books (400+ pages each) that have occupied our time-and-a-half for 18 months. We have been overworked! No more BIG writing projects for a while.

GW-BASIC Made Easy is for beginners, people with no previous programming experience.

Chapters: Getting Started; Do It Now—Immediate Operations; Introduction to Programming; Number Crunching; Making Programs More Useful; Control Structures; Function Junction; Subroutines; Arrays; Sequential Files; Random Access Files; Graphics.

PCM

WE USE Tandy computers for work and play. If you do, too, be sure to look at *PCM*, the magazine for Tandy computer users (Models 1000, 1200, 1400, 2000, 3000, and 4000). Usually has several BASIC programs and a couple of BASIC tutorials. Ranks high on our "must read" list. PCM, P.O. Box 385, Prospect, KY 40059. \$28/year (12 issues).

CodeWorks

A MAGAZINE entirely about BASIC on MS-DOS computers and the Tandy Models I, III, and IV. Each issue has long BASIC programs plus tutorials, hints, letters, and other stuff. Seems oriented to intermediate to advanced users. CodeWorks, 3838 S Warner St., Tacoma, WA 98409. \$24.95/year (6 issues).

Decisions, Decisions

THE IF statement tells the computer to make a simple decision. It tells the computer to do a certain operation **if** a given condition is **true**. However, if the condition is **false**, the operation is not done. Here is a simple IF statement:

```
IF number = 0 THEN PRINT z$
```

This IF statement tells the computer:

- If the value of the numeric variable **number** is equal to zero (0), then print the value of the string variable **z\$**.
- If the value of the numeric variable **number** is not equal to zero, then don't print the value of the string variable **z\$**.

Here is another way to think about it:

- If the **condition** **number = 0** is **true**, then do the statement following the keyword **THEN**.
- If the **condition** **number = 0** is **false**, then don't do the statement following the keyword **THEN**.

IF condition THEN statement

AN IF statement can consist of the keyword **IF**, followed by a condition, followed by the keyword **THEN**, followed by a statement. The **statement** can be any GW-BASIC statement. The **condition** is usually a comparison between a variable and a value, between two variables, or between two expressions. Below is a handy table that shows some comparison symbols in both math notation and GW-BASIC notation.

Comparison	Math	GW-BASIC
is equal to	=	=
is less than	<	<
is greater than	>	>
is less than or equal to	≤	<=
is greater than or equal to	≥	>=
is not equal to	≠	<>



THIS IS the condition

IF number = 0 THEN PRINT z\$

Do this if the condition is **true**
Don't do this if the condition is **false**

Suppose the value of **number** is **1.23**.

The condition is **false** and the computer does not print the value of **z\$**.

Suppose the value of **number** is **0**.

The condition is **true** and the computer prints the value of **z\$**.

Suppose the value of **number** is **-2**.

The condition is **false** and the computer does not print the value of **z\$**.

Negative, Zero, or Positive

HERE IS an easy problem. We want to write a program to ask for a number, then tell whether the number is negative, zero, or positive. When we run the program, we want it to go like this:

Type a number and press the ENTER key.
I will tell you whether your number is
negative, zero, or positive.

Number, please? 0
zero

Number, please? 1.23
positive

Number, please? -2
negative

Number, please? _



IF Statements Can Do It!

PROGRAM TYB0901 uses three IF statements. Look for them in lines 340, 350, and 360.

Line 320 acquires a value of the numeric variable number that you enter from the keyboard. This value is then checked by lines 340, 350, and 360.

If the value of number is negative, line 340 prints the value of the n\$, which is "negative." (See line 110.)

If the value of number is zero, line 350 prints the value of z\$, which is "zero." (See line 120.)

If the value of number is positive, line 360 prints the value of p\$, which is "positive." (See line 130.)

For any number, only one message is printed.

Program TYB0901

Negative, Positive, or Zero #1

```

1 REM ** Negative, Positive, or Zero #1 ***
2 ' The BASIC Teacher #9.  Filename: TYB0901.BAS

100 REM ** Assign messages to string variables **
110 n$ = "negative"
120 z$ = "zero"
130 p$ = "positive"

200 REM ** Tell what to do **
210 CLS : KEY OFF
220 PRINT "Type a number and press the ENTER key."
230 PRINT "I will tell you whether your number is"
240 PRINT "negative, zero, or positive."

300 REM ** GOTO loop: Get number and tell about it **
310 ' Ask for a number
320 PRINT : INPUT "Number, please"; number      'Top of loop

330 ' Tell about number
340 IF number < 0 THEN PRINT n$
350 IF number = 0 THEN PRINT z$
360 IF number > 0 THEN PRINT p$

370 GOTO 320                                     'Bottom of loop

```


Checkbook Balancer

SUPPOSE YOU have misplaced your \$7 solar-powered calculator and are faced with the dreary prospect of balancing your checkbook. Relax—use your computer to do the work.

We wrote a bare-bones, no frills program to help you use your \$1000 computer instead of your \$7 calculator to balance your checkbook. A sample run is shown below. Program **TYB0902** is on the next page.

When you run the program, remember to enter checks as negative numbers and deposits as positive numbers. After you have entered all checks and deposits, then enter zero (0) to quit. The program checks every entry to see if it is zero. This program knows when to quit, if you enter zero to tell it when.



A Sample Run

The date is 08-05-1989

The time is 19:20:05

I will help you balance your checkbook.

When I ask, type your old balance and press ENTER.

Type a check as a negative number and press ENTER.

Type a deposit as a positive number and press ENTER.

To quit, type zero (0) and press ENTER.

Old balance? 123.45

First, enter the old balance.

Check or deposit? -3.95

Enter checks as negative numbers.

The new balance is 119.5

Check or deposit? -25

The new balance is 94.5

Check or deposit? -89.25

The new balance is 5.25

Check or deposit? 50

At last, a deposit! And just in time.

The new balance is 55.25

Check or deposit? 0

Enter zero (0) to quit.

The final balance is 55.25

Ok

—

Program TYB0902

```

1 REM ** World's Most Expensive Checkbook Balancer **
2 ' The BASIC Teacher #9.  Filename: TYB0902.BAS

100 REM ** Set up **
110 CLS : KEY OFF
120 PRINT "The date is "; DATE$
130 PRINT "The time is "; TIME$

200 REM ** Tell what to do **
210 PRINT
220 PRINT "I will help you balance your checkbook."
230 PRINT "When I ask, type your old balance and press ENTER."
240 PRINT "Type a check as a negative number and press ENTER."
250 PRINT "Type a deposit as a positive number and press ENTER."
260 PRINT "To quit, type zero (0) and press ENTER."

300 REM ** Get the old balance **
310 PRINT
320 INPUT "Old balance"; Balance

400 REM ** Get checks & deposits, compute & print new balance **
405 ' Use GOTO loop -- exit on zero
410 PRINT                                'Top of loop
420   INPUT "Check or deposit"; CheckOrDeposit
430   IF CheckOrDeposit = 0 THEN 510
440   Balance = Balance + CheckOrDeposit
450   PRINT "The new balance is"; Balance
460 GOTO 410                            'Bottom of loop

500 REM ** Print final balance & end program **
510 PRINT
520 PRINT "The final balance is"; Balance
530 END

```

A Minor Modification

USE TAB and PRINT USING to make the new balances line up as shown here.

Old balance?	123.45	
Check or deposit?	-3.95	
The new balance is		119.50
Check or deposit?	-25	
The new balance is		94.50
Check or deposit?	_	

Note that two digits are printed to the right of the decimal point.

Total Checks? Total Deposits?

IT WOULD be nice to have the computer keep a running total of the checks and deposits, then print these totals along with the old balance and new balance, perhaps like this:

The old balance is	123.45
Total of checks is	-118.20
Total of deposits is	50.00
The new balance is	55.25

"May the dragons of good fortune keep your balances positive."

—Laran Stardrake

Your Turn

HERE ARE some problems for you to ponder:

PROBLEM 1

Negative, Zero, or Positive #2

We showed you one way to solve our Negative, Zero, or Positive problem. There's always another way. We know several more ways and will show at least one next time. We challenge you to write another program, different from ours:

- Don't use IF statements
- Look up ON ... GOTO and SGN in your BASIC reference manual. You can use these to do it a different way.
- Or ... surprise us with your original method.

PROBLEM 2

English Name of a Decimal Digit

The decimal digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

Their English names are zero, one, two, three, four, five, six, seven, eight, and nine.

Write a program to ask for a decimal digit (0 to 9), then print the English name of the digit. Our program goes like this:

```
Decimal digit? 3
three

Decimal digit? 10
That's not a decimal digit!

Decimal digit? 0
zero

Decimal digit? -1
That's not a decimal digit!

Decimal digit? 2.5
That's not a decimal digit!

Decimal digit? _
```



Can you solve this problem in at least two quite different ways?
More ways?!

By Don Inman

WE WILL talk a little bit more about your Disk Operating System (DOS) in this issue. We've talked about how DOS is quite often the most ignored part of a computer system. DOS is necessary, but it is usually used only for the most basic things—like formatting, copying disks and disk files, and loading the software that we use.

In "Browsing BASIC #3," we discussed how to label disks and how to use DOS to write a CONFIG.SYS file to the DOS disk. In "Browsing BASIC #6," we used COPY CON to turn DOS into a notepad to write short text files. We also discussed the two distinct types of DOS commands: **Resident** commands and **Transient** commands. The DOS features discussed were for MS-DOS, version 3.2 and 3.3.

In this episode, we will browse information on a disk's main directory (called the **root directory**), **subdirectories**, and **file paths**.

The MS-DOS version 3.20 used here is Tandy version 03.20.21. As you browse the following discussion, remember that there may be slight differences between this version and the version of MS-DOS that you are using.

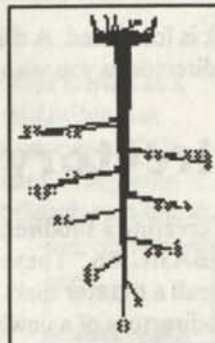
Microsoft MS-DOS Version 3.20
(C) Copyright Microsoft Corp 1981, 1986
Tandy Version 03.20.21
Licensed to Tandy Corp.
All rights reserved.

Organizing Disk Files

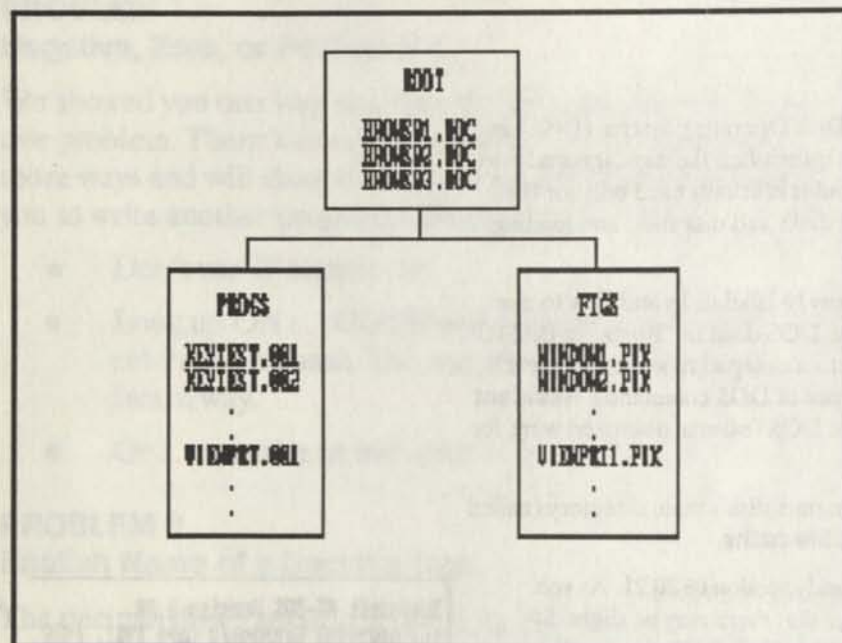
MANY OF us only use the **root directory** when saving files to a disk. If you have not specified a file path for DOS to use when searching for a file, it will look only in the root directory. For many purposes the root directory is all that is needed.

As the number of files on a disk grows, it is often beneficial to separate the files into some type of organized system. This can be done by using **subdirectories**. You can visualize the organization of your files in such a system as an inverted tree with its roots up in the air and its branches reaching downward and outward.

The root directory of a disk is represented by the roots of the inverted tree. Subdirectories are represented by the branches. The first level of subdirectories are accessed directly from the root directory. The next level of subdirectories are accessed through the first-level subdirectories.



When preparing "Browsing BASIC" articles for *The BASIC Teacher*, I use one disk for several articles. I save the text of the articles in the root directory of the disk. Files used to produce figures are placed in a subdirectory related to the article as shown in the following figure.



DOS Directory Commands

THE **transient** DOS commands used for manipulating directories include:

- MKDIR – to make a new directory
- CHDIR – to change to a different directory
- RMDIR – to remove a directory
- DEL – to delete a file (when used with a directory name)

The root directory is created when a disk is formatted. A disk has only one root directory. You must create any subdirectories you wish to add.

Creating A Subdirectory

WE WILL now go through the process of creating a subdirectory called **FIGS** that will store figures used in "Browsing BASIC #6." These files now exist in a hodgepodge with others on what I will call a **master** disk. The files will be transferred from the master disk to a subdirectory of a newly formatted disk that is given the title, **BROWSING**.



The names used for the drives in this demonstration are drive A, drive B, and drive D. Drives B and D are the logical names that I assigned to my 5-1/4" drive in "Browsing BASIC #3." Assuming you are starting in disk drive A, the disk is formatted and the subdirectory is created by the following steps:

1. Put the new disk in drive B and format it with the **V** option:

```

A>FORMAT B:/V
Insert new diskette for drive B:
and strike ENTER when ready

Format now complete

Volume label (11 characters, ENTER for none)? _
  
```

2. Label the disk **BROWSING**:

```

A>FORMAT B:/V
Insert new diskette for drive B:
and strike ENTER when ready

Format complete

Volume label (11 characters, ENTER for none)? BROWSING

362496 bytes total disk space
362496 bytes available on disk

Format another (Y/N)?
  
```

3. Access drive B with the command:

```
A>B:
```

4. The command to create the **BROWS6** subdirectory on the new disk is:

```
B>MKDIR \PROGS or B>MD \PROGS
```

Note that you use the **backslash** (\), not the regular slash that is used as a division sign when performing arithmetic. A backslash is a **delimiter** that precedes any subdirectories name. The backslash is used to specify to DOS that the name represents a subdirectory. It also is used at times to separate the subdirectory's name from the directory in which the subdirectory is found.

When step 3 has been completed, the top of the screen appears as shown at the right. Drive B was accessed, the **MKDIR** command given with the specified subdirectory. The subdirectory is created, and the computer waits for the next command.

```

A>B:
B>MKDIR \PROGS
B>
  
```


If you look at the directory of the disk after the subdirectory has been created, you will see the subdirectory's name (**PROGS**). Notice that the letters, **DIR**, are enclosed in left and right carets **<DIR>** in place of the extension used for files. Also note that the subdirectory is included in the file count of the root directory.

Adding Files to the PROGS Subdirectory

AFTER THE subdirectory has been created, you can add files to it. The basic DOS commands (**COPY**, **ERASE**, etc.) will work with subdirectories.

To add the first program file from "Browsing BASIC #1," the following steps were used.

1. The master disk with the programs from "Browsing BASIC #1" was placed in drive B.
2. Drive B was accessed and the file name entered. The target drive and the subdirectory where the copy is to be placed follows the file name.

```
B>COPY KEYTEST.001 D:\PROGS
```

<i>DOS</i>	<i>source</i>	<i>target disk and</i>
<i>command</i>	<i>file name</i>	<i>subdirectory</i>

Since I am using drive B and drive D to name the same physical 5-1/4" disk drive, the computer prints a prompt to insert the diskette in drive D: and strike any key when ready.

3. Put the newly formatted disk in the 5-1/4" drive (B/D) and press ENTER. The file is copied.

```
B>COPY KEYTEST.001 D:\PROGS
```

```
Insert diskette for drive D: and strike  
any key when ready
```

```
1 File(s) copied
```

```
B>_
```

To see that the file has been copied to the desired subdirectory, the subdirectory is included with the **DIR** command as follows:

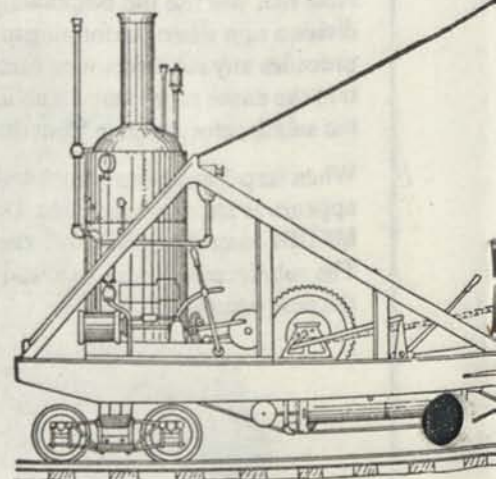
```
B>DIR \PROGS
```

```
B>DIR
```

```
Volume in drive B is BROWSING  
Directory of B:\
```

```
PROGS      <DIR>      8-04-88  18:24a  
1 File(s)  361472 bytes free
```

```
B>
```



When this directory command is executed, the directory appears on the screen as follows:

```
B>DIR \PROGS

Volume in drive B is BROWSING
Directory of B:\PROGS

.                <DIR>          8-04-88  10:24a
..               <DIR>          8-04-88  10:24a
KEYTEST 001      207    1-23-88   3:33p
      3 File(s)    356352 bytes free

B>
```

The disk title (**BROWSING**) is given on the first line of the directory. The second line shows the subdirectory title. The one-dot and two-dot entries are references to the current directory and the parent directory and are seldom used. Ignore them for the present. Then the **KEYTEST.001** file is listed, followed by the number of files and the number of free bytes left on the disk. The one-dot and two-dot entries are counted as files.

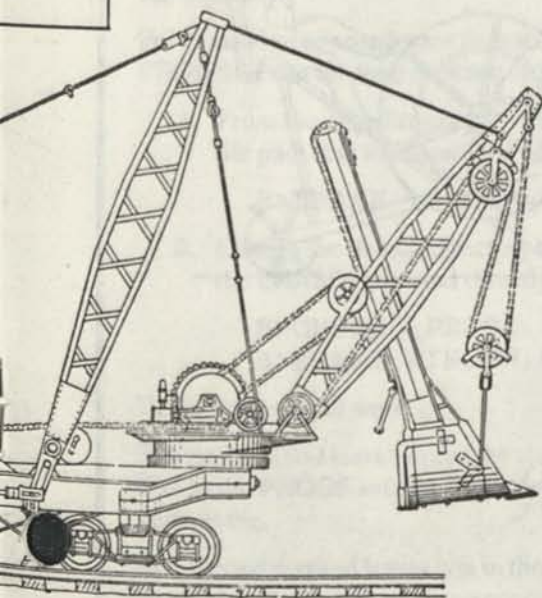
I continue copying program files from "Browsing BASIC #1" and "Browsing BASIC #2" using the same three-step process. When all the program files for these two articles have been entered, the **PROGS** subdirectory displays the following:

```
B>DIR \PROGS

Volume in drive B is BROWSING
Directory of B:\PROGS

.                <DIR>          8-04-88  10:24a
..               <DIR>          8-04-88  10:24a
KEYTEST 001      207    1-23-88   3:33p
KEYTEST 002      302    1-23-88   3:32p
KEYTEST 003      279    1-23-88   3:38p
KEYTEST 004      1136   1-23-88   3:57p
VIEWPRT 001      586    1-30-88   3:42p
WINDOW  001      1620   1-31-88   6:10p
      8 File(s)    353200 bytes free

B>
```



Creating and Adding Files to the FIGS Subdirectory

A NEW subdirectory is created in the same way as the PROGS subdirectory. This time it is named **FIGS** and is created on the same disk.

```
B>MKDIR \FIGS
```

After the directory has been created, the root directory of the disk titled **BROWSING** shows the following:

```
B>DIR
Volume in drive B is BROWSING
Directory of B:\

FIGS      <DIR>      8-04-88  3:35p
PROGS     <DIR>      8-04-88  10:24a
      2 File(s)    352256 bytes free

B>
```

The figure files are copied from the master disk in the same way the program files were copied. For example:

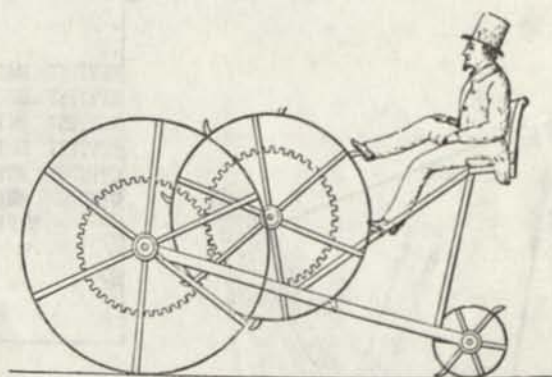
```
B>COPY VIEWPRT1.PIX D:\FIGS
```

After all six of the figure files had been copied into their subdirectory, the new subdirectory contains:

```
B>DIR \FIGS
Volume in drive B is BROWSING
Directory of B:\FIGS

.          <DIR>      8-04-88  3:35p
..         <DIR>      8-04-88  3:35p
VIEWPRT1.PIX 2891  3-14-88  6:43p
VIEWPRT2.PIX 2328  3-14-88  7:28p
WINDOW1.PIX  1969  1-30-88  1:07p
WINDOW2.PIX  2899  1-31-88  6:23p
WINDOW3.PIX  1981  1-30-88  1:41p
WINDOW4.PIX  1973  1-30-88  2:04p
      8 File(s)    334048 bytes free

B>_
```



One chore remains. That is copying the text file for "Browsing BASIC #1" and "Browsing BASIC #2" into the root directory. We have used the root directory many times before, so this is an easy job with the copy command.

```
B>COPY BROWSE1.DOC D: or B>COPY BROWSE1.DOC D:\
B>COPY BROWSE2.DOC D: or B>COPY BROWSE2.DOC D:\
```


The backslash, when used alone following the disk drive letter, specifies the root directory. It is optional in this case since we are operating from the root directory. When using subdirectories, the backslash gives a clear indication that you are referring to the root directory.

When these two documents have been copied to the root directory, the disk organization for two articles is complete. The root directory shows the titles of the two subdirectories plus the two text files for "Browsing BASIC #1" and "Browsing BASIC #2."

```
CDIR
```

```
Volume in drive B is BROWSING
Directory of B:\
```

```
FILES      <DIR>      8-04-88  3:35p
PROGS      <DIR>      8-04-88  10:24a
BROWSE1    DOC       17624  3-30-88  9:40p
BROWSE2    DOC       23151  1-23-88  12:31p
4 File(s)  294912 bytes free
```

```
B_
```

The same procedure is carried out for other "Browsing BASIC" articles.

Accessing A Directory

THE ROOT directory is normally the default directory for a disk. However, DOS allows you to select (or change) the default directory. Changing to a given directory or subdirectory, allows you to work directly with the files contained in the selected directory or subdirectory. Then you don't have to enter the directory name in order to access a file tucked away in a seemingly distant subdirectory.

For example, suppose I want to erase a file that resides in the subdirectory, **PROGS**. It can be done in either of the following two ways.

1. From the root directory, give the **ERASE** command followed by the file path that will locate the file you wish to erase:

```
B>ERASE \PROGS\WINDOW.001
```

2. Change the default directory to the subdirectory, **PROGS**. Then give the **ERASE** command directly. No file path is needed.

```
B>CHDIR \PROGS
B>ERASE WINDOW.001
```

Either method will work.

The first method leaves you in the root directory. Any DOS commands for files in the **PROGS** subdirectory must use the long file path to find the desired file.

The second method leaves you in the **PROGS** subdirectory. Any DOS commands for files in this subdirectory can be given directly without the file path.

Files for this Article

The writing that I do is sent to various publishers. I find the organization of files discussed is useful for myself as well as for copying disks to be sent to publishers. For this particular article, the root directory of the publisher's disk is:

```
Volume in drive B is BROWSEB9
Directory of B:\

FIGS      <DIR>      9-28-89  1:22p
BROWSEB9.DOC  18892  9-15-89  3:46p
2 File(s)  312328 bytes free
```

There are no programs in this article. Therefore, there is no program subdirectory. The FIGS subdirectory is:

```
Volume in drive B is BROWSEB9
Directory of B:\FIGS

.      <DIR>      9-28-89  1:22p
..     <DIR>      9-28-89  1:22p
FIG9-1.PIX  3433  9-11-89  6:45p
FIG9-2.PIX  3462  9-11-89  6:48p
FIG9-3.PIX  3642  9-11-89  6:51p
FIG9-4.PIX  3419  9-11-89  6:53p
FIG9-5.PIX  1677  9-11-89  6:55p
FIG9-6.PIX  1379  9-12-89  18:14a
FIG9-7.PIX  1528  9-12-89  18:15a
FIG9-8.PIX  3267  9-12-89  18:28a
FIG9-9.PIX  1586  9-12-89  18:21a
FIG9-10.PIX 1668  9-12-89  18:28a
FIG9-11.PIX 1549  9-12-89  18:30a
FIG9-12.PIX 1659  9-12-89  18:32a
FIG9-13.PIX 3751  9-12-89  18:34a
FIG9-14.PIX 1527  9-28-89  12:31p
FIG9-15.PIX 1787  9-28-89  12:33p
17 File(s) 388224 bytes free
```

We will continue DOS file manipulation at a later time. If you have particular DOS features that you would like discussed, send suggestions to **The BASIC Teacher**, 2814 - 19th Street, San Francisco, CA 94110.





The People's Computer Language

QUICKBASIC is rapidly replacing GW-BASIC as the standard general-purpose programming language, the People's Computer Language. Universities will adopt QuickBASIC as the language for students in engineering, business, science, and most any discipline other than computer science (who will use a language called C). QuickBASIC, or something very similar, will become the macro language in applications software, such as word processors, database managers, and spreadsheets. If you know QuickBASIC, you can tell the computer to do it **your way!**

Screen Modes

YOUR COMPUTER probably has at least three screen modes, as follows:

- *Text only*
- *Medium-resolution graphics*
- *High-resolution graphics*

The text mode allows you to print ASCII characters on the screen: letters, numbers, punctuation, special characters (\$, #, @, etc.), foreign alphabets, and graphics **characters**.

The graphics modes allow you to plot tiny dots called **pixels** on the screen; draw lines, boxes, circles, and more complex shapes; and also print text and other characters.

SCREEN 0

WHEN YOU first start QuickBASIC, it automatically selects the text mode for the Output Screen. This is **SCREEN 0**. You can use a **SCREEN 0** statement in your program or in the Immediate Window to select this mode.

In **SCREEN 0** you can print text and other ASCII characters in 16 **foreground** colors on 8 **background** colors. You use the **COLOR** statement to select colors. Of course, you must have a color monitor to see these colors!

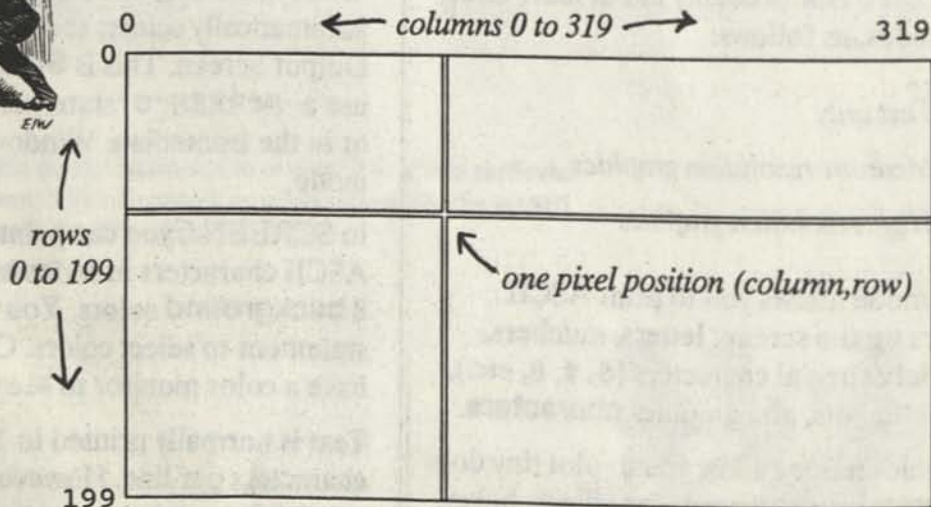
Text is normally printed in 25 lines with 80 characters per line. However, you can use a **WIDTH 40** statement to tell the computer to print in 25 lines with 40 characters per line.

Graphics Modes

IF YOUR computer has a Color Graphics Adapter (CGA), it has at least two color graphics screens:

- **SCREEN 1** is a medium-resolution graphics screen with 16 background colors and 4 foreground colors at any one time. Text is printed 40 characters per line.
- **SCREEN 2** is a high-resolution graphics screen with one background color (black) and one foreground color (white). These colors may be different on a monochrome monitor. Text is printed 80 characters per line, the same as in SCREEN 0.

If your computer has an Extended Graphics Adapter (EGA) or Video Graphics Array (VGA), it has additional screen modes. The Tandy 1000 computers also have enhanced CGA with additional screen modes.



SCREEN 1

WE WILL start with standard CGA capabilities and begin with things you can do in SCREEN 1. For text, you can use SCREEN 1 just like you use SCREEN 0 in its 40 characters per line mode. Enter this tiny program in the View Window and run it.

```
SCREEN 1
PRINT "This is SCREEN 1"
```

The Output Screen will appear as shown below in big letters (same as WIDTH 40 in SCREEN 0):

This is SCREEN 1

Press any key to continue

For graphics, the screen is divided into 64,000 tiny rectangles called **pixels**. Pixel is short for "picture element." A pixel position is identified by two **coordinates**, a column number (0 to 319) and a row number (0 to 199), as shown below:

Background Colors

IN SCREEN 1, the **default** background color is color #0, black. You can use a **COLOR** statement to select any of 16 background colors, including black. Use the following tiny program to color the entire screen blue.

```
SCREEN 1
COLOR 1
```

Run the program. You will see a blue screen, empty except for the familiar "Press any key to continue..." on the bottom line of the screen. Note that this message is in double-width letters. The statement **COLOR 1** selects blue as the background screen color.

Press any key to return to the View Window, then change the program slightly, as follows:

```
SCREEN 1
COLOR 2
```

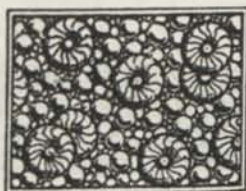
Run the program. Now you see a green screen. The statement **COLOR 2** selects green as the background screen color. Press a key to return to the View Window and change the program again:

```
SCREEN 1
COLOR 3
```

Run the program. This time you see a cyan screen. Cyan is a pale blue color. Press a key to return to the View Window and change the program to the one shown here:

```
SCREEN 1
COLOR 7
```

Run the program to get a light gray screen. Oops! What happened to the "Press any key to continue..." message? Well, it's there, but in the same light gray color as the background color—so it is invisible.



You Pick a Background Color

YOU CHOOSE a color number from 0 to 15 and complete the program.

```
SCREEN 1
COLOR ____ <-- your color number
```

Here is a table of available background colors and color numbers.

COLOR number	Color selected	COLOR number	Color selection
0	black	8	dark gray
1	blue	9	bright blue
2	green	10	bright green
3	cyan	11	bright cyan
4	red	12	bright red
5	magenta	13	bright magenta
6	brown	14	yellow
7	light gray	15	white

Of course, the colors you see on your color monitor might be somewhat different, depending on how you set your color control dials & knobs. On our screen, **COLOR 6** ("brown") looks more like orange. Good! We can use red, orange (**COLOR 6**), yellow, green, blue, and magenta for the six colors of the rainbow.

What happens if you try **COLOR 16** or **COLOR 17** or **COLOR 120**? Try it and find out. Then try this program:

```
REM ** SCREEN 1 Background Colors **
' The BASIC Teacher #9. Filename: TYQB0901.BAS

SCREEN 1

ColorNumber = 0

DO
  COLOR ColorNumber
  CLS
  PRINT "COLOR"; ColorNumber
  akey$ = INPUT$(1)
  ColorNumber = ColorNumber + 1
LOOP
```

Run this program. It begins with the message "COLOR 0" on a black screen. Press a key to see "COLOR 1" on a blue screen. And so on. Run through colors 0 thru 15, then 16, 17, 18, and so on.

Pick a Palette

THINK OF the screen as a canvas available in 16 background colors. You can pick a palette of **foreground** colors for drawing on the screen. The Color Graphics Adapter (CGA) provides two palettes, each with four colors numbered 0, 1, 2, and 3.

Palette #0 has the following colors:

- 0 is the background color
- 1 is green
- 2 is red
- 3 is brown (orange on our screen)

Palette #1 has the following colors:

- 0 is the background color
- 1 is cyan
- 2 is magenta
- 3 is light gray

Remember, the above numbers are **palette color numbers**, not to be confused with the numbers that pick background colors for the screen.

You pick a palette by including the palette number, 0 or 1, in a **COLOR** statement. For example:

The statement: **COLOR 0, 1**

picks black as the background color and palette #1 for the foreground colors. So the foreground colors will be 0 for black, 1 for cyan, 2 for magenta, and 3 for light gray.

The statement: **COLOR 15, 0**

picks bright white for the background color and palette #0 for the foreground colors. The foreground colors will be 0 for bright white, 1 for green, 2 for red, and 3 for brown. Brown looks like orange on our screen

Note that text is printed in palette color #3, brown (or orange) in palette #0, or light gray in palette #1.

A **COLOR** statement to select a background color and a palette consists of the word **COLOR**, a number from 0 to 15, a comma, and 0 or 1.

COLOR ,
 / \
 a number, 0 to 15 0 or 1

Well, it's time to pick a canvas color (background color), a palette of four colors, and plot some pixels.

Plot a Pixel

A **pixel** is a picture element, a tiny rectangle on the screen. Use a **PSET** statement to plot a pixel. **PSET** means *pixel set*. If you want to plot a pixel, you must tell the computer where to put it. To do so, you name the column and row where you want your pixel to go. You also tell what color you want (0, 1, 2, or 3) in a previously selected palette. For example:

The statement: **PSET (0, 0), 1**

tells the computer to plot a pixel at column 0, row 0 in palette color number 1. In palette 0, this is green; in palette 1, it is cyan. Column 0, row 0 is the upper left corner of the screen.

The statement: **PSET (319, 199), 2**

tells the computer to plot a pixel at column 319, row 199 in palette color number 2. In palette 0, this is red; in palette 1, it is magenta. Column 319, row 199 is the bottom right corner of the screen.



Plot Pixels in Four Corners

PROGRAM **TYQB0902**, shown at the bottom of this page, plots a pixel in each of the four corners of the screen. The pixels appear on a black background.

The statement: `COLOR 0, 1`

picks black as the background color and palette 1 for the foreground colors.

`COLOR 0, 1`

black background palette #1

The statement: `PSET (0, 0), 1`

plots a pixel in the upper left corner. This pixel is cyan. Why? Because palette #1 was previously picked.

`PSET (0, 0), 1`

location of pixel palette color number

The statement: `PSET (319, 0), 2`

plots a pixel in the upper right corner. This pixel is magenta in palette #2.

The statement: `PSET (319, 199), 3`
plots a pixel in the lower right corner. This pixel is light gray in palette #1.

The statement: `PSET (0, 199), 0`

plots a pixel in the lower left corner. This pixel is the background color, so you can't see it. A pixel plotted in palette color number 0 is always invisible—so why bother? Well, you can use it to erase a pixel that was previously plotted in another color.

After plotting four pixels in four corners of the screen, the computer waits for someone to press a key. Press a key and the program ends.

The statement: `akey$ = INPUT$(1)`

tells the computer to wait for someone to press one key. When a key is pressed, the string value of the key is assigned as the value of the string variable `akey$` and the computer moves on to the next statement. In this program, there isn't any next statement, so the program ends with the usual message, "Press any key to continue..."

Program TYQB0902

```
REM ** Plot Pixels in the Four Corners of SCREEN 1 **
```

```
' The BASIC Teacher.  Filename: TYQB0902.BAS
```

```
SCREEN 1
```

```
COLOR 0, 1
```

```
'Black background, palette 1
```

```
PSET (0, 0), 1
```

```
'Upper left corner, palette color 1
```

```
PSET (319, 0), 2
```

```
'Upper right corner, palette color 2
```

```
PSET (319, 199), 3
```

```
'Lower right corner, palette color 3
```

```
PSET (0, 199), 0
```

```
'Lower left corner, palette color 0
```

```
akey$ = INPUT$(1)
```

```
'Wait for a key press to end program
```


Zappy Artist Plots Pixels

WHILE WE were writing this episode of "Teach Yourself QuickBASIC," Zappy Artist stopped by. Zappy likes to zap around the screen plotting random pixels, and drawing random lines, boxes, circles, and other stuff.

Zappy spent an hour or so humming to himself and entering programs. Then, "Gotta go," said Zappy. Without further ado, he handed us a disk and left, promising to return soon and explain how his programs work.

One of Zappy's programs is shown below. Since you don't have Zappy's disk, enter it from the keyboard and run it. Also try a variation, as follows:

Instead of: COLOR 0, 1

Try: COLOR 15, 0

Or, you pick the background color number (0 to 15) and palette number (0 or 1).



Program TYQB0903

```
REM ** Zappy Artist Plots Pixels Everywhere **
' The BASIC Teacher.  Filename: TYQB0903.BAS
```

```
REM ** Set up **
DEFINT A-Z
RANDOMIZE TIMER
SCREEN 1
COLOR 0, 1
```

'Medium resolution graphics
'Black background, palette #1

```
REM ** Plot pixels in random places & colors, with sound **
```

```
DO
```

```
'Compute random place and color
col = INT(320 * RND)
row = INT(200 * RND)
Pcolor = INT(3 * RND) + 1
```

'Random column: 0 to 319
'Random row: 0 to 199
'Random palette color: 1, 2, or 3

```
'Plot a pixel
PSET (col, row), Pcolor
```

```
'Make a short random sound
frequency! = 4000 * RND + 37
SOUND frequency!, .25
```

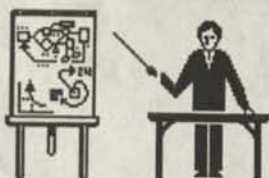
'Try other frequency ranges
'Try other durations

```
LOOP
```

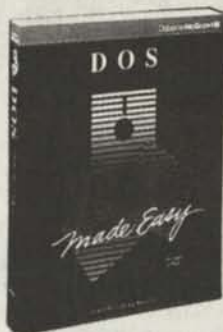
To stop the program, hold down CTRL and press BREAK. If you have a Tandy 1000TX or older Tandy 1000, hold down CTRL and press HOLD.

For Power Users!

The Shareware Book: Using PC-Write, PC-File+ & PC-Calc+, Second Edition by Ramon Zamora, et al (Osborne/McGraw-Hill, 744pp). This outstanding guide combines instruction and reference material into one single resource that you'll use again and again. Learn to use *PC-Write* (word processor), *PC-File+* (database) and *PC-Calc+* (spreadsheet) with ease and efficiency. (OMH-881591 \$22.95)



Ideas



TBT-1 Issue No. 1.	..	\$3.00
TBT-2 Issue No. 2.	..	\$3.00
TBT-3 Issue No. 3.	..	\$3.00
TBT-4 Issue No. 4.	..	\$3.00
TBT-5 Issue No. 5.	..	\$3.00
TBT-6 Issue No. 6.	..	\$3.00
TBT-7 Issue No. 7.	..	\$3.00
TBT-8 Issue No. 8.	..	\$3.00

Due to circumstances beyond our control, this issue is late again. Our landlord sold the building and we had to make an emergency move. Luckily we found suitable quarters across the street. And then the earthquake hit. No one

was hurt but it threw our schedule off like we never imagined. Please bear with us and we should be back on schedule soon. Subscribers will get the full number of issues promised. The number next to your name on the mailing label on the back page is the last issue in your sub.

Order Form

Send check or money order.
Foreign customers send
international money order
available at post offices and
banks.

Name _____
Address _____
City/State _____ Zip _____
Phone () _____

[illegible]

SATISFACTION GUARANTEED

Postage & Handling: Add \$2.00 for first item and \$1.00 for each additional item. To shipping points outside U.S. add \$3.00 for first item and \$1.00 for each additional item. Shipped via surface mail.

NEW BOOKS ON BASIC

HANDBOOK OF BASIC: THIRD EDITION

FOR THE IBM PC, XT, AT, PS/2,
AND COMPATIBLES

David I. Schneider
University of Maryland



Handbook of BASIC: Third Edition by David I. Schneider. "I can recommend [this book] without reservation to everyone, whether you're a beginner, semi-professional, or professional programmer. Anyone who takes the time to look between the covers will find that this book is indeed a very valuable reference."—Richard Aarons, *PC Magazine*. Update your programming library with this third edition of the most thorough and complete reference book on the BASIC language for PCs and compatibles ever put together. Complete and detailed explanations of statements are illuminated by over 600 examples. Each listing has the most popular form of the BASIC command, followed by subtle variations and extensions. Both a ready source and a tutorial, this is the most comprehensive BASIC reference manual available! (BRA-372582-0 \$24.95)

Advanced BASIC for the IBM PC and Compatibles: Tips and Techniques by Larry Joel Goldstein. Now that you're ready to make more sophisticated use BASIC, this book will help you develop your skills right. This book is a thorough, hands-on guide to BASIC's complex commands for creating more powerful programs. As you work through the key principles in the text, you'll literally build a program called the Bar Chart Generator. This practical case study reinforces all the valuable techniques you'll learn, to let you go on to create your own applications programs. (BRA-010307-1 \$19.95)

Hands-On QuickBASIC by Larry Joel Goldstein. You've chosen Microsoft's bestselling compiler to get programs up and running quickly. Now choose *Hands-On QuickBASIC* to get all the information you need to learn and use it quickly and efficiently. Clear, concise tutorials teach you everything you need to know. First-time programmers will find author Larry Joel Goldstein's approach both accessible and complete. Experienced users will benefit from his coverage of advanced topics. *Hands-On QuickBASIC* starts as a tutorial, then becomes a reference that you'll return to over and over again. (BRA-383480-8 \$21.95)

Math History & Fun

Martin Gardner
ENTERTAINING
MATHEMATICAL
PUZZLES



A Short Account of the History of Mathematics by W.W. Rouse Ball (Dover, 522pp). One of the clearest, most authoritative surveys from the Egyptians and Phoenicians thru 19th-century figures such as Grassman, Galois, Riemann. (DOV-20630-0 \$9.95)

Mathematics in the Time of the Pharaohs by Richard Gillings (Dover, 286pp). First book-length study—from simple commercial computations to trigonometric functions used in construction of pyramids. Fascinating, provocative. (DOV-24315-X \$6.95)

Perplexing Puzzles and Tantalizing Teasers by Martin Gardner (Dover, 256pp). 93 riddles, mazes, illusions, tricky questions, word and picture puzzles, other entertainments for youngsters. Many hilarious drawings. Solutions. (DOV-25637-5 \$3.95)

Entertaining Mathematical Puzzles by Martin Gardner (Dover, 112pp). Entertaining collection includes stimulating puzzles involving arithmetic, money, speed, plane and solid geometry, topology, more. Solutions. (DOV-25211-6 \$2.95)

Mathematics, Magic and Mystery by Martin Gardner (Dover, 176pp). Math behind card tricks, stage mind reading, coin and match tricks, etc. Plus more than 400 tricks, guaranteed to work. (DOV-20335-2 \$3.95)

Ingenious Mathematical Problems and Methods by Louis A. Graham (Dover, 237pp). Sophisticated material from Graham's *Dial*, applied and pure; stresses solution methods. Logic, number theory, networks, inversions, etc. (DOV-20545-2 \$5.95)

The Surprise Attack in Mathematical Problems by Louis A. Graham. Second volume from *Dial*. Difficult, sophisticated, challenging situations, stressing optimal approaches. Applied and pure. (DOV-21846-5 \$5.95)

Mathematical Brain Benders by S. Barr (Dover, 224pp). "Another collection of devilish problems, everyone original."—Martin Gardner. Over 100 fresh paradoxes, word and number games with wit, humor. Answers. (DOV-24260-9 \$4.95)

Mathematical Puzzles of Sam Loyd edited by Martin Gardner (Dover, 167pp). Bizarre, original, whimsical puzzles by America's greatest puzzler. Elementary math. (DOV-20498-7 \$3.95)

More Mathematical Puzzles of Sam Loyd edited by Martin Gardner (Dover, 177pp). 166 more problems from *Cyclopedia*. Arithmetic, algebra, speed and distance problems, game theory, more. (DOV-20709-9 \$4.50)

The Master Book of Mathematical Recreations by Fred Schuh (Dover, 430pp). Possibly the finest book work ever prepared on mathematical puzzles, stunts, recreations. (DOV-22134-2 \$6.95)

Test Your Logic by George J. Summers (Dover, 100pp). 50 more truly new puzzles with new turns of thought, new subtleties of inference. (DOV-22877-0 \$2.95)

Codes, Ciphers and Secret Writings by Martin Gardner (Dover, 96pp). Cipher and decipher codes: transportation, polyalphabetic, famous codes, typewriter and telephone codes, much more to challenge minds. (DOV-247661-9 \$2.95)

Different Worlds Publications

2813 - 19th Street
San Francisco, CA 94110

Address Correction Requested!

BULK RATE
U.S. Postage
PAID
San Francisco, CA
Permit No. 11798



MICHAEL K ERICKSON # 14
COMMUNICATIONS ASSOCIATES
BOX 250
MONTE RIO, CA 95462-0250

Different Worlds

2814-19th Street
San Francisco CA 94110

[415] 282-0999

BACK ISSUES OF
"YOUR BASIC BACKPACK"
FROM THE COMPUTER SHOPPER

IF YOU can read a comic book or a newspaper, you can learn to read and understand programs in BASIC. "Your BASIC Backpack," the popular column in the World's Biggest Computer Magazine, The Computer Shopper, provides tutorials for beginners who want to learn how to use Microsoft BASIC and QuickBASIC.



"Your BASIC Backpack" also poses problems for you to think about, play with, and solve.

Copies of past issues of "Your BASIC Backpack" are available as follows:

Number 1	Sep '87	10pp	\$2.00
Number 2	Oct '87	14pp	\$2.50
Number 3	Nov '87	14pp	\$2.50
Number 4	Dec '87	14pp	\$2.50
Number 5	Jan '88	18pp	\$3.00
Number 6	Feb '88	20pp	\$3.00
Number 7	Mar '88	23pp	\$3.00
Number 8	Apr '88	22pp	\$3.00
Number 9	May '88	23pp	\$3.00
Number 10	Jun '88	24pp	\$3.00
Number 11	Jul '88	20pp	\$3.00
Number 12	Aug '88	22pp	\$3.00
Number 13	Sep '88	16pp	\$3.00
Number 14	Oct '88	16pp	\$3.00

Prices include paper, toner, wear & tear on our trusty Canon copier, our time spent over hot copy machine, envelope, and first-class postage.

These are full-size (8-1/2" x 11") copies, not the reduced print published in The Computer Shopper. Perfect for all of you who don't have hyperscopic eyes.

Please order from Different Worlds Publications, 2814 - 19th Street, San Francisco, CA 94110.

ASK DR. JOHN

by John Heilborn

BASIC Difficulties

Dear Dr. John:

I enjoy reading your articles in *Computer Shopper*. I am 41 years young and have been programming in BASIC for about six years. I first learned to program on my Texas Instrument TI-99/4A, then progressed to my Atari 800XL. I now have an MS-DOS computer and need your help with several programming questions.

Problem #1: Input. For example:
10 CLS

20 LOCATE 10,2:INPUT "",K\$
30 LOCATE 20,2:PRINT K\$
40 LOCATE 10,2:INPUT K\$
50 LOCATE 21,2:PRINT K\$

When I run this program and input "CAT," then K\$="CAT" will be displayed on row 30. Now at the second input, enter "R" and press the Enter key to change from CAT to RAT. Now K\$="R." My Atari would have accepted the second input as RAT even though you pressed the Enter key after the letter "R." The only way I can do this with my MS-DOS computer is to either enter the word completely or to move the cursor to the end of the word. Will I have to write a subroutine to get each character as it is entered, then place it into a string?

Problem #2: Explain how to program the function keys. For example:

If F1 is pressed then...(go to subroutine)

If F2 " ", etc., etc.

Problem #3: How can I differentiate the numeric pad from the cursor keys? Using the ASCII codes did not help.

Problem #4: How can I read the directory from BASIC so I can display it as it appears at the DOS level? I do not like the BASIC FILES command.

Leonard M. Briones
Baton Rouge, La.

Dear Leonard:

I am assuming that you are referring to GW-BASIC in these questions. Although most interpreted BASICs are quite similar, some of these questions would be answered

differently for different BASICs such as QuickBASIC, which, by the way, can operate in either the compiled or interpreted mode.

NOTE: Computers do not really understand BASIC, or any other high-level languages. So in order to

computer store a whole word consisting of the new first character and all of the previous ones you might try this approach:

10 CLS
20 LOCATE 10,2:INPUT "",K\$
30 LOCATE 20,2:PRINT K\$

Computers do not really understand BASIC, or any other high-level languages.

run the programs we write in those languages, they must translate each command into machine code (something they *do* understand). And this must be done before they can perform the tasks outlined in our programs. One way of translating the instructions (the programs we write) is by using an interpreter like the one that is a part of GW-BASIC.

In interpreted mode, BASIC commands are "interpreted" as they are encountered, while the program is running. As a result, the program must be translated each time it is run, making interpreted languages a bit slower than programs that are written in something the computer can understand directly. Compiled languages like QuickBASIC or QuickC are compiled into something the computer can understand the first time they are written. As a result, when they are run, they can operate much more quickly than interpreted languages. The disadvantage of this is that the computer is running one thing and you are writing another, which can make trouble-shooting a bit more complex. However, these issues are better dealt with in another column.

So, keeping in mind that there are always lots of solutions to every problem in programming, here are the answers to your questions:

#1: First of all, you're right, GW-BASIC for the IBM computers will not read the screen characters unless the cursor has passed over them or you have entered them yourself. However, if your goal is to be able to enter a single character and have the

35 K1\$ = RIGHT\$(K\$, (LEN(K\$)-1))
40 LOCATE 10,2:INPUT "",K\$
45 K\$=K\$+K1\$
50 LOCATE 21,2:PRINT K\$

By the way, this is actually your program with just three changes. First, I changed line 40. In your version, the computer inserts a space and question mark over your previous entry because you omitted the double quotes before your variable (K\$). Incidentally, this alternatively could have been solved by removing the double quotes from 20 if you'd prefer the question mark as a prompt for each entry.

The second change I made was to store all of the data entered into K\$ (to the right of the first character) in a second variable, K1\$. This allows us to reuse K\$ in line 40 without losing the information we got in line 20.

Now, since I used a couple of new commands in this line, let me explain exactly how this works. The RIGHT\$ command returns the selected rightmost characters of a string. For example, if you entered the command:

PRINT RIGHT\$("BANANA",4)

the computer would respond by printing: NANA, the four rightmost characters of BANANA. Now in this instance, I used a specific word, and if you were always entering the same word, you could simply put that word in the RIGHT\$ statement and the computer would return the right thing every time. But since you're using a variable and might have any word in that string, we'd want to use a variable in place of "BANANA" like K\$ perhaps.

Secondly, if all of the words that you'd be entering in your program were of the same length, you could simply enter the number of characters in the variables (minus 1, since you want to replace the first character) and you'd be done. For example, if K\$ was "BANANA" then:

PRINT RIGHT\$(K\$,5)

would return the string: ANANA. But since your entries might be any length, we use LEN to obtain the length of the variable that was entered and subtract 1 from it, giving us all but one character (the leftmost one) from the string.

PRINT RIGHT\$(K\$, (LEN(K\$)-1))

The last (third) change I made was to add a line that put your newly entered character together with your old string so line 50 can print it:

45 K\$=K\$+K1\$

#2: Your question on how GW-BASIC accesses the function keys (F1-F10) is answered by looking at the operation of the ON KEY command. To get an idea of how this works, here's a sample program (with details following):

10 CLS

20 PRINT "This program demonstrates the use of the function keys (F1-F10)"

30 PRINT "to test the function keys, press any key from F1 to F10"

40 PRINT

50 PRINT "NOTE: F11 and F12 are not supported by GW-BASIC."

60 KEY(1) ON

70 KEY(2) ON

80 KEY(3) ON

90 KEY(4) ON

100 KEY(5) ON

110 KEY(6) ON

120 KEY(7) ON

130 KEY(8) ON

140 KEY(9) ON

150 KEY(10) ON

160 ON KEY(1) GOSUB 290

170 ON KEY(2) GOSUB 300

180 ON KEY(3) GOSUB 310

190 ON KEY(4) GOSUB 320

200 ON KEY(5) GOSUB 330

210 ON KEY(6) GOSUB 340

220 ON KEY(7) GOSUB 350

230 ON KEY(8) GOSUB 360

240 ON KEY(9) GOSUB 370

250 ON KEY(10) GOSUB 380

260 A\$=INKEY\$: IF A\$="x" OR A\$="X" THEN END

◆ ASK DR. JOHN

```
270 GOTO 170
280 CLS:PRINT "YOU PRESSED
THE F1 KEY,":PRINT:PRINT "to try
another function key":PRINT "press
another, to exit, press the x
key.":RETURN
```

```
290 CLS:PRINT "YOU PRESSED
THE F2 KEY,":PRINT:PRINT "to try
another function key":PRINT "press
another, to exit, press the x
key.":RETURN
```

```
300 CLS:PRINT "YOU PRESSED
THE F3 KEY,":PRINT:PRINT "to try
another function key":PRINT "press
another, to exit, press the x
key.":RETURN
```

```
310 CLS:PRINT "YOU PRESSED
THE F4 KEY,":PRINT:PRINT "to try
another function key":PRINT "press
another, to exit, press the x
key.":RETURN
```

```
320 CLS:PRINT "YOU PRESSED
THE F5 KEY,":PRINT:PRINT "to try
another function key":PRINT "press
another, to exit, press the x
key.":RETURN
```

```
330 CLS:PRINT "YOU PRESSED
THE F6 KEY,":PRINT:PRINT "to try
another function key":PRINT "press
another, to exit, press the x
key.":RETURN
```

```
340 CLS:PRINT "YOU PRESSED
THE F7 KEY,":PRINT:PRINT "to try
another function key":PRINT "press
another, to exit, press the x
key.":RETURN
```

```
350 CLS:PRINT "YOU PRESSED
THE F8 KEY,":PRINT:PRINT "to try
another function key":PRINT "press
another, to exit, press the x
key.":RETURN
```

```
360 CLS:PRINT "YOU PRESSED
THE F9 KEY,":PRINT:PRINT "to try
another function key":PRINT "press
another, to exit, press the x
key.":RETURN
```

```
370 CLS:PRINT "YOU PRESSED
THE F10 KEY,":PRINT:PRINT "to try
another function key":PRINT "press
another, to exit, press the x
key.":RETURN
```

Now, at first glance, the program above may seem a bit complex, but it's actually pretty simple. Line 10, for example, just clears the screen. And lines 20-50 print an introductory message that tells the user what the program does.

The program is based on a principle called trapping. You see, the function keys don't operate exactly

the same way that other keys do. Instead of generating an ASCII code when they're pressed, you need to trap them for your program to use them. This makes using the function keys a two-step process. First, you need to enable key trapping by

```
70 LOCATE X,Y:PRINT "#";
80 KEY(11) ON
90 KEY(12) ON
100 KEY(13) ON
110 KEY(14) ON
120 ON KEY(11) GOSUB 240
130 ON KEY(12) GOSUB 200
```

Cursor keys are trapped just the same way as the function keys. The main difference is trap numbers.

turning it on and then you need to detect the trap and tell the program what to do.

Lines 60-150 turn on the key trapping (which must be re-turned on after each trap). The syntax of this process is:

```
KEY(n) ON
in which "n" is the number of the
function key you want to detect—1=
F1, 2=F2, and so on. So lines 60-150
turn on key trapping for all of the
function keys.
```

Lines 160-250 check for function key entries and direct the program to a subroutine that corresponds to each different key (lines 280-370 are these lines). And finally, line 270 simply tells the program to keep looking if it hasn't found a key yet.

#3: The answer to this question is almost identical to the one above. The cursor keys are trapped just the same way as the function keys. The main difference is trap numbers. While the function keys are numbers 1 through 10, the cursor keys are: 11 (for cursor up), 12 (for cursor left), 13 (for cursor right), and 14 (for cursor down). To give you an idea how you might use the cursor keys in a program, here's another program example:

```
10 CLS
20 X=14:Y=40
30 PRINT "This program demon-
strates the use of the cursor keys."
40 PRINT "To move the pound sign
around on the screen,"
50 PRINT "press any of the four
arrow keys..."
60 PRINT "TO END THE
DEMONSTRATION, PRESS THE 'X'
KEY"
```

```
140 ON KEY(13) GOSUB 220
150 ON KEY(14) GOSUB 180
160 A$=INKEY$: IF A$="X" OR
A$=" " THEN END
170 GOTO 120
180 KEY(14) ON: LOCATE X,Y:
PRINT " ":X=X+1: IF X>24 THEN
X=5
190 LOCATE X,Y: PRINT
"#":RETURN
200 KEY(12) ON: LOCATE X,Y:
PRINT " ":Y=Y-1: IF Y<1 THEN Y=80
210 LOCATE X,Y: PRINT
"#":RETURN
220 KEY(13) ON: LOCATE X,Y:
PRINT " ":Y=Y+1: IF Y>80 THEN
Y=1
230 LOCATE X,Y: PRINT
"#":RETURN
240 KEY(11) ON: LOCATE X,Y:
PRINT " ":X=X-1: IF X<5 THEN X=24
250 LOCATE X,Y: PRINT
"#":RETURN
```

In this example, as before, line 10 clears the screen. Then line 20 sets up the initial position of the screen object (in this case, the pound sign). Lines 30-60 print some information about the program and line 70 prints the pound sign at its initial position near the center of the screen.

Then, just like the previous program, lines 80-110 set up the cursor-trapping parameters. Lines 120-150 check for a pressed cursor key and send the program off to the appropriate subroutine (up, down, left, or right), depending upon which cursor key was pressed.

Line 160 tests for an X key (if the user wants to end the program) and 170 repeats the key-searching process. Finally, lines 180-250 actually

move the pound sign around. Note that if the pound moves off of the edge of the screen or into the text at the top or bottom of the screen, it wraps around to the opposite side.

#4: To read the directory the way that DOS does it, all you need to do is go out to DOS, read the directory, and put the information into a file. That way, when you return to BASIC, you can read the directory any way you want. Here's an example of how that might be done:

```
10 KEY OFF: CLS: X=0
20 SHELL "DIR > BASDIR"
30 OPEN "T",1,"BASDIR"
40 IF EOF(1) THEN 100
50 INPUT#1, A$
60 PRINT A$
70 X=X+1
80 IF X=23 THEN 110
90 GOTO 40
100 END
110 PRINT
120 X=0: PRINT " paused, press
any key to continue..."
130 A$=INKEY$: IF A$="" THEN
GOTO 130
140 GOTO 90
```

In this example, line 10 turns off the keys that are usually present at the bottom of the screen in GW-BASIC, clears the screen and sets the line counter (X) to 0.

Line 20 uses the SHELL command to execute a DOS command. In this case, the DOS command puts the contents of the directory into a file called BASDIR.

In line 30, the program returns to BASIC and opens the file called BASDIR. Line 40 checks to make sure that we haven't gone past the end of the file and lines 50 and 60 get the data from the file and print it.

Line 70 increments the line counter and line 80 tests the line count to see if we've got a full page yet. If not, then the program returns to line 40 to get more data. But if we have a full line, the program prints a pause message and waits for the user to press a key in line 130. And when a key is pressed, line 140 continues with the next page of directory listings.

Got a question about computers? Send your queries on any computer-related topic to: Ask Dr. John, P.O. Box 20102, Castro Valley, CA 94546.

ASK DR. JOHN

by John Heilborn

Using a DataVue 25

Dear Dr. John:

I have a used DataVue 25 computer and a Panasonic KX-1091i printer. They work just fine with programs such as EasyWriter, but will not print graphics with shareware products such as World Atlas or PC Picture. Additionally, it will not run Ninja or NCR Golf although it will run Microsoft Flight Simulator. I also don't get the RAM drive when I type the command as in the manual.

What I'd like to know is: where can I obtain meaningful information on this machine? Also, where can I get information on the Xerox 4020 Color Ink Jet printer that I just bought, as well? I am getting a little too old to not know what I am doing, but I can't resist the adventure of learning about computers.

I'd appreciate any answers or suggestions you may have.

John Martin
New York, N.Y.

Dear John:

The DataVue 25 is a fairly old computer, so you may find that a lot of the newer software will simply not be compatible with it. For example, the DataVue 25 has an LCD type display that emulates the older IBM CGA display.

However, instead of being a color display, it displays shades of gray. If the software you are running is designed for an EGA or monochrome display, it won't run on the DataVue 25.

As far as your RAM drive is concerned, that is controlled by a setup routine that is offered each time the computer boots up. To enable the RAM drive, you'll need to enter the setup routine and tell the computer how large you want the RAM drive to be and so on. After that, it should behave as listed in the manual.

Regarding your Panasonic 1091i printer, most printers will be able to print simple text from almost any word processor. However, when you try to run graphics on printers, you need to specify the printer much more carefully. If you have a

setup routine with your application(s), you'll need to select the correct printer from the options list(s) before it will work properly. This is so because although there is a text standard that nearly everyone follows—ASCII (though at one time there were more text standards, too)—there are no real graphics standards that are universally followed for printers. As a result, if your graphics program doesn't specifically support your printer, it may not print properly (or at all).

As I recall, though, the KX-1091i has an emulation mode that behaves like the IBM Proprinter. To set this up, you'll need to set the DIP switches in the KX-1091i.

To find the DIP switches from the front of the printer, look straight down into the open area that holds the printer ribbon and print head. Slide the print head over to one side and look straight down near the center of the printer. You will see a small, clear plastic sheet. Beneath this sheet is an 8-switch DIP switch. To put your printer in IBM Proprinter emulation mode, move switch No. 1 (the far left switch) into the ON position. Once you've done this, just install your software as you would for an IBM Proprinter, and you should be in business.

As far as getting "meaningful information on your machine" goes, I'd suggest trying the technical support folks at DataVue. Their number is (404) 564-5555. If you cannot get the information you want through them, then, as always, I'd recommend finding a good, local user's group in your area.

Finally, for information about your Xerox 4020, call Xerox Americare: 1-800-334-6200.

Large Type for the Visually Impaired

Dear Dr. John:

Let me begin by saying that I am visually impaired. The trouble is, even the large text size I type with (24 point) is becoming too small for me to read.

In an effort to create larger text on my computer and printer, I called Berkeley Softworks some time ago and asked about larger print sizes.

They suggested I get a copy of FontPack Plus. So I purchased the program and, although the resident fonts in that package were no larger than the ones in Geos 2, I thought I would make my own with GeoFont.

Poring over the screen with a magnifying glass for many hours, I was able to enlarge one of the fonts to 36 points, pixel by pixel. During the process, I ran and printed out the partial alphabet. Since everything seemed to be in order, I redoubled my efforts to finish. But when I had completed the font, Geos would not call it up.

I read the Geos 2 manual and discovered Geos will not support "megafonts," but it did not define that term. So I called Berkeley Softworks again, but they no longer have phone support. Unfortunately, I can't use their modem help, because I can't read the small text I get over the modem.

If there is a way for me to use the font I have made, please let me know. I have a C64 computer, a 1541, a 1764 REU, and an Okimate 10 printer.

John H. Fajen
Wisconsin

Dear John:

Actually, you're on the right track already. The trouble is, the C64 doesn't have enough memory to support a full 36-point character set. What's happening is that you are simply running out of memory and Geos is selecting a font that is small enough to fit in memory.

The solution to this problem is to break your 36-point font into two halves: uppercase and lowercase. That way, your font will fit into the 4K of space that the C64 allows for fonts. To do this, make a copy of the font and delete the uppercase characters from one set and delete the lowercase characters from the other.

Once this is done, you can write your documents in, for example, lowercase (initially) and add the uppercase as a different font.

DecisionMate V Operating System

Dear Dr. John:

I have a pair of NCR Deci-

sionMate V computers that I wish to utilize. The problem is, I can't find any software to run on the computers. I understand that the computers are no longer manufactured, but there should be someone who knows something about the systems—even if they are discontinued.

NCR's customer service has not been helpful. I have tried NCR DOS on the computers with no positive results. The computer responds with: A:MOUNT O.S. Disk <CR>.

Additionally, I cannot find an owner's manual. What can I do to get these computers running? Any comments you may make will be greatly appreciated.

Larry E. Hardy
Pennsylvania

Dear Larry:

Your computers will not run under NCR DOS. They require CP/M to be able to operate. Unfortunately, CP/M, like all operating systems, is customized for every computer it operates. So although you could get a generic copy of CP/M from Digital Research (the company that created it), you might not be any better off than you are now.

Technically, NCR should be able to supply you with a properly customized version of CP/M. However, as you discovered, it's a bit difficult to get through to them.

If you want to persist with NCR, let me suggest that you try Kim Warnock who is in charge of the PC Division of their public relations department. Her number is (513) 445-4732. Or you may want to try your luck with Digital Research. Their number is (408) 443-4200.

And finally, if all else fails (or sometime before that), you may want to contact the First Osborne Group (FOG). They are a national user's group for CP/M and MS-DOS computers. They have more information on old computers than almost anyone. You can reach FOG at (415) 755-2000.

Got a question about computers? Send your letters to John Heilborn, Ask Dr. John, P.O. Box 20102, Castro Valley, CA 94546.

BASIC for beginners

The Smalltown 500

Larry Cotton

Last month, I promised that I'd offer help in finding the average speed of each car in our Smalltown 500 race. To do that, we must rewrite the program slightly:

```
10 PRINT CHR$(147)
20 DIM S(4,5),SP(4)
```

Recall that the DIM statement reserves space in the computer's memory—in this case, for the speed data. The first array is two-dimensional (four cars by five laps) and will contain each car's individual lap speed. The second array is reserved for the four cars' five-lap speed totals. This will become clear in a minute.

For purposes of this discussion, we'll assume that all four cars survive five laps. We now need to set up a nested FOR-NEXT loop to read the speeds (which will be in DATA statements) into the computer's memory:

```
30 FOR C=1 TO 4
40 FOR L=1 TO 5
```

The speeds are read with the READ statement:

```
50 READ S(C,L)
```

Let's close the FOR-NEXT loops:

```
60 NEXT L:NEXT C
```

When the program is run, C starts as 1. While C is 1, L increments from 1 to 5. The L loop finishes. C increments to 2. L loops again five times, and so on until C is 4, at which time all 20 speeds have been read into the computer's memory.

Up to this point, our program looks very similar to last month's. But now we must calculate the average speed of each car. This could be done inside the above FOR-NEXT loops, but for clarity we'll create separate loops for the math calculations:

```
70 FOR C=1 TO 4
80 FOR L=1 TO 5
```

```
90 SP(C)=SP(C)+S(C,L)
100 NEXT L:NEXT C
```

Here's where the SP(C) array is used. At the end of all this looping, SP(1) will be the sum of the speeds of all five laps of car number 1, SP(2) will be the sum of all five laps of car number 2, and so on.

Average Speeds

We still haven't found the cars' average speeds. Let's do that now with still another FOR-NEXT loop:

```
110 FOR T=1 TO 4
120 PRINT "CAR" T "S AVERAGE
    SPEED ="SP(T)/5
130 NEXT T
```

We must, of course, have the cars' speed data to read:

```
200 DATA 108,110,122,120,117
210 DATA 118,114,116,114,110
220 DATA 120,123,119,124,125
230 DATA 100,112,115,117,119
```

As mentioned, the two sets of FOR-NEXT loops could be combined into one. Replace lines 60 and 70 with these, and remove lines 80-100:

```
60 SP(C)=SP(C)+S(C,L)
70 NEXT L: NEXT C
```

Another Approach

If all this has been slightly difficult to understand, let's go back and look at arrays in a slightly different light.

Here's the most important concept: Any time you need to use your computer to deal with a number of related items, be they lap speeds in the Smalltown 500 or insects in a collection, array variables should be used to represent the data. That data can come from several sources: input from the user, DATA statements, and so on.

Last month we looked at one- and two-dimensional arrays, which serve most purposes quite well. But you should be aware that most versions of BASIC support arrays (at least theoretically) with a maximum

of 255 dimensions. The maximum number of elements allowed in each dimension is 32,767. Rarely, however, will you need arrays of more than 2 or 3 dimensions.

Here's an illustration which may help make the concept of arrays clearer:

```
10 PRINT CHR$(147)
20 ROW=5: COLUMN=7
30 DIM X(ROW, COLUMN)
40 X(3,4)=21
50 FOR J=1 TO ROW
60 FOR K=1 TO COLUMN
70 PRINT X(J,K); NEXT K
80 PRINT
90 NEXT J
```

If you enter and run this program, you'll see a graphic display (on your TV or monitor screen) of the contents of the 35 allocated memory locations—X(1,1) through X(5,7). All will be 0 except the one that was given a value of 21 in line 30. It will be printed in the third row of the fourth column.

Line 20 defines two constants, ROW and COLUMN, which become the size limits of our two-dimensional array. They can be changed to any values for which the computer has sufficient memory.

Borrowing an analogy from last month, we have a grid of five by seven pigeonholes. Line 30 dimensions the array of 35 elements. Line 40 assigns a value of 21 to one particular pigeonhole in the third row of the fourth column. Lines 50-90 contain nested FOR-NEXT loops which print the array as a 5 × 7 grid.

Numeric vs. String Arrays

This example uses numeric-variable arrays; the lack of the \$ character indicates that. As numeric variables, the values that are stored in the slots can be mathematically manipulated, as they were in our speed-averaging example.

But if you expect the computer to handle a lot of letters or names (not numbers), you must use a string-variable array, which is

denoted by the \$ character. Here's a modification of the above program which does just that:

```
10 PRINT CHR$(147)
20 ROW=2: COLUMN=13
30 DIM LTR$(ROW, COLUMN)
40 FOR J=1 TO ROW
50 FOR K=1 TO COLUMN
60 READ LTR$(ROW,COLUMN)
70 PRINT LTR$(ROW,COLUMN)";
80 NEXT K
90 PRINT
100 NEXT J
110 DATA A,B,C,D,E,F,G,H,I,J,K,L,M
120 DATA N,O,P,Q,R,S,T,U,V,W,X,Y,Z
```

Memory Requirements for Arrays

To conserve memory in long BASIC programs, you should dimension any arrays (single- or multidimensional) only to the maximum number of elements you expect the program to use. If the user will be entering data and you don't know how many entries to expect, you can ask him or her to furnish this number:

```
10 PRINT CHR$(147)
20 PRINT "DO YOU KNOW HOW
MANY": PRINT "ENTRIES YOU
WILL MAKE?"
30 GET RS:IF RS<>"Y" THEN IF
RS<>"N" THEN 30
40 IF RS="Y" THEN PRINT
CHR$(17);INPUT "HOW MANY";X:
DIMAS$(X): GOTO 60
50 DIMAS$(1000)
60 PRINT CHR$(17)"DIMENSIONED
TO"X"ELEMENTS
```

Run the program and try different responses to the questions. When the user knows how many entries will be made, A\$(X) will be automatically dimensioned to that size. (On a Commodore 64, the actual maximum number of elements this short program can be dimensioned to is 12,898.)

If the user types an N, this array will be dimensioned to 1000. You, the programmer, should choose a number that you know will be at least as great as—but, to avoid wasting memory, no greater than—the number of entries the user will make. To make sure that the computer has room for that number of entries, you need to know how much memory is available for the arrays and how much memory the array variables use.

To determine how much memory is free on a 64, type (in the immediate mode)

```
PRINT FRE(0)-(FRE(0)<0)*65536
```

On a Commodore 128, type

```
PRINT FRE(0)
```

to see the number of free bytes for BASIC programs. Or type

```
PRINT FRE(1)
```

to see the number of free bytes for BASIC variable storage.

FRE is a BASIC function that returns the number of available bytes in memory. It's usually used in immediate mode but can be used within a program. Sometimes the execution of FRE is very time-consuming.

Any variable (or constant) takes up a certain amount of the computer's memory, whether or not it's an array variable. The *Programmer's Reference Guide* for the 128 explains very clearly how much memory each type of array requires:

5 bytes for the array name
+ 2 bytes for each dimension
+ at least 2 bytes for each element

We haven't studied the type of variables that use the least amount of memory—integer variables. These simply represent whole numbers. Integer variables must be identified by a percent sign, such as A%(3). The DIM statement could look like this:

```
100 DIM A%(X)
```

X should be whatever number of elements you decide to use as the maximum.

If you identify the array variable without the percent sign, as in A(3), add three more bytes for each element. This is called a *floating-point variable* because the number it represents contains a decimal and as many as nine digits following it.

If you identify the array variable as a string, such as A\$(3), each element will require three bytes (not three additional bytes—just three bytes) plus one byte per character in each string element.

COMPUTE!'s Gazette is looking for utilities, games, applications, educational programs, and tutorial articles. If you've created a program that you think other readers might enjoy or find useful, send it, on tape or disk, to: **Submissions Reviewer, COMPUTE! Publications, P.O. Box 5406, Greensboro, NC 27403.** Please enclose an SASE if you wish to have the materials returned. Articles are reviewed within four weeks of submission.

FREE Catalog

Software, Accessories
& Leroy's Cheatsheets®



32 pages — filled with goodies
for your computer.

What are you waiting for? Clip and
mail the coupon, today, and start
shopping (and saving)!

Canadian residents — \$1.00 (U.S.) shipping

NAME _____

STREET _____

CITY _____

STATE _____ ZIP _____

CPI — Cheatsheet Products, Inc

P.O. Box 111368 Pgh, Pa. 15238

Dept. G10 412-781-6811



**GREENSBORO
COMPUTER
CENTER**

**AN AUTHORIZED
COMMODORE REPAIR CENTER**

**72-HOUR TURNAROUND
FOR MOST COMPUTERS**

C64	\$55.00
C128	\$87.50
1541	\$65.00
1541 and 1571	\$27.95
	Perm-alignment only
1571	\$75.00
A1000	\$45.00 hr.
	(plus parts)
CBM PRINTERS	\$45.00 hr.
	(plus parts)

FOR OTHER PRICING CALL!

Please enclose \$7.50 for return shipping.

All repairs come with a 30-day warranty and we guarantee the entire keyboard to work properly not just the repaired section. **POWER SUPPLIES ARE NOT INCLUDED IN THE ABOVE PRICING AND ARE PURCHASED SEPARATELY.**

If you have any questions about our services, please call me at 919-855-5792. Thank you.

1109 S. Chapman St.
Greensboro, NC 27403

simple answers to common questions

Tom R. Halfhill

Each month, *COMPUTE!'s Gazette* tackles some questions commonly asked by Commodore users. If you have a question you'd like to see answered here, send it to this column, c/o *COMPUTE!'s Gazette*, P.O. Box 5406, Greensboro, North Carolina 27403.

Q. I bought a BASIC compiler for my Commodore 64 to make my programs run faster. Why is it that the compiled programs are so much larger than the uncompiled programs?

A. To answer this question, we'll have to briefly review what a BASIC compiler is and how it works.

Normally, when you run a BASIC program on a Commodore 64 or 128, you're using the computer's built-in BASIC interpreter. An interpreter takes each individual instruction in a program and translates it into the corresponding machine language instructions that the computer really understands.

When you run a BASIC program, the interpreter does its job—translating BASIC statements one at a time. Note that even a seemingly simple BASIC instruction such as PRINT may translate into a fairly large number of machine language instructions. Due to these two factors, BASIC interpreters run programs at a relatively slow speed.

Machine language programs, on the other hand, run at the computer's top speed. That's because the program is already written in the true language that the computer understands, so no interpretation or translation is necessary.

It would be great if all programs were written in machine language, but that just isn't practical. Machine language (a term that we use synonymously with *assembly language*, by the way) is more diffi-

cult to master than higher-level languages like BASIC, and machine language programs take longer to design, write, and debug. As with all labor-intensive tasks, sometimes the high quality of the results aren't judged to be worth the investment in time.

That's why compilers were invented. A compiler lets you write a program in a familiar high-level language like BASIC. When you have a debugged version of the program working, the compiler translates the program into machine language instructions.

Unlike an interpreter, however, a compiler does not carry out this translation "on the fly" as the program runs. Instead, it translates the BASIC instructions into machine language instructions just once, during a step known as *compilation*. The translated machine language instructions are then stored in a disk file that usually can be run like any other machine language program.

As you've noticed, though, this compiled program is much longer than the original BASIC program with which you started. It's also much longer than an equivalent program would be if written directly in machine language in the first place.

The main reason is that all of the machine instructions required to carry out a BASIC instruction such as PRINT must be included in the program when it's compiled. Every command you use in the BASIC program forces the compiler to add a whole series of machine language instructions to the final, compiled version.

In addition, the compiler must include many more instructions to handle such routine jobs as keeping track of variables, translating decimal numbers into binary, performing mathematical computations, and so forth. Most compilers automatically include all of the machine

instructions for executing these functions whether they're actually used in the program or not. This is referred to as *overhead*, and it explains why even a one-line program compiles into a file several kilobytes long.

An interpreted BASIC program doesn't need to include this overhead because it's built into BASIC itself. The machine language instructions for PRINT and all other BASIC commands are permanently stored in the computer's read-only memory (ROM) chips. When the computer encounters a PRINT command in a BASIC program, the BASIC interpreter jumps to the appropriate machine instructions in ROM that print a character on the screen.

To put things into perspective, you could consider the BASIC interpreter in ROM as the "overhead" for an interpreted BASIC program. The BASIC interpreter in a Commodore 64 occupies 10K of ROM; when you add this to the length of an interpreter BASIC program, it's more in line with the length of an equivalent compiled BASIC program.

In case you're also wondering why even a compiled BASIC program runs more slowly than a similar program written directly in machine language, it's because today's compilers aren't nearly as efficient as the competent machine language programs. If you were to examine the compiled code (with a *disassembler*), you'd find numerous examples of sloppy programming.

Much more efficient compilers (known as *optimizing compilers*) are available for larger computers. These compilers analyze and improve the code that they produce, resulting in smaller and faster programs. Unfortunately, it will probably take several years for advanced optimizing techniques to "trickle down" to compilers made for home computers like your 64.

Larry Cotton

Now that we've learned how to program the four BASIC math functions, let's find some ways to put our new abilities to practical use.

Geometry is a good start. Suppose you wanted to calculate the distance around certain figures, such as triangles, rectangles, squares, and circles. The distance around a plane (flat) figure is called the *perimeter*, except in the special case of the circle, where it's known as the *circumference*.

The accompanying illustrations show various geometric figures. As we write our programs, refer to these illustrations to see the logic behind the mathematic formulas we use.

Let's start with the triangle. We'll find its perimeter. Type in this program:

```
10 INPUT "LENGTH OF FIRST SIDE IN INCHES";X
20 INPUT "LENGTH OF SECOND SIDE IN INCHES";Y
30 INPUT "LENGTH OF THIRD SIDE IN INCHES";Z
40 P=X+Y+Z
50 PRINT
60 PRINT "THE PERIMETER OF THE TRIANGLE IS"
70 PRINT P"INCHES."
```

The three INPUT statements get the lengths of the three sides. Line 40 calculates the perimeter, line 50 prints a blank line, and line 60 prints the answer in sentence form.

Notice that the variable P in line 70 is not within the quotation marks. If it were, the letter P would be printed instead of the value that the variable P holds.

Suppose we want to calculate a rectangle's perimeter. Since there are four sides, but only two different lengths, we can use multiplication and addition:

```
10 INPUT "LENGTH OF RECTANGLE IN INCHES";L
20 INPUT "WIDTH OF RECTANGLE IN INCHES";W
30 P=2*L+2*W
```

```
40 PRINT
50 PRINT "THE PERIMETER OF THE RECTANGLE IS"
60 PRINT P"INCHES."
```

Last month we learned about My Dear Aunt Sally—the mnemonic phrase that reminds us that multiplication and division are performed before addition and subtraction. In line 30, variable L is multiplied by 2, W is multiplied by 2, and then the two results are added together and are assigned to the variable P. Note that line 30 could be replaced by this mathematical equivalent:

```
30 P=2*(L+W)
```

The parentheses keep My Dear Aunt Sally from multiplying L by 2 and then adding W. Parentheses are the only way to short-circuit My Dear Aunt Sally.

Here's a program to calculate the perimeter of a square. Since all four sides of a square are of equal length, we can simply multiply one side by 4.

```
10 INPUT "LENGTH OF SQUARE'S SIDE IN INCHES";S
20 P=4*S
30 PRINT
40 PRINT "THE PERIMETER OF THE SQUARE IS"
50 PRINT P"INCHES."
```

The Ever-Popular Pi

Calculating the value of circle's perimeter is a little trickier. We can envision a triangle's, a rectangle's, or a square's sides and logically arrive at the correct mathematical operations to total their lengths. But for a circle we'll need this formula:

$$\text{Circumference} = \pi \times \text{Diameter}$$

Pi (or π) is a constant used in problems which involve circles. You can see the value of pi by entering this line and pressing RETURN:

```
PRINT  $\pi$ 
```

This never-ending decimal number is a subject unto itself, so for now, just think of pi as the con-

stant 3.14. The diameter of a circle is its width through the center. Enter this program:

```
10 INPUT "CIRCLE'S DIAMETER IN INCHES";D
20 C= $\pi$ *D
30 PRINT
40 PRINT "THE CIRCLE'S CIRCUMFERENCE IS" C "INCHES."
```

Your answer will be about nine digits long with a decimal. For this month, let's leave it that way; we'll save rounding—the shortening of a number to fewer decimal places—for next month.

Calculating Areas

The areas of plane figures are expressed in square units, such as square inches. The simplest formula for calculating the area of a triangle uses the length of the triangle's base (B) and its height (H). Refer to the accompanying illustration. The formula is:

$$A = B \times H / 2$$

Here's one possible program to calculate a triangle's area:

```
10 PRINT "ALL MEASUREMENTS ARE IN INCHES."
20 PRINT
30 INPUT "WHAT IS THE TRIANGLE'S HEIGHT";H
40 INPUT "WHAT IS THE TRIANGLE'S BASE";B
50 A=B*H/2
60 PRINT
70 PRINT "THE TRIANGLE'S AREA IS"
80 PRINT A"SQ. IN."
```

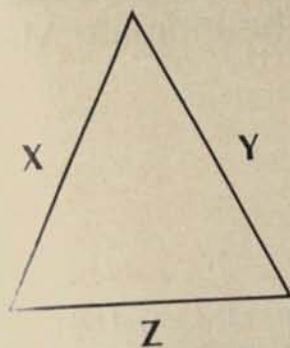
The other formulas for areas are somewhat easier. For a rectangle, one side is multiplied by the other:

$$A = L \times W$$

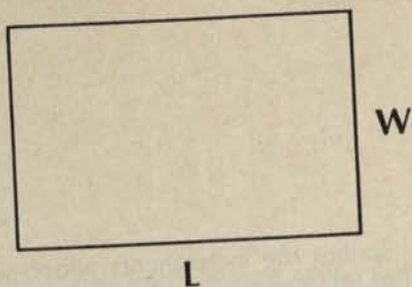
The program:

```
10 INPUT "LENGTH OF RECTANGLE IN INCHES";L
20 INPUT "WIDTH OF RECTANGLE IN INCHES";W
30 A=L*W
40 PRINT
50 PRINT "THE AREA OF THE RECTANGLE IS"
60 PRINT A"SQ. IN."
```

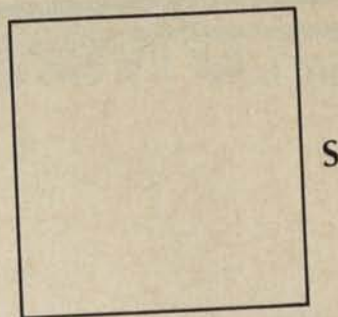
Now we return to the square.



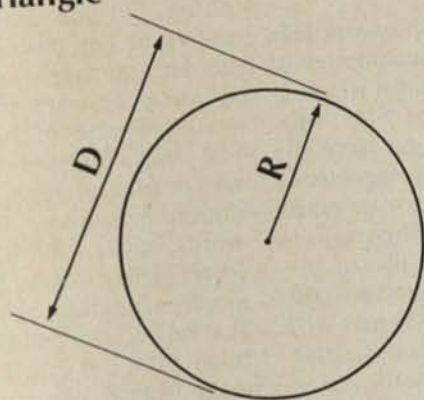
Triangle



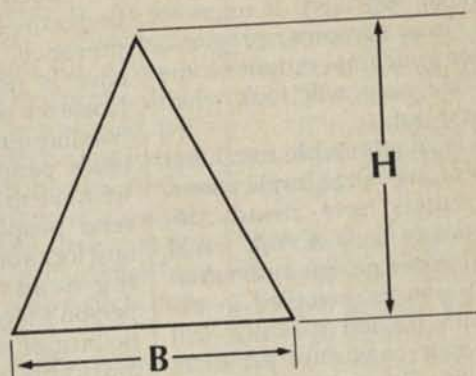
Rectangle



Square



Circle



Triangle

Now we return to the square. Here's one way it can be calculated:

```
10 INPUT "LENGTH OF SQUARE'S
   SIDE IN INCHES";S
20 A=S*S
30 PRINT
40 PRINT "SQUARE'S AREA IS" A "SQ.
   IN."
```

Numbers and Powers

Variable S times variable S can also be expressed as S^2 , which is called "raising S to a power of two" or simply "S-squared." S^2 on a computer is entered by typing S^2 . The ^ is (at least on Commodore computers) coincidentally on the same key that π is on. The 2 is the number of times S is multiplied by itself. Try this:

```
S=5:PRINT S^2
```

Enter this in the immediate mode and press RETURN. You should see 25. Try making S equal to other numbers. You always see the "square" of S (S multiplied by itself) as the answer.

The final exercise for this month will be to find the area of a circle. For this we need to know the circle's radius, which is half its diameter. The formula for a circle's

area is

$$A = \pi \times R^2$$

Here we use both π and ^. The formula in words is: The area equals pi times R-squared or simply pi R-square. We are multiplying π (the constant equal to about 3.14) times the radius multiplied by itself. Here's the program:

```
10 INPUT "CIRCLE'S DIAMETER IN
   INCHES";D
20 R=D/2:REM RADIUS IS HALF THE
   DIAMETER
30 A=PI*R^2
40 PRINT
50 PRINT "THE CIRCLE'S AREA
   IS" A "SQ. IN."
```

My Dear Aunt Sally doesn't address raising numbers to a power. Numbers are raised to powers before any multiplication, division, addition, or subtraction takes place. If that were not true, line 30 would have to look like this:

```
30 A=PI*(R^2)
```

The parentheses then would guarantee that the radius is multiplied by itself before the result is multiplied by pi.

That's our mathematical work-

out for this month. We should now be familiar with adding, subtracting, multiplying, dividing, using parentheses, and squaring numbers.

Don't be discouraged if all this has been a bit difficult to absorb in one sitting. As I've said before, the only way to learn anything well is to practice—so spend a little time playing with these exercises, entering various values at the input prompts. Next month we'll take a look at rounding.

Use the handy
Reader Service Card
in the back of the
magazine to receive
additional information
on our advertisers.

Fred D'Ignazio
Contributing Editor

A term that is growing in popularity these days is WYSIWYG (What You See Is What You Get). It refers to the way newer computer programs let you see your final output on the screen—just as it will look when you print it out.

This is an admirable trend. But think of its long-range implications. Futurist writers have already described advanced CAD/CAM (computer-aided design and manufacturing) systems installed in every person's basement which will fabricate new consumer products on demand. For example, if you want a new pair of shoes, just design them on your computer and "print" them out.

The concept of WYSIWYG has already reached an astounding stage in advanced laboratories. A newspaper recently reported on a new compact disc (CD ROM) drive in which the search time for the disk had been reduced dramatically by replacing the physical lens, which had to be moved mechanically, with a laser-simulated lens. Think of it. A real-world object—a lens—was created out of nothing but pure light. Something from nothing. WYSIWYG!

Multimedia Hackers

As computers become more intimate and personalized, the concept of WYSIWYG may extend to how we think about machines. When we look at a computer in the future, what will we see? What will we get?

I'm reading a great book which I recommend to anyone interested in personal computers of the future. It's called *The Media Lab*, and it was written by Stewart Brand, the author of *The Last Whole Earth Catalog*. (*The Media Lab*, from Viking Press, came out in late 1987, and should be out soon in paperback.) It de-

scribes the experiments underway at MIT's prestigious Media Lab by a group of ingenious, multimedia "hackers."

Much of the group's work falls under the heading "transmission of presence." Transmission of presence is reminiscent of Star Trek. However, since we don't have the Starship Enterprise's transporter to beam people from place to place, we have to figure out other ways to send people electronically to distant locations. One method is "talking heads." A TV signal of a person's face is beamed onto a plastic bust of a human's head. It's remarkable how lifelike the bust becomes with the TV picture superimposed onto its generic features—almost like having the person in the room with you.

Look into My Eyes

Another goal of MIT's researchers is to make technology more personal and more intimate. They have developed joysticks that fight back in a videogame; touch screens which let you "feel" data; cartoons with intelligent characters (sharks, skeletons, and worms); playful, cuddly robot blimps, chairs, and stuffed animals which interact with children; and computers that read lips and track eye movements so they can tell where you're looking on the screen.

Brand described an eerie experiment in which the intelligent character in a computer cartoon turned and faced him while he was staring at the computer screen. The character looked Brand directly in the eye. If this character had had the ability to gauge where Brand was looking, it would have known Brand was looking into its eyes.

Brand describes the experience as almost hypnotic and a little scary. The day is not far off when we'll come eye to eye with a computer. Will this be WYSIWYG?

What will we see? What will we think we see?

You can get intimate with computers, but you can also use computers to get intimate with other people—perhaps unintentionally. If you ever want to get personal with a member of the opposite sex, just chat with them for a few minutes in computerese. Have you ever noticed how many computer buzzwords have a kind of TV dating game feeling about them? For example, *baud* describes the transmission rate of data from one computer to another, but it sounds to the average listener like you are describing the computer ("bod") as a hunk or a "number 10." Or else, even worse, it sounds as if you are talking about a computer with an off-color, risqué sense of humor (a computer "bawd").

And we chatter mindlessly about computers, printers, monitors, and so on, as being *compatible* or *incompatible*. Again, the computer dating game. Just think how this sounds to other people.

Careful with Those Semantics!

An example of this blindness to our own lingo happened recently when I made a presentation to elementary school teachers. I talked for an hour about mating male and female cables with lots of vivid examples of plugging cables together.

Suddenly I noticed the blushes on several teachers' faces, and I realized how I sounded. Mating incompatible machines using male and female connectors so they'll share the same baud sounds more like a talk on sex education than a lecture on high tech.

To all you computer jocks out there, my advice is, when talking to noncomputerists, mind your manners and watch what you say. You may think you're talking high tech, but to your audience you sound like Dr. Ruth.

Todd Heimarck
Contributing Editor

A biological virus is a germ that enters your system, replicates, and makes you sick. An influenza virus gives you the flu, for example. Before you're actually ill, you may not know that you've got a bug; you might unwittingly spread it to others.

A computer virus acts similarly. It's a program that gets into a computer, spreads contagiously by making copies of itself (usually before anyone guesses that the computer has been infected), and eventually does something nasty.

One of the first examples of a computer virus is a key element in the book *Shockwave Rider* by John Brunner. Written before the advent of personal computers, the book presents a society that uses a huge supercomputer hooked up to millions of dumb terminals located around the country. (A dumb terminal isn't a real computer. It only works if it's connected to a remote computer—sort of like having a 64 that only works in conjunction with QuantumLink.)

The hero of the novel is a genius who controls his very own computer virus. Whenever he wants to change his identity, he activates the program. It creates the new identity and erases all records of the old one.

Trojan Horses

Viruses are sometimes called *Trojan Horses* because computer users willingly invite them into their computers only to find something unpleasant inside.

The contagious program may be downloaded from a bulletin board system, borrowed from a friend, or obtained at a user group meeting. Perhaps it prints a calendar, calculates mortgage payments, or plays tic-tac-toe. On the surface, it looks like an innocent program.

But it contains an active virus.

When you exit the original program, the virus remains in memory. Without resetting your computer, you continue using it. At some point, you look at a directory or load or save a file. During disk access, the virus checks the disk's boot sector for a copy of itself. If it's not there, the virus copies itself to the disk. If the virus does exist on disk, it might decrement a counter. Whenever you boot from that disk in the future, the virus copies itself into memory. If you switch disks, the virus spreads.

There's more. The virus's internal counter counts down until it hits 0. It might wait for 10 or 250 disk accesses before going into action. At that point, it formats the disk in the drive or scrambles your data files. The screen then flashes a message like *Ha Ha. Gotcha.*

The 64's Natural Immunity

Most computers load the disk operating system (DOS) into memory from a disk. A DOS is a program that knows how to move around the disk, reading or writing disk sectors. It also protects sectors in use and frees them up when you scratch a program. It takes care of updating the directory, formatting disks, and other disk-oriented jobs.

If the disk-based DOS is later upgraded, you simply get a new boot disk. The DOS disk is the place where viruses live. To infect such a disk, all you need is a single program that puts the virus in the boot sector that loads DOS. The virus then copies itself to any other disks that might come along.

The 64 and 128 have their operating system in read only memory (ROM). The DOS is built into the disk drive. The disadvantage to this approach is clear: To upgrade, you must install replacement ROM chips.

But there's also an advantage:

Viruses can't be installed on Commodore boot disks because the 64 doesn't use them. The DOS is already in the disk drive.

The 128 does make provision for booting from disk, but most 128 owners don't use boot disks for 64 or 128 mode.

It's possible to create a 128 virus, but it probably wouldn't spread very far.

Survival of the Fittest

Several years ago, *Scientific American* published an idea for a computer game called *Core Wars* (*core* is an old name for computer memory). The battlefield is a section of memory that wraps around from the highest byte to the lowest byte. The combatants include two or more computer programs that use a simple language, with instructions for branching, conditional branching, looping, math, copying a byte from one location to another, and so on. There is also a STOP command that halts a program.

The goal of the game is survival. You can pursue several interesting strategies. The all-out offensive program sprays STOPs throughout memory, attempting to hit the other program. Defensive tactics include building buffer zones of STOPs around the program's perimeters, and copying the program to another location and jumping there if the enemy gets too close.

You might discover that program A usually beats program B, but B beats C, and C beats A. You might attempt to write a program that adjusts its actions according to the opponent it's facing. However, the longer the program is, the more memory it uses, which makes it more vulnerable.

If you're interested in exploring viruses, don't write one that formats disks or scrambles data files. Instead, try inventing your own *Core Wars* language.

Douglas M. Blakeley

This new printer driver for Epson, Star, and compatible dot-matrix printers offers near-laser-printer-quality printing with both GEOS and GEOS128. A customizer is also included to allow you to fine-tune the driver.

If you have an Epson or compatible printer and you use GEOS or GEOS128, this new printer driver can give you near-laser-printer-quality printouts with print densities of 60, 72, 80, 120, 144, or 240 dots per inch (depending on your printer's capabilities). The driver comes with preinstalled codes for Epson FX-85/86e, Epson LX80/86, and Star SG-10/15; it also has an option that allows you to customize the driver for other Epson-family printers as well.

The printer driver program comes in two parts. "Driver" (Program 1), is the machine code for the printer driver. "Customizer" (Program 2), tailors the driver to a specific printer and converts the driver to a GEOS-format file. Program 2 also permits you to select the printer device number (4 or 5) and disable the paper-out sensor to permit single-sheet printing with *Writer's Workshop*.

Getting Started

Since Driver is written in machine language, you'll need to enter it with "MLX," the machine language entry program printed elsewhere in this issue. When you run MLX, you'll be asked for the starting and ending addresses of the data you'll be entering. Here are the values to use for Driver:

Starting address: 7804
Ending address: 7F33

Follow the MLX instructions carefully, and be sure to save a copy of the Driver data with the filename PR.OBJ before you leave MLX.

Customizer is written in BASIC, so simply type it in, save a copy on the same disk as Driver, and type RUN. Customizer sets the top of BASIC memory to 30720 to provide a safe work area and then loads PR.OBJ into memory addresses 30720 to 32557. Please note that, although the driver you create with customizer can be used with either GEOS or GEOS128, you must customize the driver on the 64 (or a 128 in 64 mode).

When you run Customizer, it asks you for your printer type, printer address (4 or 5), and whether you want the paper sensor disabled. After these questions are answered, Customizer patches the Driver's object code in memory and saves the customized Driver to disk. This Driver is then converted to a GEOS format file. The filename EPSON FH-85, EPSON LH-80, or STAR SG-10 is used depending on the printer you specified. The PR.OBJ file is not destroyed in this process, so if you make a mistake, you can start over.

Once the file has been converted, treat the disk just like a GEOS disk. Don't use the standard disk validate command; use the GEOS validate command instead. As a reminder that the printer driver is multidensity, the file icon is modified to include the letter M in the upper left corner.

Using the Printer Driver

Once the conversion program has been run, load the GEOS operating system and transfer Driver to a GEOS work disk. If the disk containing the printer driver has not been used under GEOS, you'll be asked if you want the disk converted. You should answer yes, or you won't be able to transfer the file with a single disk drive.

The new printer driver can be activated by selecting the GEOS menu in the upper left corner of the screen and choosing the Select

Printer option. After choosing to print a *geoWrite* or *geoPaint* document, a new dialog box will appear, allowing you to select the printer density. Choose the density you want by clicking once on the corresponding icon. The F icon selects the filled 240-dots-per-inch mode, while the 240 icon selects the enhanced mode.

Once the density has been chosen, the printer initializes to this format and prints your document. For those owning *Writer's Workshop*, the new driver's menu will appear after the initial print menu, which permits you to select starting and ending pages as well as high, draft, or NLQ modes. If draft or NLQ modes are selected, the second menu will still appear. In this case, select 80 dots per inch to continue printing.

When using printer densities of 72 and 144 dots per inch, *geoWrite* and *geoPaint* will make adjustments on the printed page width. *GeoWrite* will widen the text by two-thirds of an inch while maintaining the same number of characters per line as shown on the monitor, making up the difference by narrowing the margins. *GeoPaint* will not print the rightmost three-fourths of an inch of the graphic. For this reason, don't use this rightmost area when planning on using 72- or 144-dots-per-inch densities.

Customizing

For those with printers that are in the Epson or Gemini family but whose printer control codes differ, there is an option to customize your own printer driver. The use of this option requires careful consultation of your printer manual and should only be used once you understand the correct codes.

After this option is selected, you'll be asked for the codes to select certain features. For each question, the customizer will display the number of bytes it expects for that

Your BASIC Backpack #27

by Rob Albrecht & Don Inman

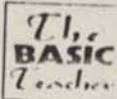
If you can read a comic book or a newspaper, you can learn to read and understand programs in BASIC. Well, you can if the programs are written so as to encourage humans to understand them. That's how we try to write programs in "Your BASIC Backpack."

You can learn to read and understand BASIC, express yourself in BASIC, and use it to make the computer do what you want it to do the way you want it done.

"Your BASIC Backpack" provides tutorials for beginners who want to learn how to use Microsoft QV-BASIC and QuickBASIC. We also pose problems for you to think about, play with, and solve. Rob & Don, P.O. Box 1635, Sebastopol, CA 95472.



***** The BASIC Teacher — a FREE Short Subscription



The BASIC Teacher is our magazine about learning and teaching Microsoft QV-BASIC and QuickBASIC. Every issue has "Teach Yourself QuickBASIC" and "Teach Yourself QV-BASIC" for beginners, and "Browsing BASIC" for people at the novice level. Readers of "Your BASIC Backpack" may have a FREE short subscription (3 issues). Send your request to Tadashi Hara, the BASIC Teacher, 2014 - 19th St., San Francisco, CA 94116. This offer expires November 10, 1989.

Teach Yourself QuickBASIC

This section began in Your BASIC Backpack #6, April 1988. So we now assume that you are acquainted with several of QB's windows, menus, and operation procedures. There are more than 200 keywords in QuickBASIC's vocabulary.

This time we will use only the following keywords:

AS	DIM	FOR	LEN	OPEN	PRINT
CHR\$()	DO	GET	LINE	PRINT	INPUT
CLIP	END	IF	LOCATE	PUT	VIEW
CLS	EOF	INPUT	LOF	RANDOM	WRITE
DEFINT	PRINT	INPUTS	LOOP	REM	



Random Access Files

Last time (YB #26) we began a slow tutorial on random-access files. We created and named Notepad.Ran, a random-access file of notes, any kind of notes. Notepad.Ran is a very simple file: each record is one string exactly 72 characters long. If you enter a record with fewer than 72 characters, QuickBASIC adds spaces to make it the right size: 72 characters. If you enter a record with more than 72 characters, QuickBASIC chops it to the right size: 72 characters.

As created last time, Notepad.Ran has 7 records with a total of 504 bytes (7 x 72 = 504). Here are the records we put into the file:

Record Number	Record
1	This is the Notepad.Ran file.
2	Notepad.Ran is a random-access file.
3	Each record is one string field with exactly 72 characters.
4	Use Notepad.Ran for notes of any kind.
5	Meet with Jester at noon on 4/1.
6	Library books due 4/15 — also mail tax return.
7	Reality expands to fulfill the available fantasies.

Last time we showed you program YB2601Q to create the file by entering records from the keyboard and program YB2602Q to read the file sequentially from the top (record #1) to the bottom (record #7).

In past episodes, you learned how to use the KEY command to display a sequential file on the screen. Let's see what happens if we use TYPE to display the Notepad.Ran file on the screen. Our Notepad.Ran file is on our QuickBASIC Work Disk in disk drive A. The KEY "A prompt" is on the screen, as follows:

A>

Therefore, drive A is now the default drive. So, use the hint:

Type:

type notepad.ran

and press ENTER

The computer displays the Notepad.Ran file like this:

Alttype notepad.ran	
This is the Notepad.Ran file.	
Ran is a random-access file.	
one string field with exactly 72 characters.	
of any kind.	
return	
on	

Notepad.
Each record is one
string field with
exactly 72 characters.
Use Notepad.Ran for
notes of any kind.
Meet with Jester at
noon on 4/1.
Library books due
4/15 — also mail tax
return.
Reality expands to
fulfill the available
fantasy.

The strange appearance of the Notepad.Ran file is because there are no end-of-record characters to cause a "carriage return" (CR) and line feed (LF). To the KEY TYPE command, the Notepad.Ran file looks like one long string of

ASCII characters. The TYPE command prints this long sequence of characters to the screen, 80 characters per line. Note that the first line contains the first record (72 characters), plus the first eight characters of the second record. The second line contains the rest of the second record (64 characters), and the first 16 characters of the third record. And so it goes to the end of the file.

Create the Notepad.Ran File with ROR Characters

When you create a sequential file using the WRITE# or PRINT# statements, the computer automatically adds end-of-record (ROR) characters, CR (ASCII code 13) and LF (ASCII code 10). However, in creating a random-access file, if you want ROR characters, you must include them as part of the record type definition. Program YB2701Q does this.

REM ** Create the Notepad.Ran File with End of Record Characters **
* Your BASIC Backpack #27, 8/31/1989
* Microsoft QuickBASIC. Filename: YB2701Q.BAS

REM ** Set up **
DEFINT A-Z
CLS

REM ** Define record structure **
' One record is one string with 72 characters
TYPE RecordType
 Note AS STRING * 72 'String field with 72 characters
 Eor AS STRING * 2 'End of record characters, CR & LF
END TYPE

REM ** Declare a variable of above type **
DIM Record AS RecordType
Record.Eor = CHR\$(13) + CHR\$(10) 'End of record characters

REM ** Open Notepad.Ran on default drive **
OPEN "Notepad.Ran" FOR RANDOM AS #1 LEN = 74

REM ** No records yet, so set record number to zero **
RecordNumber = 0

REM ** Enter records from keyboard, write to Notepad.Ran **
DO
 CLS
 PRINT "Press ESC to quit or another key to enter a record"
 PRINT "to the Notepad.Ran file. "; akey\$ = INPUT\$(1)
 IF akey\$ = CHR\$(27) THEN EXIT DO
 RecordNumber = RecordNumber + 1
 PRINT : PRINT "Please type record #"; RecordNumber; "and press ENTER."
 PRINT : LINE INPUT Record.Note
 PUT #1, RecordNumber, Record
LOOP

REM ** Print length of file, close file, and program **
PRINT
PRINT "Notepad.Ran file has"; LOF(1); "bytes."
CLOSE #1
END

A Sample Run

A run of program YB2701Q looks just like a run of program YB2601Q, shown last time. However, if you didn't read the last issue, here is a replay of a sample run. It begins like this.

Press ESC to quit or another key to enter a record
to the Notepad.Ran file.

Press a key other than ESC and next you see:

Press ESC to quit or another key to enter a record
to the Notepad.Ran file.

Please type record # 1 and press ENTER.

Type a record. Before pressing ENTER, the screen might look like this.

Press ESC to quit or another key to enter a record
to the Notepad.Ran file.

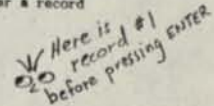
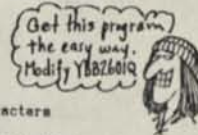
Please type record # 1 and press ENTER.

This is the Notepad.Ran file.

OK, that's the record we want, so we press ENTER. QB adds end-of-record characters, CR and LF, and writes the record to the file. Soon we see the following:

Press ESC to quit or another key to enter a record
to the Notepad.Ran file.

Record #1 has been entered. Press a key other than ESC to enter record #2
(the space bar is a good choice).



Press ESC to quit or another key to enter a record to the NotePad.Ran file.

Please type record # 2 and press ENTER.

Note that the computer is keeping track of the record number. It is waiting for record #2. Enter record #2 as follows. Before pressing ENTER, the screen looks like this.

Press ESC to quit or another key to enter a record to the NotePad.Ran file.

Please type record # 2 and press ENTER.

NotePad.Ran is a random-access file.

Press ENTER, then continue entering records up to and including record #7. After entering record #7, press ESC to quit. This finishes the creation of the NotePad.Ran file. The program computes the number of bytes in the file, displays it for your reading pleasure, and closes the file. Our screen looks like this:

Press ESC to quit or another key to enter a record to the NotePad.Ran file.

NotePad.Ran file has 518 bytes.

Note that the file has 518 bytes, 14 bytes more than the NotePad.Ran file created last time by program YBR2601Q. Each of the 7 records has 2 more bytes, the end-of-record characters, CR and LF. The file now contains 7 records. Each record has 72 characters plus 2 end-of-record characters.

7 records x 74 bytes/record = 518 bytes

View the NotePad.Ran File in the View Window.

You can load the NotePad.Ran file into QB's View Window and view it. However, since it is not a program, if you try to edit it, you will see a Syntax error dialog box. Of course, you can invoke the Full Menu and turn off the syntax checking. We assume you know how to change from QB 4.5's Easy Menu to Full Menu, and conversely. Is our assumption correct?

If you are using the QuickBASIC Interpreter, Windows Edition, you can load the NotePad.Ran file into the View Window more easily -- load it as a Document instead of as a program. We use the QB Interpreter for most of our BASIC brainstorming. Well, QB 4.5 or QB1, here is the NotePad.Ran file in the View Window.

```
File Edit View Search Run Debug Options Help
NotePad.Ran
This is the NotePad.Ran file.
NotePad.Ran is a random-access file.
Each record is one string field with exactly 72 characters.
Use NotePad.Ran for notes of any kind.
Meet with Jester at noon on 4/1.
Library books due 4/15 -- also mail tax return.
Reality expands to fulfill the available fantasies.
```

Use the DNE TYPE Command to Display the File.

Now that the NotePad.Ran file has end-of-record characters, it will be easier to read when displayed by the DNE TYPE command. Exit to DOS and try it. When we did it, our screen looked like this.

```
A)TYPE NotePad.Ran
This is the NotePad.Ran file.
NotePad.Ran is a random-access file.
Each record is one string field with exactly 72 characters.
Use NotePad.Ran for notes of any kind.
Meet with Jester at noon on 4/1.
Library books due 4/15 -- also mail tax return.
Reality expands to fulfill the available fantasies.
```

Scan the NotePad.Ran File

Last time we showed you program YBR2602Q to scan the NotePad.Ran file as it was created by program YBR2601Q. We modified that program so that it can scan the new version of NotePad.Ran, as created by program YBR2701Q. When you run the program, it begins with the following message.

The NotePad.Ran file has 7 records with a total of 518 bytes.

The bottom line tells you: Press a key to get a record.

Here is a complete sample run of program YBR2702Q.

```
The NotePad.Ran file has 7 records with a
total of 518 bytes.

This is the NotePad.Ran file.

NotePad.Ran is a random-access file.

Each record is one string field with exactly 72 characters.
```

Use NotePad.Ran for notes of any kind.

Meet with Jester at noon on 4/1.

Library books due 4/15 -- also mail tax return.

Reality expands to fulfill the available fantasies.

*** End of file ***

Press any key to continue

And here is program YBR2702Q. We got it by making a few changes in last time's program YBR2602Q. The changed (or new) lines are marked by arrows.

```
REM ** Scan the NotePad.Ran (with QB) file **
' Your BASIC Backpack #27, 8/31/1989
' Microsoft QuickBASIC, Filename: YBR2702Q.BAS

REM ** Set up **
DEFINT A-Z
CLS

REM ** Define record structure **
TYPE RecordType
    Note AS STRING * 72 'String field with 72 characters
    Eor AS STRING * 2 'End of record characters, CR & LF
END TYPE

REM ** Declare a variable of above type **
DIM Record AS RecordType
' Record.Eor = CHR$(13) + CHR$(10) 'End of record characters

REM ** Open NotePad.Ran on default drive **
OPEN "NotePad.Ran" FOR RANDOM AS #1 LEN = 74

REM ** Put an instruction in line 25 & define a view port **
LOCATE 25, 1: PRINT "Press a key to get a record";
VIEW PRINT 1 TO 23

REM ** Print number of bytes and number of records **
CLS 2: PRINT "The NotePad.Ran file has"; LOC(1) / 74; "records with a"
PRINT "total of"; LOC(1); "bytes."

REM ** Start at first record in file **
RecordNumber = 1

REM ** Get records from file and print to screen **
DO UNTIL EOF(1)
    akey = INPUT$(1)
    GET #1, RecordNumber, Record
    PRINT : PRINT Record.Note
    RecordNumber = RecordNumber + 1
LOOP

REM ** Close file, return screen to normal, and program **
PRINT "*** End of file ***"
CLOSE #1
VIEW PRINT
END
```

In both programs, YBR2701Q and YBR2702Q, the record structure is defined by the following program block:

```
REM ** Define record structure **
TYPE RecordType
    Note AS STRING * 72 'String field with 72 characters
    Eor AS STRING * 2 'End of record characters, CR & LF
END TYPE
```

This defines a record as having two fields. The first field (Note) has exactly 72 characters; the second field (Eor) has exactly two characters. Therefore, a record has exactly 74 characters. RecordType is the name of the record structure. We chose this name arbitrarily and capriciously. We could have used DonaldDuck or Frodo or WonderWoman or any name that conforms to the conventions for naming QuickBASIC variables.

Next, the following block declares Record as a variable of the type previously defined as RecordType. This automatically defines two variables called Record.Note and Record.Eor. A value is then assigned to Record.Eor.

```
REM ** Declare a variable of above type **
DIM Record AS RecordType
Record.Eor = CHR$(13) + CHR$(10) 'End of record characters
```

The value of Record.Eor is a 2-character string consisting of the end-of-record characters, CR & LF. Values of Record.Note are acquired later, when someone enters them from the keyboard. Now open the file. Note that the length of a record is 74 characters (LEN = 74).

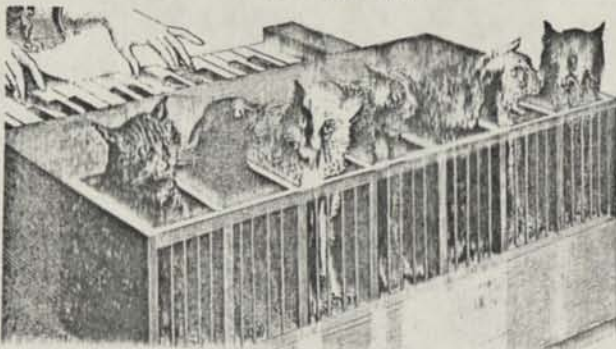
```
REM ** Open NotePad.Ran on default drive **
OPEN "NotePad.Ran" FOR RANDOM AS #1 LEN = 74
```

A better way to open the file is shown below:

```
OPEN "NotePad.Ran" FOR RANDOM AS #1 LEN = LEN(Record)
```

Why is this way better? Hint: Suppose you redesign the file structure so that a record is longer or shorter than 74 characters.

The little kids see mom and dad pushing keys and making things happen. The little kids want to push keys and make things happen, too. Write a program to make each key (well, almost each key) a little kid's delight. Press a key and you hear a sound and see something happen on the screen. Hmm... this is something Zappy Artist could do. Consult your back issues of "Your BASIC Backpack" for Zappy Artist's antics.



When little kids push keys and hear & see things happen, it probably won't be long before they figure out what key does what. How maybe they will push interesting sequences of keys and make interesting sequences of things happen.

For your first effort, you might use the ASCII code for a key to select a SOUND frequency from an array. Use the same ASCII code to compute a background color and border color for the screen. Sound the SOUND and display the colors for that key. Different keys - different sounds and colors.

Problem 73. Guess My Number for Younguns

Here is a Guess My Number game we play with quite young children, using paper and pencil or a blackboard. First, we write down the numbers from one to nine.

1 2 3 4 5 6 7 8 9

Then we secretly pick a number. Let's pick 7. The child guesses. Suppose she guesses 3. We put a "pointer" above her guess.

1 2 3 4 5 6 7 8 9

The pointer (3) points toward our secret number. Of course, it also means "greater than." So we say, "My number is greater than three."

Now suppose she guesses 9. We put a "less than" pointer above 9.

1 2 3 4 5 6 7 8 9

Aha! Perhaps she will see that my secret number is caught between 3 and 9. It is greater than 3 (>3) and less than 9 (<9). We continue until she guesses the secret number -- and put an equals sign (=) above her correct guess.

Guess number: 1 3 4 2
Hint: > = <
1 2 3 4 5 6 7 8 9

Write a program to play this game. Here is the first screen. The sentences are centered on the screen in big letters (WIDTH 40).

Guess my number game

Press any key to begin playing



When someone presses a key, pick a secret number and show the next screen.

I'm thinking of a number, 1 to 9

Guess my number

1 2 3 4 5 6 7 8 9

What is your guess?



After each guess, put an appropriate mark above the number the player guesses: < or > or =. If the player guesses the number, tell her she got it and use a little sound and color for pizzazz. The screen might look like this.

I'm thinking of a number, 1 to 9

Guess my number

1 2 3 4 5 6 7 8 9

You guessed my number!

To play again, press any key.

Blink, blink, blink... in changing colors

Designing a program like this is sometimes called "storyboarding." Use our storyboard to help you write this program, or design your own storyboard. Good luck, and keep up the good work.

Problem 74. Reaction Time Game

Okay, everybody, check your reaction time. Write a program to flash consecutive numbers on the screen. You press the space bar to stop the computer. The number on the screen tells you if you are fast or slow. It might begin like this:

How fast are you?

When I start counting,

Press the space bar to stop me.

Press any key and I'll begin.

I'm fast!

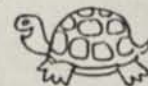


Press any key. The screen clears and for a few (random) moments, nothing happens. Then near the center of the screen, in a box, you see 1, then 2, then 3, then 4, then 5, ... and so on until you press the space bar. What number did you stop on? On a slow day, we saw this screen.

123

To play again, press a key

Not bad!



If you wish, also tell exactly how long it took from the time the computer started counting to the time someone pressed the space bar (use the TIMER function before and after).

Variation. Pick a number to stop on. Try to stop on that number or close to that number.

DragonSmoke

The PC-SIG Encyclopedia of ShareWare. This book describes and reviews the shareware programs in PC-SIG's enormous library. We counted 1127 disks. If it can be done, it can probably be done with low-cost shareware -- and you can find it in this book. From PC-SIG, 10300 E. Duane Ave., Sunnyvale, CA 94086. \$17.95.



QuickBASIC Made Easy by Bob Albrecht, Vanden Vlegard, & Dean Brown is a beginner's book on the latest QuickBASIC, version 4.5. No previous programming experience required. Workbooks and a convenience disk also available. For more information, contact The BASIC Teacher, 2814 - 19th St., San Francisco, Ca 94110.

If you are already beyond our beginners' book, try Using QuickBASIC by Don Inman & Bob Albrecht or Advanced QuickBASIC by Don Inman, et al.

GV-BASIC Made Easy by Bob Albrecht & Don Inman. This is a beginners' book on good old GV-BASIC. No previous programming experience required. An inexpensive convenience disk also available. For more information, contact The BASIC Teacher, 2814 - 19th St., San Francisco, CA 94110.



Most of the work is done by the following program block. It acquires an entire record (value of Record#) and displays the pertinent part of the record (value of RecordNote) on the screen. It goes round and round until it reaches the end of file.

```
REM ## Get records from file and print to screen ##
DO UNTIL EOF(1)
  key$ = INPUT$(1)
  GET #1, RecordNumber, Record
  PRINT #1 RecordNote
  RecordNumber = RecordNumber + 1
LOOP
```

Next time we'll continue with random-access files. What would you like to know about them?

Teach Yourself GW-BASIC

GW-BASIC is the generic form of Microsoft BASIC. If you have an MS-DOS computer, you probably have GW-BASIC. BASIC is similar to GW-BASIC, licensed to and sold by IBM. We use the Tandy version of GW-BASIC, called BASIC on the disk supplied with our Tandy 1000. GW-BASIC has nearly 200 keywords. This time, we'll use only a few, listed below.

ABS	IF	LOCATE	END	WIDTH
CHR\$	INPUT	OFF	SCREEN	
CLS	INPUT\$	PRINT	TAB	
DEFINT	INT	RANDOMIZE	THEN	
GOTO	KEY	REM	TIME\$	

These are
AOK in
QuickBASIC,
too



We show keywords in all upper case letters and variables in lower case or a mixture of upper and lower case letters. We do this to make programs easy for you to read and understand. HOWEVER, when you LIST a program, GW-BASIC lists variables in upper case.

Stars -- a Guessing Game

Stars first appeared in People's Computer Company, December, 1972. Last time, we posed Stars as Problem 71. Here is a primitive program you can use to play Stars. A run might go like this, in WIDTH 40 letters.

```
Welcome to my galaxy. Let's play a
game. I'll think of a number from 1
to 100. Guess my number. If you win,
I'll print some stars. The closer you
are, the more stars you will see.
If you see 7 stars (*****),
you are very, very close to my number!
```



```
Your guess? 10 *      Only one star. Not close!
Your guess? 20 ***     Well, a little closer.
Your guess? 30 ***     Not making much headway.
Your guess? 90 ***** That's better.
Your guess? 95 ***** Oups.
Your guess? 80 ***** OK!
Your guess? 81 ***** Ah! I think I know...
```

```
Your guess? 79
***** That's it!!!
```

Press a key to play again

What Do the Stars Mean?

Let D = distance of guess from secret number

Value of D	Number of Stars
64 or more	1
32 to 63	2
16 to 31	3
8 to 15	4
4 to 7	5
2 to 3	6
1	7

For you people who know a little math:

X = secret number

G = player's guess

$D = |X - G|$

Oh, in GW-BASIC, we'll use the following variables and the ABS function:

Secret is the secret number

Guess is the player's guess

Distance = ABS(Secret - Guess)

Program YBR2703G begins with the usual title block, followed by a setup block and instructions on how to play.

```
1 REM ## Stars -- a Guessing Game ##
2 ' Your BASIC Backpack #27, 8/31/89
3 ' Microsoft GW-BASIC. Filename: YBR2703G.BAS
```

```
100 REM ## Set up ##
110 SCREEN 0: CLS: KEY OFF: WIDTH 40
120 DEFINT A-Z
130 RANDOMIZE TIMER
```

```
200 REM ## Tell how to play ##
210 CLS
220 LOCATE 1, 1: PRINT "Welcome to my galaxy. Let's play a"
230 LOCATE 3, 1: PRINT "game. I'll think of a number from 1"
240 LOCATE 5, 1: PRINT "to 100. Guess my number. If you win,"
250 LOCATE 7, 1: PRINT "I'll print some stars. The closer you"
260 LOCATE 9, 1: PRINT "are, the more stars you will see."
270 LOCATE 11, 1: PRINT "If you see 7 stars (*****),"
280 LOCATE 13, 1: PRINT "you are very, very close to my number!"
```

At this point, we ran the program to see if everything is working. Fortunately, it was. So... here is the next block, wherein the computer "thinks" of a number, selected at random. The number is an integer in the range, 1 to 100. The secret number is called Secret.

```
300 REM ## Computer 'thinks' of a number ##
310 Secret = INT(100 * RND) + 1 'Random integer, 1 to 100
```

Now everything is ready. Time to get a guess. This is the top of a GOTO loop 'cause we'll probably get more than one guess. Note the semicolon (;) following INPUT. This prevents a carriage return and line feed after someone types a guess and presses ENTER. We do this so we can print the stars on the same line as the guess.

```
400 REM ## Get a guess ##
410 PRINT: INPUT: "Your guess"; Guess
```

OK, now we have a guess. We compute the Distance of the guess from the secret number. This is the absolute value (ABS) of the difference between the secret number and the guess.

```
500 REM ## Compute distance from secret number ##
510 Distance = ABS(Secret - Guess)
```

Well, if the distance is zero, the guess is right on. Let's check for that. If the distance is zero, zoom down to line 610 and tell the lucky player she or he is a winner.

```
600 REM ## Check for a win ##
610 IF Distance = 0 THEN GOTO 610
```

The first guess or two or more probably aren't winners. So let's print an appropriate hint and go back for another guess. Because of that semicolon following INPUT in line 410, the hint (stars) will be on the same line as the guess.

```
700 REM ## No win, so print a hint & go for new guess ##
710 IF Distance >= 64 THEN PRINT TAB(20); " * "; GOTO 410
720 IF Distance >= 32 THEN PRINT TAB(20); " *** "; GOTO 410
730 IF Distance >= 16 THEN PRINT TAB(20); " ***** "; GOTO 410
740 IF Distance >= 8 THEN PRINT TAB(20); " ***** "; GOTO 410
750 IF Distance >= 4 THEN PRINT TAB(20); " ***** "; GOTO 410
760 IF Distance >= 2 THEN PRINT TAB(20); " ***** "; GOTO 410
770 IF Distance = 1 THEN PRINT TAB(20); " ***** "; GOTO 410
```

Back in line 610, if the distance really was zero, you get here.

```
800 REM ## Winner! Press a key to play again ##
810 PRINT
820 PRINT "***** That's it!!!"
830 PRINT
840 PRINT "Press a key to play again."
850 key$ = INPUT$(1): GOTO 210
```

That's about it. The program works OK, but we don't like it much. Too many GOTOs. Needs some color and sound. Hard to change for another number range. Oh well, next time...

The Solution People

Two readers have solved most of the problems that have been posed here.

Ed Ayres, 929 Massachusetts Ave., #2-B, Cambridge, MA 02139 solves everything in good programming style using QuickBASIC and True BASIC. He also solves problems we haven't yet posed.

John Straub, 17 Lakeside Dr. Ext., Ridgefield, CT 06077 uses GW-BASIC. If we show 3 ways to solve a problem, John sends method #1.

Write to Ed and John for solutions. Please include a self-addressed stamped envelope to make it easy for them to handle lots of requests.

New Problems

Problem 72. Keyboard Experts for Little Kids

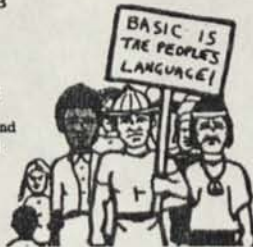
Your BASIC Backpack #28

by Bob Albrecht & Don Inman

If you can read a comic book or a newspaper, you can learn to read and understand programs in BASIC. Well, you can if the programs are written so as to encourage humans to understand them. That's how we try to write programs in "Your BASIC Backpack."

You can learn to read and understand BASIC, express yourself in BASIC, and use it to make the computer do what you want it to do the way you want it done.

"Your BASIC Backpack" provides tutorials for beginners who want to learn how to use Microsoft GW-BASIC and QuickBASIC. We also pose problems for you to think about, play with, and solve. Bob & Don, P.O. Box 1635, Sebastopol, CA 95473.



The BASIC Teacher — a FREE Short Subscription

The BASIC Teacher is our magazine about learning and teaching Microsoft GW-BASIC and QuickBASIC. Every issue has "Teach Yourself QuickBASIC" and "Teach Yourself GW-BASIC" for beginners, "Browsing BASIC" for people who are past the novice level, and a new section called (surprise) "Your BASIC Backpack" with problems and solutions for everyone. Readers of "Your BASIC Backpack" in *COMPUTER SHOPPER* may have a FREE short subscription (3 issues) to The BASIC Teacher. Send your request to Tadeasi Khara, The BASIC Teacher, 2814 - 19th St., San Francisco, CA 94110. Offer expires December 31, 1989

The Family Computer Club

We've been looking at lots of educational shareware. Here are some good ones for kids of pre-school to early elementary school age. Look for these in the shareware ads in *COMPUTER SHOPPER*.

- **ABC Fun Keys.** Has four games to help kids learn the alphabet.
- **My's First Primer.** Six games to help kids learn about the alphabet, numbers, shapes and patterns.
- **Fun with Letters & Words.** A personalized First Reader to help kids learn reading and spelling.
- **Word Processor for Kids.** For kids from about age 5 on up. If you have never used a word processor, you might try this one yourself -- when the kids aren't watching.

We'll include more information on these and other educational shareware programs in *DragonBooks #2*, so send for your free copy.

Teach Yourself QuickBASIC

This section began in Your BASIC Backpack #6, April 1986. So we now assume that you are acquainted with several of QB's windows, menus, and operation procedures. There are more than 200 keywords in QuickBASIC's vocabulary.

This time we will use only the following keywords:

AS	DIM	IF	LINE	PRINT	THEN
CHRS	DO	INPUT	LOCATE	PUT	TYPE
CLOSE	END	INPUTS	LOOP	RANDOM	UCASE\$
CLS	EXIT	LEFT\$	LOOP	REM	
DEFINT	FOR	LEN	OPEN	STRING	



We continue with our slow tutorial on random-access files. This time we'll create the Japanese.Ran file, a file you can use to study Japanese. Or, you can use our programs to create files to study something else: Spanish or antonyms or definitions of fancy words (in English) or just about anything you might use verbal flashcards to help you learn.

Create the Japanese.Ran Random-access File

The Japanese.Ran file is a random-access file that contains Japanese words and phrases and their English equivalents. Each record in this file has three string fields:

Phrase.Japanese	A Japanese word or phrase with a length of exactly 38 characters.
Phrase.English	The English equivalent, or near equivalent, of the Japanese word or phrase. This field is also exactly 38 characters long.
Phrase.Eor	End of record characters, CR (ASCII 13) and LF (ASCII 10).

We chose field lengths of 38 characters each for the Phrase.Japanese and Phrase.English fields so that they can be placed side-by-side on one line of the screen. The Phrase.Eor field is included so that you can use the DOS TYPE command to display the file on the screen, or the DOS PRINT command to print the file to your printer.

You can create the Japanese.Ran file, or a file of your choice, by entering records from the keyboard. Sample records are shown below. The spelling conventions are consistent with those used in *Learn Japanese* by John Young and Kimiko Nakajima-Okano (Honolulu: University of Hawaii Press, 1984). If you want to learn Japanese, this is an excellent book to use for self-study. We'd love to hear from people who are learning Japanese, or who would like to begin learning Japanese.

Record Number	First Field (Phrase.Japanese)	Second Field (Phrase.English)
1	Nihon'go	Japanese language
2	Ohayoo gozaimasu	Good morning
3	Kon'nichi wa	Hello, Good day
4	Kon'ban wa	Good evening
5	Sayoonara	Goodbye
6	Doozo	Please
7	Arigatoo gozaimasu	Thank you



And now, on to the program. We'll show you Program YBB2801Q a piece at a time. Here's the first piece, our usual title and setup blocks.

```
REM ## Create a Random-access 'Flashcard' File ##
' Your BASIC Backpack #28. 9/30/89
' Microsoft QuickBASIC. Filename: YBB2801Q.BAS
```

```
REM ## Set up ##
DEFINT A-Z
CLS
```

We anticipate making several files to help us study Japanese. So the next block gets the name of the file and the disk drive. It then puts these together to create the value of the string variable filename\$.

```
REM ## Get name of file & disk drive ##
LOCATE 1, 1: PRINT "Create a random-access flashcard file."
LOCATE 3, 1: INPUT "File name "; name$
LOCATE 5, 1: INPUT "Disk drive"; drive$
filename$ = drive$ + ":" + name$
LOCATE 7, 1: PRINT "Create the "; name$; " file on drive "; drive$
LOCATE 9, 1: PRINT "Press a key to begin."; akey$ = INPUT$(1)
PRINT
```

You can try this much of the program now. It might go like this:

```
Create a random-access flashcard file.
File name ? Japanese.Ran
Disk drive? b
Create the Japanese.Ran file on drive b
```



The file name and disk drive letter can be in upper case, lower case, or a mixture of both. For example, you can type the file name as shown, or as JAPANESE.RAN, or as japanese.ran.

The next block uses TYPE...END TYPE to define the structure of the file. It specifies that each record will consist of three string fields.

```
REM ## Define structure of random-access file record ##
TYPE RecordType
Japanese AS STRING * 38 'String field with 38 characters
English AS STRING * 38 'String field with 38 characters
Eor AS STRING * 2 'String field with 2 characters
END TYPE
```

This defines a record as having three fields. The first field (Japanese) has exactly 38 characters. The second field (English) has exactly 38 characters. The third field (Eor) has exactly 2 characters. Therefore, every record has exactly 78 characters. RecordType is the name of the record structure. We chose this name arbitrarily and capriciously. We could have used Gendalf or FlashCard or WalzingMathilda or any name that conforms to the conventions for naming QuickBASIC variables. We think RecordType is reasonably mnemonic. Other mnemonic choices might be RecordStructure or FieldDefinitions or ... well, you can probably invent names that tickle your memory.

Now that the record structure is defined, we need to declare a variable of the type defined as RecordType. This variable will be used to store an entire record. Since each record consists of phrases, we decide to name our variable Phrase. This automatically creates three field variables, Phrase.Japanese, Phrase.English, and Phrase.Ror. A value is then assigned to Phrase.Ror.

```
REM %% Declare a variable of above type %%
DIM Phrase AS RecordType
Phrase.Ror = CHR$(13) + CHR$(10) 'End of record characters
```

The value of Phrase.Ror is a two-character string consisting of the end-of-record characters, CR & LF. Values of Phrase.Japanese and Phrase.English are acquired later, when someone enters them from the keyboard. Now open the file.

```
REM %% Open the file named by the variable, filename$ %%
OPEN filename$ FOR RANDOM AS #1 LEN = LEN(Phrase)
```

Note that we didn't specify the length of a record as 78 characters. The clause LEN = LEN(Phrase) takes care of that automatically. This OPEN statement is pretty general. The name of the file is the value of filename\$. If you wish, you can use a variable to also specify the file number. As time goes by, we'll introduce other ways to make our programs more general.

When you open a random-access file, it is open for both input and output. You can write information to the file or read information from the file.

Records in a random-access file are numbered 1, 2, 3, and so on. When you want a record, you use its record number. The next program block sets the RecordNumber variable to zero (0). This variable is then used in a DO...LOOP to keep track of records by the numbers -- as you enter them and they are put into the file.

```
REM %% No records yet, so set record number to zero %%
RecordNumber = 0
```

Now we are ready for the part of the program that does most of the work. The following DO...LOOP acquires records entered from the keyboard and PUTs (writes) them to the file. You can quit by entering the letter "Q" as the Japanese phrase. This causes an exit from the DO...LOOP.

```
REM %% Enter records from keyboard, put into file %%
DO
  LINE INPUT "Japanese (Q to quit)? "; Phrase.Japanese
  FirstLetter$ = LEFT$(Phrase.Japanese, 1)
  IF UCASE$(FirstLetter$) = "Q" THEN EXIT DO
  LINE INPUT "English ? "; Phrase.English
  PRINT
  RecordNumber = RecordNumber + 1
  PUT #1, RecordNumber, Phrase
LOOP
```

The statement: LINE INPUT "Japanese (Q to quit)? "; Phrase.Japanese

acquires a string entered from the keyboard and assigns it as the value of Phrase.Japanese. If you enter fewer than 38 characters, spaces are added to make a total of 38 characters. If you enter more than 38 characters, the first 38 are assigned as the value and the rest are ignored. For example, suppose you enter Nihon'go and press ENTER. This string has eight characters. The value of Phrase.Japanese will consist of these eight characters followed by 30 spaces, for a grand total of 38 characters.

If you press the Q key to quit, 37 spaces are automatically appended and the result is stored as the value of Phrase.Japanese. Well, there aren't any Japanese words beginning with Q. If the first letter of the value of Phrase.Japanese is Q (or q), the following lines cause an exit from the DO...LOOP.

```
FirstLetter$ = LEFT$(Phrase.Japanese, 1)
IF UCASE$(FirstLetter$) = "Q" THEN EXIT DO
```

Remember, if the value of FirstLetter\$ is either "Q" or "q", then the value of UCASE\$(FirstLetter\$) will be "Q".

After the exit from the DO...LOOP, all that is left is a little mop-up.

```
REM %% Print length of file, close file, end program %%
PRINT
PRINT "The "; name$; " file has"; LOC(1); "bytes."
CLOSE #1
END
```

Hmm... Perhaps we should have also printed the number of records in the file. You can make that change, can't you?

Time for a test run. We'll enter the seven records shown previously.

Create a random-access flashcard file.

File name ? Japanese.Ran

Disk drive? b

Create the Japanese.Ran file on drive b

Press a key to begin.

Japanese (Q to quit)? Nihon'go
English ? Japanese language

Japanese (Q to quit)? Ohayoo gozaimasu
English ? Good morning

Japanese (Q to quit)? Kon'nichi wa
English ? Hello, Good day

Japanese (Q to quit)? Kon'ban wa.
English ? Good evening

Japanese (Q to quit)? Sayoonara
English ? Goodbye

Japanese (Q to quit)? Doozo
English ? Please

Japanese (Q to quit)? Arigatoo gozaimasu
English ? Thank you

Japanese (Q to quit)? q

The Japanese.Ran file has 546 bytes.

Press any key to continue

Let's see now -- we entered seven records and we know that each record should occupy 78 bytes. So...

7 x 78 = 546 ACK!

View the Japanese.Ran File in the View Window

You can load the Japanese.Ran file into QB's View Window and view it. However, since it is not a program, if you try to edit it you will see a Syntax error dialog box. Of course, you can invoke the Full Menu and turn off the syntax checking. We assume you know how to change from QB 4.5's Easy Menu to Full Menu, and conversely. Is our assumption correct?

If you are using the QuickBASIC Interpreter, Academic Edition, you can load the Japanese.Ran file into the View Window more easily -- load it as a Document instead of as a Program. We use the QB Interpreter for most of our BASIC brainstorming. Well, QB 4.5 or QB1, here is the Japanese.Ran file in the view window, loaded as a Document.

File	Edit	View	Search	Run	Debug	Options
Nihon'go						Japanese language
Ohayoo gozaimasu						Good morning
Kon'nichi wa						Hello, Good day
Kon'ban wa						Good evening
Sayoonara						Goodbye
Doozo						Please
Arigatoo gozaimasu						Thank you

We use INSET to get screen pictures

Use DOS TYPE to Display the Japanese.Ran File

Since each record of the Japanese.Ran file has end-of-record characters, you can use the DOS TYPE command to display the file on the screen, one record per line of the screen. Remember, each field is 38 characters long, so the two fields will fit side-by-side on one line. Suppose the Japanese.Ran file is on the disk in drive B. At the DOS A PROMPT (A_) or whatever DOS prompt you see, do the following.

Type:

TYPE b:Japanese.Ran (It is OK to type: type)

and press ENTER.

When we did it, our screen looked like this:

```
A:)TYPE b:Japanese.Ran
Nihon'go           Japanese language
Ohayoo gozaimasu   Good morning
Kon'nichi wa       Hello, Good day
Kon'ban wa         Good evening
Sayoonara          Goodbye
Doozo              Please
Arigatoo gozaimasu Thank you
```



Preview of Coming Attractions

We'll use the Japanese.Ran file to demonstrate assorted programs. For example:

- A program to scan the file sequentially from record #1 to the end of the file. Press a key to see the next entire record.
- A program to scan the file to scan the file sequentially, a phrase at a time -- first the Japanese phrase, then the English phrase. Or, if you prefer, you (or we) can change the program to show the English phrase first, then the Japanese phrase.
- A program to scan the file randomly, an entire record each time you press a key (or a designated key).
- A program to scan the file randomly and present either the Japanese or English phrase first, then the corresponding phrase in the other language. For example, you might press the J key to get the Japanese phrase first, then press E when you want to see the corresponding English phrase. It would be easy to change this program to get a timed drill. Press J to get the Japanese phrase or E to get the English phrase. Then, in x seconds (you choose the value of x), the corresponding phrase appears.
- A program to edit the file: change, add, or delete records.

You can modify program YRB2801Q and customize it for your use. For example, suppose you want to study Spanish instead of Japanese. Change the program block that defines the record structure to, for example, the following.

```
REM ** Define structure of random-access file record **
TYPE RecordType
  Spanish AS STRING * 38      'String field with 38 characters
  English AS STRING * 38      'String field with 38 characters
  Nor AS STRING * 2          'String field with 2 characters
END TYPE
```

Having made the above change, you now must make two minor changes in the DO...LOOP, as follows:

```
REM ** Enter records from keyboard, put into file **
DO
  LINE INPUT "Spanish (Q to quit)? "; Phrase.Spanish
  FirstLetters = LEFT$(Phrase.Spanish, 1)
  IF UCASE$(FirstLetters) = "Q" THEN EXIT DO
  LINE INPUT "English      ? "; Phrase.English
  PRINT
  RecordNumber = RecordNumber + 1
  PUT #1, RecordNumber, Phrase
LOOP
```

You can make the program seem even more general (to the casual reader) by changing three program blocks as shown below.

```
REM ** Define structure of random-access file record **
TYPE RecordType
  SideA AS STRING * 38      'String field with 38 characters
  SideB AS STRING * 38      'String field with 38 characters
  Nor AS STRING * 2        'String field with 2 characters
END TYPE
```

```
REM ** Declare a variable of above type **
DIM FlashCard AS RecordType
FlashCard.Nor = CHR$(13) + CHR$(10) 'End of record characters
```

```
REM ** Enter records from keyboard, put into file **
DO
  LINE INPUT "Flashcard, side A (Q to quit)? "; FlashCard.SideA
  FirstLetters = LEFT$(FlashCard.SideA, 1)
  IF UCASE$(FirstLetters) = "Q" THEN EXIT DO
  LINE INPUT "Flashcard, side B      ? "; FlashCard.SideB
  PRINT
  RecordNumber = RecordNumber + 1
  PUT #1, RecordNumber, FlashCard
LOOP
```

Well, that just scratches the surface of possibilities. What would you like to see? Remember, all of the above will work for files other than the Japanese.Ran file. What would you like to study by the flash card method?



Teach Yourself GW-BASIC

GW-BASIC is the generic form of Microsoft BASIC. If you have an MS-DOS computer, you probably have GW-BASIC. BASICA is similar to GW-BASIC, licensed to and sold by IBM. We use the Tandy version of GW-BASIC, called BASIC on the disk supplied with our Tandy 1000TX. GW-BASIC has nearly 200 keywords. This time, we'll use only a few, listed below.

ABS	IF	LEFT\$	REM	TIMER
CHR\$	INPUT	LOCATE	END	WIDTH
CLS	INPUT\$	OFF	SCREEN	
DEFINT	INT	PRINT	TAB	
GOTO	KEY	RANDOMIZE	THEN	



We show keywords in all upper case letters and variables in lower case or a mixture of upper and lower case letters. We do this to make programs easy for you to read and understand. HOWEVER, when you LIST a program, GW-BASIC lists variables in upper case.

Stars -- a Guessing Game, Version 2

We hope you enjoyed playing Stars and made some improvements to our primitive program. Here is another program to play Stars. The first part of Program YRB2802Q is almost the same as the first part of Program YRB2703Q, shown last time.

```
1 REM ** Stars -- a Guessing Game, Version 2 **
2 ' Your BASIC Backpack #27. 9/30/89
3 ' Microsoft GW-BASIC. Filename: YRB2802Q.BAS

100 REM ** Set up **
110 SCREEN 0: CLS : KEY OFF: WIDTH 40
120 DEFINT A-Z
130 RANDOMIZE TIMER

200 REM ** Tell how to play **
210 CLS
220 LOCATE 1, 1: PRINT "Welcome to my galaxy. Let's play a"
230 LOCATE 3, 1: PRINT "game. I'll think of a number from 1"
240 LOCATE 5, 1: PRINT "to 100. Guess my number. If you miss,"
250 LOCATE 7, 1: PRINT "I'll print some stars. The closer you"
260 LOCATE 9, 1: PRINT "are, the more stars you will see."
270 LOCATE 11, 1: PRINT "If you see 7 stars (S S S S S S S),"
280 LOCATE 13, 1: PRINT "you are very, very close to my number!"

300 REM ** Computer 'thinks' of a number **
310 Secret = INT(100 * RND) + 1 'Random integer, 1 to 100

400 REM ** Get a guess **
410 PRINT : INPUT "Your guess"; Guess

500 REM ** Compute distance from secret number **
510 Distance = ABS(Secret - Guess)

600 REM ** Check for a win **
610 IF Distance = 0 THEN 810
```

Block 700 is the work horse of the program. It figures out how many stars to print if the player didn't guess the secret number. Last time we did it one way. Here is another way.

```
700 REM ** No win, so print a hint & go for new guess **
710 Stars$ = "S S S S S S S"
720 IF Distance >= 64 THEN PRINT TAB(20); LEFT$(Stars$, 2); GOTO 410
730 IF Distance >= 32 THEN PRINT TAB(20); LEFT$(Stars$, 4); GOTO 410
740 IF Distance >= 16 THEN PRINT TAB(20); LEFT$(Stars$, 6); GOTO 410
750 IF Distance >= 8 THEN PRINT TAB(20); LEFT$(Stars$, 8); GOTO 410
760 IF Distance >= 4 THEN PRINT TAB(20); LEFT$(Stars$, 10); GOTO 410
770 IF Distance >= 2 THEN PRINT TAB(20); LEFT$(Stars$, 12); GOTO 410
780 IF Distance = 1 THEN PRINT TAB(20); LEFT$(Stars$, 14); GOTO 410
```

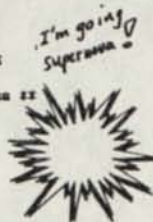
We also made a small change in the final program block (block 800). Now the computer prints the secret number at the end of the game. This makes it easier to find out if the program is really working properly.

```
800 REM ** Winner! Press a key to play again **
810 PRINT
820 PRINT "S S S S S S S S S S That's it!!!"
830 PRINT
840 PRINT "My secret number is"; Secret
850 PRINT
860 PRINT "Press a key to play again.";
870 akey$ = INPUT$(1): GOTO 210
```

Well, go ahead and play this version. Compare block 700 of this program with block 700 of last month's program. Then remember, there's always another way -- change only the title block (block 1) and block 700, as shown below, then save the modified program as YRB2803Q.

```
1 REM ** Stars -- a Guessing Game, Version 3 **
2 ' Your BASIC Backpack #27. 9/30/89
3 ' Microsoft GW-BASIC. Filename: YRB2803Q.BAS
```

```
700 REM ** No win, so print a hint & go for new guess **
710 Stars$ = "S S S S S S S"
720 PowerOfTwo = 64
730 EmbrOfStars = 1
740 IF Distance >= PowerOfTwo THEN 780
750 EmbrOfStars = EmbrOfStars + 1
760 PowerOfTwo = PowerOfTwo / 2
770 GOTO 740
780 PRINT TAB(20); LEFT$(Stars$, 2 + EmbrOfStars); GOTO 410
```



Let's run that last version, the one with the rather boggling block 700. Here are the guesses.

```

Your guess? 50      * * * * *
Your guess? 60      * * *
Your guess? 40      * * * * *
Your guess? 30      * * * *
Your guess? 45      * * * * *
Your guess? 44
* * * * * That's it!!!
My secret number is 44
Press a key to play again.
  
```



In case you have forgotten what the stars mean, here is a handy table. Of course, don't tell the player what the stars mean. Part of the game is gathering data (by playing) and figuring out what the stars mean.

Let D = distance of guess from secret number

Value of D	Number of Stars
64 or more	1
32 to 63	2
16 to 31	3
8 to 15	4
4 to 7	5
2 to 3	6
1	7

For you people who know a little math:

X = secret number

G = player's guess

$D = |X - G|$

Can you figure out how block 700 works? This block is easy to change for a different range of numbers (1 to 50 or 1 to 200 or 1 to 1000, etc.).

There's always another way. Try rewriting block 700 using the LOG function.

The Solution People

Two readers have solved most of the problems that have been posed here.

Ed Ayres, 929 Massachusetts Ave., #2-H, Cambridge, MA 02139 solves everything in good programming style using QuickBASIC and True BASIC. He also solves problems we haven't yet posed.

John Straub, 17 Lakeside Dr. Ext., Ridgefield, CT 06077 uses GW-BASIC. If we show X ways to solve a problem, John sends method $X+1$.

Write to Ed and John for solutions. Please include a self-addressed stamped envelope to make it easy for them to handle lots of requests.

New Problems

Problem 75. Missing Vowels Spelling Drill

Write a program to help a kid learn to spell. Pick a word from a bunch of DATA statements, an array, or perhaps a random-access file. Display the word on the screen with all vowels replaced by the underline character (), then wait patiently for someone to type the entire word and press ENTER. For example, display "DRAGON" like this:

D _ R _ G _ N

Use big letters (WIDTH 40). Remember, sometimes there might be two or more responses that are legitimate words. The easy way out is to say something like, "I'm thinking of a different word. Please try again." Or, you might have a short list of possible words and check them all.

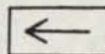
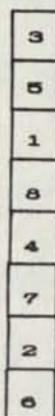
You should accept any mix of upper and lower case letters in a correct response. For example, dragon, DRAGON, and Dragon are all correct responses for the above. This is easy to do in QuickBASIC (use UCASE\$ or LCASE\$), but a little more difficult in GW-BASIC. In past episodes, we have shown a subroutine to take care of this problem.

Of course, think about including color, sound, interesting graphics, stuff to make the program more fun to use. Keep up the good work.



Problem 76. Putting Things in Order

How about a "putting things in order" game? First a game in which you put number blocks in order with the smallest number on top and the largest number on the bottom. It might begin like this.



Use the arrow keys (←→) to move the on-screen arrow to the block you want to move to the top. Make it thump against the number block and "lock-on" to it. Then "pull" the block out. The blocks above the one you pull out drop down into the empty space (thunk!). Now use the up arrow key to move the block up the screen, then use the left arrow key to put it on the top of the stack of blocks. Looks like we need a way to tell the big arrow to let go. Space bar? L key? You decide.

The goal is to put the blocks in order, with block #1 on top and block #8 on the bottom. When this is done, give a nice reward.

Variations

Instead of numbers 1 to 8, choose eight numbers from a larger set; 1 to 20, or 1 to 99, or -10 to 10, or your choice.

Instead of numbers 1 to 8, use letters A to H; or use eight letters selected from the entire alphabet.

Instead of using a big arrow to move the blocks, create a shape of your choice.

Problem 77. Hunt the Hurtle in SCREEN 0

SCREEN 0 is the text screen with 25 rows and 80 columns. Hide the Hurtle somewhere in row 1 to 20, column 1 to 80. For example, the Hurtle could be at row 8, column 73. You might use Hrow and Hcol as variables for the Hurtle's hiding place.

Now tell the player to hunt the Hurtle. A guess consists of a row number and a column number. You might use Grow and Gcol as variables for the guess.

After a guess, compute the distance of the guess from the Hurtle's hiding place, like this:

$$\text{Distance} = \text{SQRT}((\text{Hrow} - \text{Grow})^2 + (\text{Hcol} - \text{Gcol})^2)$$

If the distance is zero, reward the player for finding the Hurtle. Otherwise, color the screen location (Grow, Gcol), as follows.

Distance condition	Color	Color number
Distance < 2	light magenta	13
2 <= Distance < 6	light red	12
6 <= Distance < 24	yellow	14
24 <= Distance < 120	light green	10
120 <= Distance	light blue	9

Use the bottom five lines of the screen for communication with the player. Of course, you may want to begin with some instructions on how to play. Don't tell the player exactly what the colors mean. Just say or show that light magenta is very close (HOT!), light red is close (very warm), yellow is warm, and so on. If you need more room for communication with the player, feel free to reduce the area where the Hurtle can hide. Good hunting, but please don't hurt the friendly Hurtle!