OCTOBER 1989 BYTE MAC PORTABLE • OPTICAL TECHNOLOGIES • OPTICAL STORAGE Volume 14, Number 10

SETTING THE STANDARDS
15 YEARS

BYTE

OCTOBER 1989 A McGRAW-HILL PUBLICATION

Special Reports:

Apple's new portable

MAC *to go*

- Quattro 2
- Bill Gates on BASIC
- Parallel Processing with Macs
- Understanding Disk Caching
- Inside AppleTalk and NetBIOS
- 6 Short Takes

REVIEWS

- New IBM and Mitac SX Systems
- ColorQuick Mac Printer
- Two Portable Printers
- Logitech's MultiScope Debugger
- A new X Window System

$3.50 U.S.A./$4.50 IN CANADA
0360-5280
10
29920
0 440235 0

BYTE
SETTING THE STANDARDS
15
YEARS

THE 25TH BIRTHDAY OF BASIC

Everybody's first language is a quarter century old: You've come a long way, BASIC

Bill Gates

Since the first program ran on an English-built computer in 1948, computing has evolved by several orders of magnitude. In just 15 years, personal computers have grown from 8-bit, 4K-byte toys to powerful 32-bit, 16-megabyte tools for science and industry.

Amid all this growth in hardware, the BASIC language has quietly turned 25 years old, and the BASIC interpreter that opened up the microcomputer to millions of people has entered its fifteenth year.

BASIC didn't become the best-known and most accessible computer language simply because it comes free with every machine. BASIC's strengths—the simplicity of using an interpreter, its powerful string handling, the richness of the language, its English-like keywords and syntax, and the freedom it gives programmers to experiment—make it the ideal way for computer novices to explore the intricacies of their computers.

During this milestone year for BASIC, you can see how it may evolve by looking at the technology of operating systems like OS/2 and the promise of object-oriented programming.

BASIC was born out of a need to give nontechnical people a simple way to interact with a computer. In 1962, a math professor at Dartmouth College, Thomas Kurtz, proposed to the department chairman, John Kemeny, the radical idea that all Dartmouth students should learn about computers during their four-year stay. But the batch-oriented computers of the day made that prospect impossible, since it often took days to find out if a simple program would compile properly. Even then, the program could only calculate a result and return an answer; the person who wrote the program never actually saw it run.

Kemeny and Kurtz, expanding on research going on at MIT and Bell Labs, built a time-sharing operating system for a new computer that Dartmouth was about to receive. In the process, they decided that giving students a way to share the computer's hardware was of little use without a simple language for talking with the machine. FORTRAN and ALGOL didn't fill the bill for ease of use, so they created the Beginner's All-purpose Symbolic Instruction Code as a simplified blend of the best of FORTRAN and ALGOL.

On May 1, 1964, Dartmouth students, greeted by the now famous READY > prompt on their teletype terminals, could write simple programs and send them off to be compiled and run. Listing 1 shows a simple program (a multiplication table) written in that first BASIC. Kemeny and Kurtz recount the birth of BASIC in their book *Back to BASIC* (Addison-Wesley, 1985).

And Then There Were Micros

The first microcomputers, sporting very small memories, came on the scene in 1975 from companies like MITS and Southwest Technical Products. These machines understood only machine language. Paul Allen and I sensed an opportunity and wrote a version of BASIC to run in those small memory spaces. Our first BASIC for the MITS Altair allowed a user to run programs on a 4K-byte machine. Memory was so precious that we even replaced the READY > prompt with OK > to save a few bytes. We built this BASIC as an interpreter.

Memory constraints partly guided our decision to implement BASIC as an interpreter. But another factor was our fascination with interpreters, and the immediacy and ease of use they give to the programming art. An interpreter lets a programmer tell the computer to perform a job, and then the computer gives immediate feedback and results, including the reporting of errors. This immediacy comes from the interpreter's being built as part of the language, not as a separate program like a compiler. First-time users would find working with a compiler difficult.

Drawing on experience I had obtained writing a BASIC interpreter for a PDP-8 while in high school, Paul and I made our original microcomputer BASIC a single-representation interpreter. This means that rather than storing the exact source code in text form, we translated it into a more compact form because of the memory constraints we faced.

We did tricks to let the programmer see his or her program

continued

ILLUSTRATION: RANDY LYHUS © 1989

exactly as he or she keyed it in while executing the program at a reasonable speed. Using the lower values of a byte and the upper 128 ASCII values to tokenize BASIC keywords was an innovation in that interpreter. We also coined the short commands TRON and TROFF to turn on and off BASIC's earliest built-in debugging tool, a trace facility. Fitting the language's reserved words, error messages, and floating-point library to run programs in a 4K-byte machine required a lot of tricks—it's still my favorite piece of code because it is so refined.

We chose to write a BASIC interpreter for several reasons. First, BASIC was simple enough that we thought we could squeeze it, plus user programs, into 4K bytes. We also liked being able to write a meaningful one-line program in BASIC; with other languages, you had so many variable and environment definitions to worry about, you had to write several lines of code just to print "hello" on the screen. And we liked BASIC's powerful variable-length string-handling functions.

Listing 1: *The 1964 BASIC had no INPUT statement. Instead, you had to assign data to a variable with the LET or READ DATA statements.*

```
10 REM 1964 BASIC
20 LET N = 355/113
30 PRINT "MULTIPLICATION TABLE FOR",N
40 FOR I = 1 TO 10
50 PRINT I, N * I
60 NEXT I
70 PRINT "---------------"
80 END
```

Listing 2: *The 1977 8K-byte BASIC had longer variable names, the INPUT statement, strings, and the IF...GOTO statement. This is a more flexible version of the program in listing 1.*

```
10 REM 1977 8K BASIC
15 PRINT "ENTER NUMBER FOR MULTIPLICATION TABLE"
20 INPUT NUMBER
25 INPUT "ENTER NUMBER OF LINES"; LINES
26 IF LINES <1 GOTO 120>
30 PRINT "MULTIPLICATION TABLE FOR",NUMBER
40 FOR I = 1 TO LINES
50 PRINT I, NUMBER * I
60 NEXT I
70 PRINT "---------------"
80 PRINT "DO YOU WANT ANOTHER Y/N";
90 INPUT C$
100 IF C$ = "Y" GOTO 15
115 REM
120 REM ERROR HANDLING
130 PRINT "WRONG NUMBER OF LINES"
140 END
```

Listing 3: *The 1985 Microsoft QuickBASIC. It and other BASICs of the period eliminated line numbers and added subroutines and IF...THEN...ELSE statements.*

```
INPUT "File tb be searched";F$
INPUT "Pattern to search for";P$
OPEN F$ FOR INPUT AS #1
WHILE NOT EOF
   LINE INPUT #1, TEST$
   CALL LINESEARCH(TEST$,P$)
WEND

SUB LINESEARCH(TEST,P$) STATIC
    STATIC NUM
    NUM = NUM + 1
    X = INSTR(TEST$,P$)
    IF X = 0 THEN
       EXIT SUB
    ELSEIF X > 0 THEN
       PRINT "Line #";NUM;":";TEST$
    END IF
END SUB
```

Why Use BASIC?

We convinced the makers of the early microcomputers that they needed BASIC in their machines to let people use them. Otherwise, you could turn on the machine, but you couldn't do much with it. And until disk operating systems came along in about 1979, BASIC was the only way most people could use a personal computer. Our BASIC caught on because we encouraged people to write books about BASIC programming that contained lots of source code listings. We also promoted the idea of people writing programs in our BASIC and then selling or giving away those programs.

Other people were writing BASICs at that time: Li Chen Wang (Tiny BASIC), Steve Leininger (Radio Shack Level 1 BASIC), Steve Wozniak (Apple Integer BASIC), and Gordon Eubanks (BASICe and CBASIC). But Microsoft BASIC eventually appeared in the ROMs of over 50 types of machines and became a de facto standard.

That early BASIC was primitive compared to today's language. It forced you to use only two-letter variable names and required line numbers (see listing 2). But it proved to be an acceptable way to create useful programs for computers. It gave you instant feedback as you wrote a program, and its string-handling features let people write fairly sophisticated text-oriented applications. We invented new BASIC verbs like PEEK and POKE, and INP and OUT, to let people get at the resources of the machine. We devised a way to let programmers call machine language routines from BASIC so they could make critical parts of their programs faster.

Interpreters can be extended quite easily, so we frequently added new features to BASIC. We developed the music and graphics-string macro languages that used the verbs PLAY and DRAW. Many features that ended up in GWBASIC (short for Gee Whiz) came from our experience writing interpreters for Japanese machines.

BASIC hit a plateau when GWBASIC was put into the IBM PC ROM. Without IBM's updating the ROM, it was hard to popularize BASIC extensions beyond the interpreter. This BASIC became extremely popular, of course, and even today I'm amazed at what a high percentage of PC users have experimented with that BASIC because it is so simple to learn and use.

When BASIC faltered on the plateau of being an IBM PC-compatible standard, we created QuickBASIC. Other companies were extending BASIC, too—for example, BASIC's inventors, Kemeny and Kurtz, with True BASIC.

Not So Basic

When I show programmers the BASIC I use today, they say something like, "That looks a lot like Pascal." People who think of BASIC as the IBM PC's BASICA are surprised that it doesn't have to be a line-oriented language with little structure. BASIC has become very modern since 1983 (see listing 3).

In the mid-1980s, BASIC improved on three fronts: as a language, with the addition of long variable and label names, record structures, and better block-structuring tools; as an environment, with a built-in editor, debugger, and compile-run facility; and as a compiled language, with faster response in the classic compile/debug/edit/compile cycle while generating fast, high-quality code. BASIC also gained the ability to run

continued

Circle 96 on Reader Service Card →

High Performance Systems

Starting Price: $2480

Enjoy extra performance from your computer with Eltech!

"One of the best numeric performers of the bunch"
----MIPS Magazine, August 1989

The Eltech 9870 386 computer utilizes the best technology available today. Featuring the Floating Point Processor 80387-25/Weitek 3167, Eltech is now offering one of the fastest affordable computers that can be shipped to you TODAY! The 9870 also includes Cache Memory: 64 or 256KB S-RAM ; Expansion RAM: 4 or 8MB on 32-bit Board; Maximum RAM: 16MB Data Bus Width: 32-bit; Zero Wait States and 1 Serial and 1 Parallel Ports.

30-Day Money-Back Guarantee.
Next Day Free On-Site Service.
Next day delivery available.

- One year free on-site service
- Consulting service program
- Leasing and financing program

ELTECH RESEARCH INC.

Distributors & VAR inquiries welcome
Call for details.

1725 McCandless Drive
Milpitas, CA 95035
Telephone: 408-942-0990
FAX: 408-942-1410 Technical Support: 408-942-1067

~~Seven~~ ~~Six~~ ~~Five~~ easy ways to boost your BASIC

UPDATED!

PROBAS™ Basic Programming Library

So who cares that BYTE magazine calls **PROBAS** a "Supercharger for QuickBASIC" or that PC Tech Journal says that **PROBAS** is a "high-quality, high-quantity package"? Who buys a product just because Jerry Pournelle said "Anyone doing serious QuickBASIC programming would do well to get [**PROBAS**]"? And who cares that Wayne Hammerly calls **PROBAS** "The greatest thing since sliced bread"?

Who?--Only those who want to write better, faster, slicker programs and save hundreds of programming hours in the process. With all of that hoopla out of the way, we are formally announcing the momentous release of **PROBAS** Version 3.1, now with over 400 assembly routines to make BASIC programs faster and more powerful than you ever dreamed with features like:

- A 1,000-page two-volume manual
- Full mouse support
- Extended and EMS memory support
- Full-featured windowing
- Moveable, resizable windows
- Screen snapshots (text & graphics)
- Virtual screens in memory
- Lightning-fast file I/O
- Critical error handling
- String, array, and pointer sorts
- Search directories and archives

Create dazzling screens in text, CGA, EGA, VGA, and Hercules graphics modes with windows that can overlay one another and be moved and resized on the fly. Store megabytes of string, data, or screen snapshots in extended or EMS memory. Draw complex text or graphic screens to memory and snap them on in an eyeblink. The **PROBAS** file I/O routines allow you to read or write huge chunks of data at a clip, far faster than with BASIC.

PROBAS also has over 300 other essential services, including handy string, date, time, directory, and array manipulation routines; string, screen, and data compression routines; valuable equipment and input routines; and faster replacements for many BASIC commands.

Whether you are a professional or a novice, **PROBAS** will boost your BASIC in ways you never thought possible. **PROBAS** allows the professional to write faster, tighter code in much less time and allows novices to quickly and easily write professional-quality programs that would be impossible with BASIC alone. The bottom line is **PROBAS** adds power and saves time. After all, how much is a few hundred hours of your time really worth?

For all DOS versions of QuickBASIC and BASCOM. ***Just $149.00!***

UPDATED!

PROREF™ On-Line Help For PROBAS

PROREF provides on-line help for the routines in the **PROBAS** library. This hypertext manual links directly to the QB Advisor in QuickBASIC 4.5 so that the **PROBAS** reference becomes an integral part of your QuickBASIC on-line manual. Includes information and examples on **PROBAS** routines and helpful hints on programming in BASIC. ***Just $50.00!***

UPDATED!

PROSCREEN™ Screen Management

PROSCREEN is a full-featured screen generator/editor that will save you more design and coding time than you ever thought possible. **PROSCREEN** treats screens like a word processor treats text to provide complete control over characters, colors, and placement. Design input screens with up to 130 fields, 19 predefined and 2 user-defined masks. Save screens to screen files or .OBJ files and use the tight BASIC/Assembly code that comes with **PROSCREEN** to access the screens. There's no kludgy code generator here! Access hundreds of input screens with less than 25k of total code. ***Just $99.00!***

PROMATH™ Mathematics Library

PROMATH is a collection of over 150 high-level routines that provide mathematical functions and operations for programmers who often work in mathematics, science, or engineering. Complex variables, real and complex matrices, real and complex trigonometric and hyperbolic functions and their inverses, solution of linear equations, integration, differential equations, Fast Fourier transforms, graphics support, and many other useful routines are provided.

For years Fortran has been the language of choice for scientific and engineering applications, but it lacks many of the useful features of QuickBASIC. **PROMATH** contains most of the Fortran mathematical and numeric functions and allows you to easily translate Fortran code to BASIC or write new programs in BASIC, while retaining Fortran's numerical prowess.

The **PROMATH** manual is over 200 pages and provides a complete description of each routine, including any algorithm and the mathematical formula the routine uses, shown in standard notation. For QuickBASIC 4 and BASCOM 6 only. ***Just $99.00!***

UPDATED!

PROBAS™ TOOLKIT

The **TOOLKIT** is a collection of high-level BASIC and assembly modules that use the routines in the **PROBAS** library to save you even more hours of grunt work. Why spend hundreds of hours re-inventing the wheel when you can just plug in **TOOLKIT** modules like:

- Super-fast B-Tree indexing
- Ring, Bar, Pop-Up, Pull-Down menus
- Scroll-bar tag windows
- Dialog boxes with radio buttons
- Two mini-editors with word wrap
- BCD math routines
- Julian date & calendar routines
- Patch .EXE files
- Protected memory storage area

The **TOOLKIT** now supports EGA and VGA graphics modes for menus, windows, editors, calendars, and more. Complete with BASIC source code and an all-new comprehensive manual. The **TOOLKIT** requires the **PROBAS** library and helps conserve your greatest asset of all—time! ***Just $99.00!***

PROBAS™ TELECOMM TOOLKIT

The **PROBAS TELECOMM TOOLKIT** is a collection of high-level communications modules that you plug into your code to provide popular file transfer protocols, terminal emulations, login scripts and baud rates up to 115,200 baud. You get:

- Xmodem/Modem7/Xmodem-1k
- Ymodem (single and batch)
- CRC-16 and Checksum
- VT52, VT100, ANSI, BBS etc.
- Auto Dialer & data base
- Documented BASIC source

Why use clumsy SHELLs to complex terminal programs when you can plug just the communications routines you need into your code? Implement just the features and commands you want. Requires **PROBAS**. ***Just $75.00!***

Our thirty-day, money-back guarantee assures you the highest quality and our technical support staff is always ready to help.

HAMMERLY
COMPUTER SERVICES, INC.

9309 JASMINE COURT • LAUREL, MD 20707

(800) 343-7484

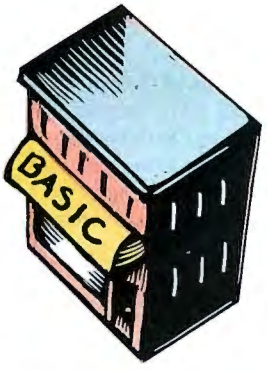
INT'L. ORDERS: (301) 953-2191 FAX: (301) 725-8147
BBS: (301) 953-7738

Add $5.00 per item ($8.00 Canada) for shipping per order. Europe: $39.00 for 1st item plus $5.00 for each additional item. Visa, M/C, C.O.D. (US Only) checks and approved POs accepted. Trademarks **PROBAS**, **PROREF**, **PROSCREEN**, **PROMATH**: Hammerly Computer Services, Inc. QuickBASIC, BASCOM: Microsoft Corp.

Circle 113 on Reader Service Card.

recursive routines, to handle sophisticated structures such as B-trees, and to deal better with graphics.

About this time, BASIC became more structured, with the addition of DO WHILE, WHILE...WEND, DO UNTIL, LOOP WHILE, and LOOP UNTIL loops, and SELECT CASE constructs. BASIC today has facilities that permit creating data structures as rich as those of C or Pascal. BASIC also shed the



That early BASIC was primitive compared to today's BASIC.

shackles of 64K-byte limits on program size; today's BASIC programs are limited only by memory, and even individual arrays can be up to the limits of memory in size.

You can also use long (32-bit) integers in your programs. Record structures added the capability to let you define your own data structures, combining many different individual data types, such as strings and numbers.

The environment for BASIC programming has changed, too. In QuickBASIC 4.0 and 4.5, for instance, a built-in full-screen editor with automatic syntax editing, a built-in debugger, and context-sensitive help makes the program-creation process as fast and easy as possible. Borland also integrated a full-screen editor into Turbo Basic.

Another important change in the modern BASICs was a move from interpreters back to compilers: Indeed, Dartmouth BASIC has remained a compiled language since its inception. BASIC compilers for microcomputers appeared soon after disk operating systems became available, and were made possible, in part, by the availability of up to 64K bytes of memory.

These compilers included Microsoft products and Digital Research's CBASIC-86 compiler. Later, Better BASIC, QuickBASIC, True BASIC, ZBasic, and Turbo Basic were developed. But the major move to compilers needed to be made without sacrificing the instant feedback and quick detection of errors, features that made the BASIC interpreter so successful.

Advancements in compiler technology enabled the development of a BASIC compiler that gave the appearance of an interpreter, thus offering the best of both worlds. Single-pass compilers that could compile first hundreds—and then thousands—of lines of code in just a few seconds gave BASIC the appearance of an interpreter but let programs execute faster.

Later, we invented an in-memory scheme combining the features of an interpreter and a compiler. It uses a special intermediate threaded pseudocode that, when combined with an environment like the QuickBASIC editor, lets you write, debug, and change the code just like in an interpreter. When you are ready, you can compile the finished product to get maximum speed and compact size in an executable file.

Compiling BASIC also made it easier to call routines written in other languages from within BASIC. Because the compiler generates standard intermediate .OBJ files, you can link C or FORTRAN .OBJ files with your BASIC .OBJ files and call

continued

NEW PRODUCT!

Unleash the BASIC Power of Hypertext!

Our latest BASIC-booster includes the first hypertext engine designed to be called from BASIC. With the **ProBas HyperHelp ToolKit** you can use the hypertext engine to put one or more manuals on-line with full hypertext search facilities. Imagine having one or more manuals on-line with all of the fantastic hypertext abilities in the QuickBASIC 4.5 compiler and then some:

- Choose single- or multi-window display
- Specify window colors and placement
- Move and resize windows with mouse or keys
- Sophisticated mouse and keyboard interface
- Up to 40 bookmarks to move between hyperlinks

In less than a dozen lines of code you can pop-up context-sensitive help at any time. Your users can then jump to related help, examples, or just browse the manual(s) in one or more windows that they can move or resize at will. The text will automatically wrap within the window to stay fully visible!

Create HyperCard Applications

Use the **HyperHelp** engine to create full-blown hypercard applications. Create sophisticated multi-window, multi-thread hypercard stacks. Mix cards and manuals for total data integration. Moving from link-to-link or card-to-card is instantaneous and the speed will amaze you.

Adding hypertext to your applications takes less than about a dozen lines of BASIC code. Converting ASCII text to hypertext is just as easy—just put delimiters around keywords, hyperlinks, and items you want to display in boldface.

Multiple Help Subsystems

The hypertext engine is just a part of the **HyperHelp ToolKit.** There is a wide selection of help subsystems with various displays, user interfaces, and memory requirements to suit almost any need. Want a small, window-oriented, keyword help system? It's in there! Need a Terminate & Stay Resident help system? It's in there! How about a lightbar indexing system that then pops-up the selected text? It's in there!

The **HyperHelp ToolKit** gets it's blinding speed by using the low-level routine in our **ProBas** Professional Basic Programming Library. **HyperHelp** requires the **ProBas** Library. See our ad on the opposite page for information on **ProBas,** shipping rates, and our thirty-day money-back guarantee.

Special Introductory Price—For a short time this powerful hypertext engine and collection of help subsystems is available for ***just $99.00!***

HAMMERLY
COMPUTER SERVICES, INC.
9309 JASMINE COURT • LAUREL, MD 20707
(800) 343-7484
INT'L. ORDERS: (301) 953-2191 FAX: (301) 725-8147
BBS: (301) 953-7738

Add $5.00 per item ($8.00 Canada) for shipping per order. Europe: $39.00 for 1st item plus $5.00 for each additional item. Visa, M/C, C.O.D. (US Only) checks and approved POs accepted. Trademarks ProBas, ProRef, ProScreen ProMath: Hammerly Computer Services, Inc. QuickBASIC, BASCOM: Microsoft Corp.

those "foreign" language routines from within your BASIC program.

With these improvements in compiler technology, BASIC's longtime reputation as a slowly executing language fell by the wayside. Today, many popular commercial applications are written in BASIC. For example, there's a speaker-design program called AccoustaCADD, a radio station playlist manager called Music Scan, and even a program for molecular biologists called DNA Inspector. Modern compiled BASIC programs run as fast as programs compiled in Pascal or C.

I'm still a big fan of BASIC. It handles many of the overhead tasks that other languages force you to deal with, thereby unleashing your programming creativity. I have challenged programmers to write code to solve any problem using any tool they choose; I wager that I can write the same program faster using QuickBASIC. In fact, there is a joke around Microsoft that says when an application is falling behind schedule, "Just give it to Bill and he'll write it in BASIC over the weekend."

Still, as powerful as BASIC is today, there are ways to make it even better.

The BASIC Clairvoyant

Even though BASIC gives you great freedom for writing programs, it still forces you to think procedurally; your programs become a series of linked, step-by-step procedures to perform certain actions. Programming this way becomes an exercise in project management: First do this, then do that if something else happens, and so on.

As programs become more complex, these "projects" get harder to manage. Structured programming concepts deal with this complexity by encouraging programmers to break up code into small chunks, and then tie the chunks together with the unfortunately error-prone convention of function calls within, and across, module and file boundaries. Object-oriented programming puts a new twist on that idea: Make each program chunk a self-contained package of data and the procedures for manipulating that data, and then have the chunks send and receive messages to interact with other chunks.

This scheme provides a "black-box" mentality for program pieces—the pieces become objects whose internal complexity stays hidden from other objects. But why do you need to think of programs as collections of objects?

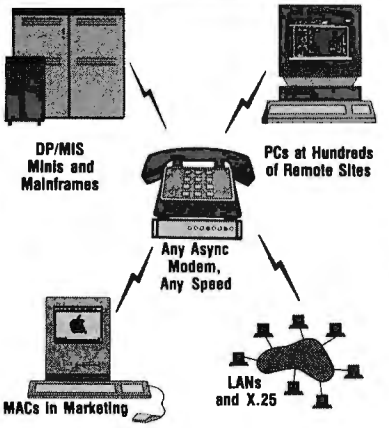
The visual environments that are used on computers today give a clue as to why thinking about program objects is important and different. In a visual environment, you choose an action or event after you choose the data that is the target of the action or event. For example, `delete character`, `delete word`, `delete paragraph`, and `delete page` are simply one command applied to different objects. You must choose the object—a character, word, paragraph, or page—before you select the command `delete`.

Object-oriented design techniques package data and the methods of manipulating that data together: The package becomes an object. If you want an object to do things that it doesn't currently know how to do, you create a new object that inherits the capabilities of the old object and then add the new capabilities you want to define. In traditional procedural programming, you set up the action first and then identify the data and supervise every step in the action's manipulation of that data.

Object-oriented programming will influence the development of all languages, including BASIC, over the next several years. Objects contain both code and data, and they manage their own behavior while hiding their internal complexity.

continued

BLAST...
Complete Communications, ONE Software Package.



PC, MAC, UNIX, XENIX, VAX,
and more. 30 popular operating systems. Connect and manage file transfers around the office or around the world.

RIGHT OUT OF THE BOX
Use regular modems, V.32, new high speed modems, X.25,LANs . . . BLAST makes them all compatible.

FOR INSTANT NETWORKS
Link two computers or 2000 . . . with
- One easy, identical interface
- One set of commands
- One powerful script language
- One reliable program

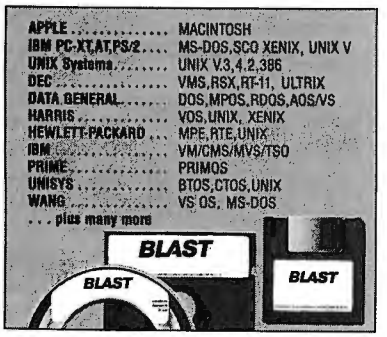
WITH ALL THE FEATURES
- Bulletproof file transfer
- Terminal Emulators – VT 100/220, etc.
- Scripting for customized routines
- PC to PC Remote Control
- Fully automated operation
- Unbeatable noise resistance

IN GOOD COMPANY
- Over 50,000 users worldwide
- Top-Notch technical support

***Call* 800-24-BLAST**

APPLE	MACINTOSH
IBM PC-XT,AT,PS/2	MS-DOS,SCO XENIX, UNIX V
UNIX Systems	UNIX V.3,4.2,386
DEC	VMS,RSX,RT-11, ULTRIX
DATA GENERAL	DOS,MPOS,RDOS,AOS/VS
HARRIS	VOS,UNIX, XENIX
HEWLETT-PACKARD	MPE,RTE,UNIX
IBM	VM/CMS/MVS/TSO
PRIME	PRIMOS
UNISYS	BTOS,CTOS,UNIX
WANG	VS OS, MS-DOS

. . . plus many more



BLAST
Communications Research Group
5615 Corporate Blvd. • Baton Rouge, Louisiana 70808
(504) 923-0888

Circle 53 on Reader Service Card (DEALERS: 54)
Circle 182 on Reader Service Card →

Discover Parallel Processing!

Monoputer/2™

The World's Most Popular Transputer Development System

Since 1986, the MicroWay **Monoputer** has become the favorite transputer development system, with thousands in use worldwide. Monoputer/2 extends the original design from 2 to 16 megabytes and adds an enhanced DMA powered interface. The board can be used to develop code for transputer networks or can be linked with other Monoputers or Quadputers to build a transputer network. It can be powered by the 20 MHz T414 or T800 or the new 25 MHz T425 or T800.

Parallel Languages

Fortran and C Make Porting a Snap!

Microway stocks parallel languages from 3L, Logical Systems and Inmos. These include one Fortran, two Cs, Occam, Pascal, and our own Prolog. We also stock the NAG libraries for the T800 and Rockfield's structural and thermal finite element package. A single T800 node costs $2,000, yet has the power of a $10,000 386/1167 system. Isn't it time you considered porting your Fortran or C application to the transputer?

Quadputer™

Mainframe Power For Your PC!

MicroWay's **Quadputer** is the most versatile multiple transputer board on the market today. Each processor can have 1, 4 or 8 megabytes of local memory. In addition, two or more Quadputers can be linked together with ribbon cables to build large systems. One MicroWay customer reduced an 8 hour mainframe analysis to 15 minutes with five Quadputers, giving him realtime control of his business.

For further information, please call MicroWay's Technical Support staff at (508) 746-7341.

MicroWay

World Leader in PC Numerics

P.O. Box 79, Kingston, MA 02364 USA (508) 746-7341
32 High St., Kingston-Upon-Thames, U.K., 01-541-5466
USA FAX 617-934-2414 Australia 02-439-8400 Germany 069-75-1428

Objects interact with other objects by passing messages back and forth. That way, any object can be changed without affecting other objects. Any programmer can create an object that can interact with any other object; all the programmer has to know is the message-passing syntax.

To adapt to this object-oriented emphasis in programming, BASIC will need the ability to create and manipulate classes and subclasses, which are the principal data abstraction devices of object-oriented programming. BASIC will also need a visual component. After all, the best way to design a form is to draw the form, not write code to reproduce it. With a mouse and a palette of predrawn graphics images, you should be able to combine lines, boxes, and buttons interactively on a screen to design a form for a program. That kind of interactive design of objects should also let you attach to or combine your creations within a program.

Future versions of BASIC will increasingly provide support for this kind of programming. The programs will look different from the BASIC we're used to. A visual BASIC program will be a mixture of code, programmer-written objects, and visually specified objects. With separate windows, you would be able to edit both the visual and code parts of the program.

HyperCard provides an interesting example of this combination of visual and more standard procedural programming. The screen display of the card, with its buttons and display areas, provides the visual field; the script language, HyperTalk, lets you create procedures that relate to the card and the information that the card links with. Although HyperCard is only a partial implementation of object-oriented programming, it forms an understandable intermediate step between procedural and object-level programming.

Another major change in BASIC's future will be its universality. This change will result from BASIC's becoming part of the overall computing environment. BASIC may soon become the equivalent of today's MS-DOS command interpreter, COMMAND.COM. If this happens, BASIC will play a role similar to the one it played in the earliest personal computers—as a universal "macro" language through which users interact with their computers and all the applications they run.

A universal BASIC would come in different flavors. One flavor would serve as a central control language for the computing environment. As an environment control tool, BASIC would be a command-line adjunct to a visual user interface. Another flavor would serve as an embedded language within application programs. Embedded BASIC would allow power users to embellish the performance of their applications, as Lotus 1-2-3 users do today, without having to learn a new macro language for each application they use.

You would be able to use BASIC syntax to tell your application programs to perform their various functions in the order you want them performed. For example, if you needed a certain report each month, you'd be able to write a short BASIC program that tells the application to gather the relevant data, format it properly, and have it waiting for you at the appointed hour. BASIC can provide this power while remaining easy to learn and use. These kinds of changes will keep BASIC vibrant for a long time.

Perpetually Young

The next 25 years in BASIC's life will probably be even more exciting than the first 25. BASIC's development has mirrored the explosive growth of the personal computer industry: When the hardware was weak, BASIC was pretty cramped, too. The rapid improvements in hardware in the last 15 years have been matched by the growth of BASIC in both power and flexibility. If you haven't looked at BASIC in the last few years, you'll be surprised by how it's changed. Compare listings 1, 2, and 3 to see the evolution of BASIC from 1964 through 1985.

In the late 1970s, when C started becoming very popular, many people predicted the demise of the COBOL and FORTRAN languages. Pascal was supposed to sound the death knell for BASIC. But all these languages continue to survive. I believe that in the case of BASIC, the language is flourishing. If BASIC becomes a universal macro language, as I think it will, it may outlive all the procedural languages. BASIC may, in fact, outlive us all. ■

Bill Gates is chairman and CEO of Microsoft Corp. in Bellevue, Washington. He cofounded the company in 1975 with Paul Allen. He can be reached on BIX c/o "editors."

EDC Electronic Design Center

High Tech in Perfection
Transputer Products

Transputer Boards

EDC-B016	Motherboard for 1-16 TRAM
EDC-4T4MB-T414	Transputer board with 4 x T414
EDC-4T4MB-T800	Transputer board with 4 x T800
EDC-4T4MB-ITEM	Rack for 10 EDC-4T4MB cards
EDC-4T4MB-TDS	Software Development System

Transputer Module Series 501

EDC-501-128KS*	Transputer module with T414 or T800 and 128K SRAM
EDC-501-256KS*	Transputer module with T414 or T800 and 256K SRAM
EDC-501-1MD-T414-70	Transputer module with T414 and 1MB DRAM (70ns)
EDC-501-1MD-T414-100	Transputer module with T414 and 1MB DRAM (100ns)
EDC-501-1MD-T800-70	Transputer module with T800 and 1MB DRAM (70ns)
EDC-501-1MD-T800-100	Transputer module with T800 and 1MB DRAM (100ns)

All transputer modules pin-compatible with INMOS TRAMS.

Transputer Interface Module

EDC-UBX1	UNIBUS interface card
EDC-TVM1	VME-Bus interface card
EDC-TQB1	Q-Bus interface card.

Transputer Software

EDC-SWMS1	Mathematical and statistical package in OCCAM
EDC-SWVDOT	RAM-Disk for Transputer boards

EDC Memory Module

EDC-SX32-064	64K x 8 SRAM module
EDC-SX32-256	64K x 32 SRAM module
EDC-DX32-256	256K x 32 DRAM module
EDC-DX40-256	256K x 40 DRAM module

EDC GmbH, Taunusstr. 51/III, 8000 Munchen 40 Tel.: (89) 350 70 76 Fax: (89) 359-61-80 Tx.: 521 25 99

Circle 92 on Reader Service Card