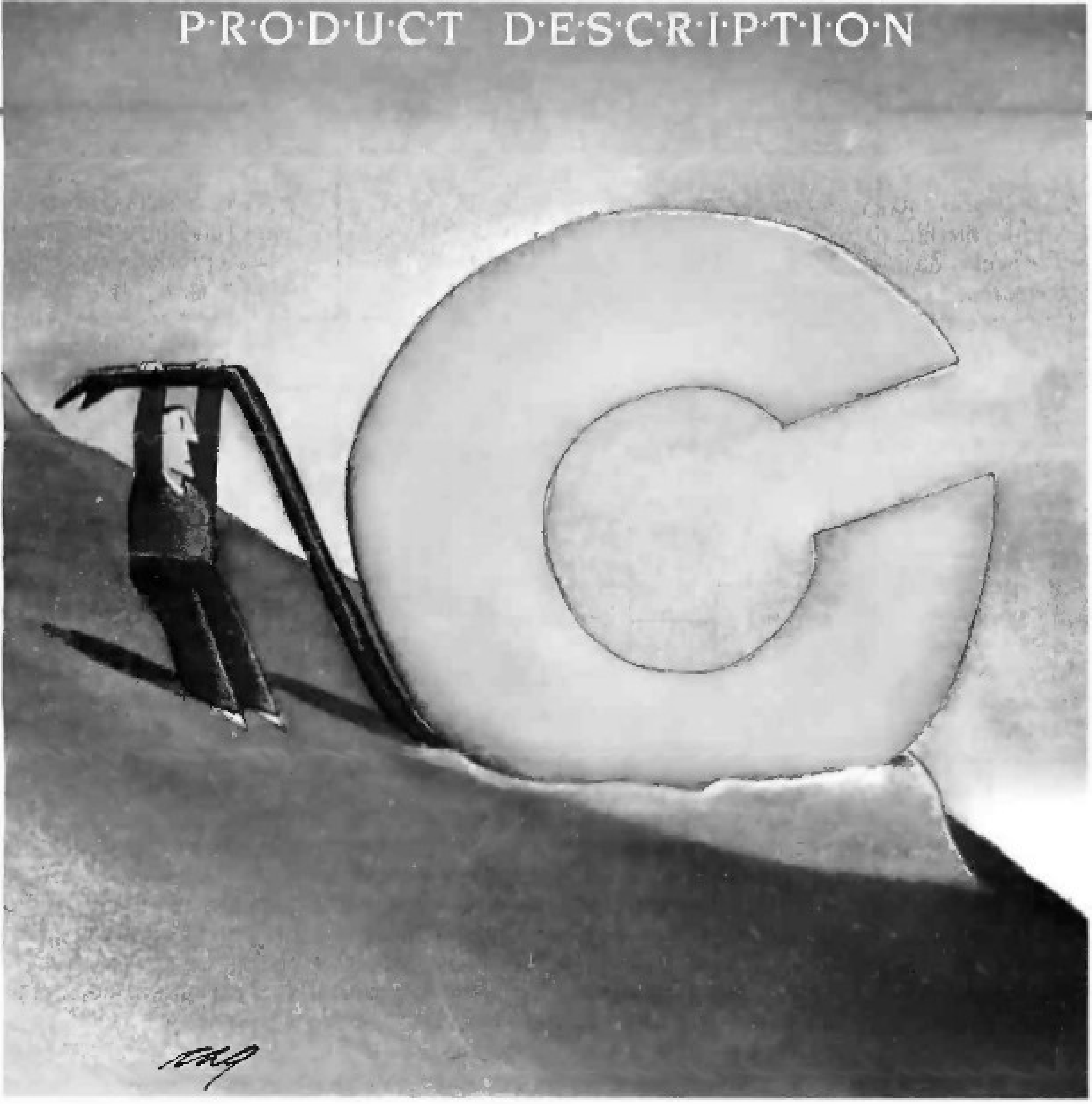

C–LANGUAGE DEVELOPMENT TOOLS

G. MICHAEL VOSE

Editor's note: The following is a BYTE *product description. It is not a review. We provide an advance look at these new products because we feel they are significant. We plan to provide complete reviews in subsequent issues.*

BASEMENTS AND GARAGES may have been the birthplace of the technology that has evolved into the modern microcomputer, but researchers at one of the world's largest corporations created the C programming language. In the eighties, C has evolved into the software development tool of choice for many serious programmers. Bell Labs' UNIX-driven language has been adapted for use with other operating systems and used to create a multitude of significant personal computer programs, including VisiCorp's VisiOn, Microsoft Windows, and the outline processor, MaxThink.

C falls into the mid-level language category—higher than CPU (central processing unit) register- and memory-manipulating, low-level assembly languages but lower than machine-isolating, high-level lan-

(continued)

G. Michael Vose *is* BYTE*'s senior technical editor for themes. He can be contacted at* POB 372, Hancock, NH 03449.

ILLUSTRATED BY RICHARD GOLDBERG

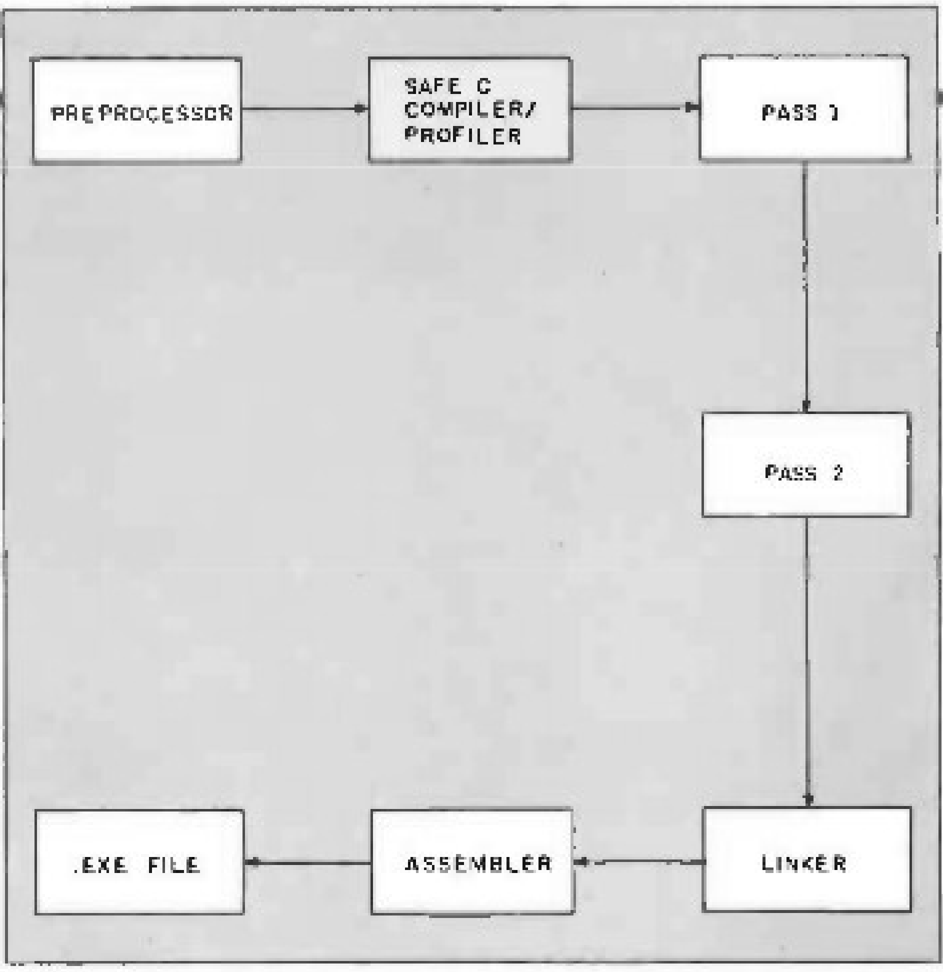

Figure 1: *Adding a preprocessing pass to a typical C compile cycle.*

guages like Pascal and Ada. Mid-level languages provide easy access to memory and the CPU while retaining the control and data structures of classic high-level languages. This access enables bit shifting and manipulation using a rich set of operators for incrementing, decrementing, and performing Boolean operations.

While C grew in popularity, many programmers agonized over its complexity and the lack of debugging tools like the monitors for tracking errors in assembly-language programs, utilities that enable you to set breakpoints in the code and use single-step execution. Outside of the UNIX realm, where C debugging facilities are part of the operating system (which is itself written in C), programmers faced a barren landscape.

In microcomputerdom, however, where entrepreneurial fervor thrives, vacuums do not exist for long. Recent months have brought forth a number of new C-language development tools and debugging aids. Many of these tools were created specifically for the microcomputer programmer.

In this product description I'll look at three of these products: the Safe C Compiler and Profiler from Catalytix Corporation of Cambridge, Massachusetts; the Instant-C interpreter from Rational Systems Inc. of Newton, Massachusetts; and the C Source Debugger from Mark Williams Company of Chicago.

Safe C Compiler/Profiler

One of the more significant problems a C programmer faces is run-time error detection. Running a C program in the early stages of its life cycle resembles a Cessna pilot trying to land a 747—he might know how to fly but isn't sure which are the right controls.

Unlike Pascal compilers, C compilers do not provide extensive run-time error checks during compilation. C compilers cannot check for array indexing errors, divide-by-zero, and similar run-time errors. C programmers have no tool to verify the correctness of argument data types passed to functions or to detect the existence of a stray pointer. Dangling pointers can cause system crashes on many microcomputer implementations of C.

The Safe C Compiler/Profiler is a checkout compiler designed to uncover run-time errors. It is not a production compiler; it is simply a development tool designed to increase programmer productivity (while decreasing the programmer's frustration level).

Safe C adds a source-to-source preprocessing pass to the normal C compile cycle (see figure 1). Safe C inserts checkout code into the original source file and then compiles the resulting file. The checkout code swells the original source in size by a factor of two or three.

As the compiler compiles the modified source file, the checkout code prompts the generation of error messages for every possible run-time error encountered. Multiple errors generate a stream of error messages. Safe C error messages are warnings only—the compiled code is executable, and the programmer must decide whether to heed the warnings generated.

The error conditions detected by Safe C include

- out-of-bounds array indexes
- arithmetic overflow and division by zero
- overflow in standard string routines
- mismatch of actual and formal function parameters
- misuse of standard I/O (input/output) routines
- indirection through and stack corruption by stray pointers

The compiler also reports standard syntax errors.

Probably the most important function the Safe C Compiler provides is detecting mismatches in function parameters. C's main strength derives from its modularity. Ideally, C programmers write their programs as small, individual units that are later linked together. The potential for error among the interacting parts is high, mostly due to the passing of arguments of the wrong data type.

The Safe C Profiler is a dynamic software development tool that provides both function and statement execution counts. When a programmer needs to know where a program spends most of its time, the Profiler provides a count of the times a function is called and, within functions, the number of times a

(continued on page 382)

(continued from page 120)

statement executes. The Profiler even generates a histogram showing the percentage of time the program executes a function or statement.

By revealing which parts of a program execute most often, the Safe C Profiler points the programmer to the code where optimization yields the greatest results. For example, in a function where multiple if . . . else tests are made, the test executed most often should appear first for optimal performance. The Profiler identifies this code for proper alignment by the programmer. This process is analogous to a delivery service analyzing all its truck deliveries to ensure that the truck doesn't backtrack when delivering goods.

The Safe C Profiler also profiles code that does not get executed. Often this exposes subtle errors, such as using a single = instead of the equivalence operator ==. This error, syntactically and lexically correct, could be hard to find unless you could show that a certain test was never being made—an indication that a bug existed. Profiling identifies code that hasn't been tested by repeated program execution, giving the testing phase of program development a new dimension.

Using these tools is similar to using a standard compiler. Catalytix's products are tailored to existing compilers so that the appearance of both is the same. If you work with the Lattice compiler, for example, the Safe C products designed to work with this compiler look identical to it.

The Safe C Compiler/Profiler is available for the IBM Personal Computer (PC), its compatibles, and computers from at least 17 other manufacturers. Prices range all the way from $400 for an MS-DOS version to $4000 for minicomputer versions.

Catalytix will shortly announce a C interpreter and an English-to-C/C-to-English translator, products BYTE will preview.

INSTANT-C

The value of interaction with programming languages is evident in the recent consumer release of an interpreted Pascal (see "Macintosh Pascal" in the June BYTE, page 136) and the success of Turbo Pascal, a fast compiler that provides the illusion of interpretation for small programs.

Several attempts at writing an interpreter for C have failed within the university community, but at least two commercial attempts appear headed for success. One is the Catalytix product mentioned earlier; the second is Rational Systems' Instant-C, which runs on Intel 8086-/8088-based com-

(continued)

Early buyers of Instant-C help finance its development.

puters under MS-DOS or CP/M-86 and costs $500.

Instant-C supports all standard C features except initialization, parameterized #defines, declaration in compound fields, and an assembly-language interface.

Envisioned primarily as a language-development tool, Instant-C features a compiler, an interpreter, a full-screen editor, a linker/loader, a library of C functions, a source debugger, a UNIX-like lint utility (that checks for the number of arguments passed to a function and for external variable declarations), and a source-code formatter.

Although Instant-C has been advertised since midsummer, as of this writing (October) the product is still under development. A potential buyer receives an explanation of this situation; if he decides to buy, he gets the latest version (in my case version 0.88) with the promise of subsequent versions as they become available. This means that early buyers of the product help to finance its development. The people at Rational Systems deserve credit for being honest with their customers about this unusual marketing technique.

To use Instant-C, you load the interpreter and can subsequently enter and execute any valid C-language code. You can call library functions or user-written functions stored on disk. Typing a function's name, including arguments where they are necessary, calls the function. The interpreter cannot use compiled code but must have source files for interpretation.

The most obvious use of the interpreter for execution of individual commands or library functions is for instructing the novice C programmer.

Experienced programmers will use a different scheme. From the interpreter, typing ed *filename* places you in the editor to create a new function or edit an existing function. The full-screen editor offers reasonable functionality and an attractive display area. On the IBM PC, the editor uses the cursor, insert/delete, and page-move keys.

Within the editor, you write C source code in the normal manner. When you execute the editor's exit command, the source code is partially compiled and, if there are no errors, you return to the interpreter. Calling the name of the function just written executes the code, providing an im-

(continued)

mediate way to verify logical correctness.

Checking of syntax errors is handled by a compiler pass made when exiting the editor. If errors are discovered, an error message appears on the top line of the screen and the cursor drops back within the source code to the location of the error. Errors are uncovered one at a time until the source file is syntactically clean.

When I examined Instant-C, the debugging facilities were just being defined and implemented. The debugging commands include

back—show all the functions called to a breakpoint
go—resume after breakpoint
local—enter a function for local execution
trace—set breakpoint functions
untrace—turn off tracing
reset—turn off breakpoints

These facilities implement well-known debugging techniques.

Rational Systems said it expects to complete and ship version 1.0 of Instant-C by November 15. The program requires 256K bytes of memory, but 384K bytes is recommended for developing programs of any length.

C Source Debugger

Mark Williams Company, purveyor of the UNIX-like Coherent operating system, introduced its MWC-86 C compiler and an accompanying C Source Debugger on September 1. The products work together and sell as a package for $495. In this description, I'll look only at the debugger.

The C Source Debugger (called by the company, in typical UNIX fashion, dbc) can debug your C programs in C instead of in machine language. dbc does not add code to your programs and enables you to view the source code as you debug. dbc can provide

- a trace of the execution of any statement with or without breakpoints
- a display of the value of variables and expressions during program execution
- single-step execution of code
- separate windows for source, program output, history, and evaluation of expressions
- exploration of the stack

These features make use of a simple user interface that relies heavily on function keys and page-move keys.

Using dbc resembles using an assembly-language monitor. Instead of looking at representations of the computer's memory and central-processor registers, you look at C source code and a variety of windows that show program output and evaluation of variables and expressions. You can execute the code with or without breakpoints, backtrace through the code, single-step through the program, and track the changing of variables and expressions. You can also record a history of your debugging sessions, available in the separate history window.

The programmer records errors and changes that need to be made to the source code. You can make changes to the source within the debugger, permitting a way to test different strategies for solving a problem; however, the changes cannot be saved.

You can set breakpoints (*tracepoints* in the lexicon of dbc) to halt execution when a line of code executes or when the value of an expression changes. Programmers can toggle tracepoints on and off. You can execute programs with tracepoints through to the end with tracepoints overridden, but a trace history is listed in the history file.

The program window displays the output of your program as if it were executing without dbc. This window is saved and restored when the programmer switches between windows.

The underlying theory behind dbc is analogous to building a car's engine block out of clear plastic so that mechanics can watch the internal engine parts function. If something goes wrong, the mechanic can spot it and then later go inside the engine to make repairs.

The C Source Debugger, like the MWC-86 C Compiler, requires 256K bytes of memory. ■