Four C Language Interpreters

John Unger

An overview of Run/C, Run/C Professional, C-terp, and Instant-C

Run/C version 2.03 and Run/C Professional version 1.03 by Lifeboat Associates ($120 and $250, respectively), C-terp version 2.22 by Gimpel Software ($298), and Instant-C version 2.20 by Rational Systems ($495) are four C interpreters for MS-DOS machines. The packages are designed to accomplish the same general tasks, but the approaches they take and how they interface with the user are all distinct.

Run/C and Run/C Professional are essentially the same program. The major difference between them is that the Run/C Professional package includes the ability to load external object libraries and code. Because they differ only in this respect and because their interfaces, much of the content of their manuals, and their benchmark results are identical, throughout most of this review I will use one term, Run/C, to refer to both of these interpreters.

All four software packages are classified as interpreters. Instant-C can also produce a stand-alone executable file. However, only Run/C executes programs in a mode similar to most BASIC interpreters—interpreting each line of source code on a line-by-line basis without producing any intermediate code. This situation accounts for the relatively slow speed of Run/C in the benchmark tests. The other interpreters feature a compile mode, in which the source code is compiled internally into an intermediate code and then is run from this compiled form.

Instant-C and Run/C allow you to operate in a direct mode, which is similar to BASIC. This means that you can enter any valid C expression, including calls to library functions, while in the interpreter, and the result will be evaluated and displayed on the console.

Hardware and Software Requirements

The four interpreters are designed to run on the IBM PC, XT, AT, and compatibles and require at least one 360K-byte floppy disk drive for loading their software. All four packages run on MS-DOS 2.0 or higher.

Each of the interpreters requires a lot of free memory. C-terp requires 256K bytes of RAM, Run/C requires a minimum of 320K bytes, and Instant-C needs at least 512K bytes. None of the interpreters uses overlays. This means the entire interpreter, editor, library, and other required modules are all part of one executable file and load into memory at run time. Access to the various parts of the interpreters is fast, but your computer must have sufficient memory for the operating system, the interpreter, and your program. The maximum size of the source code you can load into any of the four packages is usually limited only by the amount of RAM you have. Run/C can handle a maximum of 640K bytes of RAM, C-terp can handle up to 16 megabytes of virtual memory (you activate this memory option by a command when you start the program), and Instant-C can manage a maximum of 1 megabyte of nonextended RAM, or an additional 64K bytes of expanded Lotus/Intel/Microsoft (LIM) memory.

All four interpreters use C syntax that is compatible with the Kernighan and Ritchie definition of the language. You can set up Run/C Professional, Instant-C, and C-terp to use libraries specific to either Microsoft or Lattice C compilers. C-terp also comes in versions that are compatible with Manx Aztec C, Computer Innovations' C-86, and Mark Williams' C.

None of the interpreters is copy-protected. All four user's manuals urge you to make backup copies of the distribution disks immediately. I had no problems copying any of the programs to a hard disk.

Overview of the Interpreters

The authors of Run/C strived to make their C interpreter look like the environment familiar to users of Microsoft BASIC interpreters. Most of the commands closely follow those used in BASICA and GW-BASIC. The in-memory full-screen editor has many of the features and uses the same keystrokes as the popular WordStar editor.

C-terp moves further away from familiar patterns and provides an integrated environment of a menu-driven command level, a powerful screen-oriented editor with line numbers, and a debugger.

Instant-C is the most sophisticated of the four packages. It provides all the features of the other interpreters plus a few extras. Instant-C lives up to its name as far as speed of program compilation and execution is concerned; it was by far the fastest in the benchmark tests.

All four packages are designed to execute C source code only from within the confines of their individual interpreters to produce the desired output. What good are they, then? I think that there are two principal roles they might play: as a tool for learning how to program in C or as a tool for quickly writing portable C programs that you will later compile into executable programs. The least expensive package, Run/C, is especially good if you're a beginner in C and are familiar with the screen layout and operation of GW-BASIC or BASICA.

All four packages' user's manuals provide good descriptions of how to get the programs running on your computer and take you step by step through a simple example program. Each manual provides

continued

John Unger (P.O. Box 95, Hamilton, VA 22068) is a geophysicist for the U.S. government who uses the C language to write software for earthquake research.

	RUN/C version 2.03 RUN/C Professional version 1.03	C-terp version 2.22	Instant-C version 2.20
Type	C language interpreters	C language interpreter	C language incrernental compiler
Company	Lifeboat Associates 55 South Broadway Tarrytown, NY 10591 (914) 332-1875	Gimpel Software 3207 Hogarth Lane Collegeville, PA 19426 (215) 584-4261	Rational Systems P.O. Box 480 Natick, MA 01760 (617) 653-6194
Format	Two 5¼-inch floppy disks	Two 5¼-inch floppy disks	Three 5¼-inch floppy disks
Computer	IBM PC, XT, AT, or compatible with at least one 360K-byte floppy disk drive and 320K bytes of RAM running DOS 2.0 or higher	IBM PC, XT, AT, or compatible with at least one 360K-byte floppy disk drive and 256K bytes of RAM running DOS 2.0 or higher	IBM PC, XT, AT, or compatible with at least one 360K-byte floppy disk drive and 512K bytes of RAM running DOS 2.0 or higher
Documentation	530-page user's manual	148-page user's manual	520-page user's manual
Price	Run/C: $120 Run/C Professional: $250	$298	$495

you with a listing of the files on the distribution disks and describes the ones you will need to run the interpreter. All the manuals have tables of contents and indexes to help you locate specific subjects.

The amount and usefulness of the documentation supplied with the interpreters vary considerably. Run/C has the best manual of the lot. Nearly 400 of the manual's 530 pages contain descriptions and meaningful examples of how to use all the 115 library functions included with the interpreter. This is in sharp contrast to C-terp's 66 functions, which are described without examples in 50 pages of its user's manual. Instant-C's manual contains almost 300 pages of terse descriptions of its extensive collection of 162 library functions and has only a few short examples.

Although complete at 430 pages, the Instant-C manual is difficult to use and confusing, mostly because it is difficult to locate specific information on the interpreter's commands and functions, despite the manual's index. Instant-C is a large, sophisticated program with no on-screen help. Fortunately, a well-written chapter in the manual that contains an example showing how to create, edit, and debug a program helps to overcome some of these difficulties. Anyone starting out with Instant-C should read this chapter first.

The Run/C Interpreter

When you start Run/C, you are greeted by a familiar-looking screen layout that mimics that of Microsoft BASIC: 10 highlighted boxes appear across the bottom of the screen to show the commands that are assigned to the function keys. Run/C's editor also has easily accessible help screens that are just a keystroke or two away.

You use the `LOAD` command to get an existing file from your disk. The `FILES` command allows you to see the files in the default directory. These and similar BASIC-like commands make it easy for someone who has used BASIC to get started with Run/C. Run/C's debugging aids consist of the `TRON` and `TRACE` commands, along with the ability to `DUMP` the values of variables after halting the program with Control-C or Control-Break. The `DUMP` command displays these values on the computer's screen; a corresponding `LDUMP` command dumps the values to a printer.

The aspect of Run/C that I found most bothersome was its rather slow execution speed. You can't do much about this, except to use the program on a faster microcomputer. Run/C also has sluggish disk I/O that makes the interpreter a bit tedious when loading large files from or saving them to disk. Beginners may not become as frustrated with Run/C's slow speed as I did, but I think it's fair to say that the slow execution of source code will render Run/C unfit for use by experienced programmers. [Editor's note: *The latest version of Run/C Professional is 1.11. This new version offers support of the Microsoft C 4.0 compiler's features.*]

C-terp

C-terp first appears on the screen as a menu, giving you a choice of 14 operations. Most choices are obvious, such as `LOAD`, `COMPILE`, `EDIT`, or `RUN`. To choose a command, you need only type its first letter, which is highlighted on the menu. A normal sequence would be to load a file with C source code from disk, edit it to make changes or just to preview the code, and then compile and run the program. Each of these operations takes just one keystroke, with the exception of typing in the name of the file.

The screen-oriented editor is fast and has most of the features that programmers have come to expect from editors, including block moves and copies, but it lacks macro commands and auxiliary buffers. An Alt-H keystroke combination instantly displays a helpful menu of editor commands on the screen. You use line numbers for the editor, and they act as a handy reference for the debugger and error messages. The line numbers are not saved as part of the file.

C-terp allows you to set breakpoints anywhere in your source code by use of a `breakpt( )` function. This function serves as an entry point into the debugger. In the debug mode, you can display or change the value of any active variable and then continue execution of the program. A split-screen feature shows the source code in the top half of the screen, with the debugger commands and their results in the lower half. You can also flip back and forth to a screen showing the program's output or browse through the source code without ever leaving the debugger.

A major shortcoming of C-terp is its lack of a built-in library of mathematical

functions, such as sine, tangent, or square roots. Normally, C is not looked upon as a language for writing scientifically oriented software, but I have used it for this purpose and routinely write scientific programs in C rather than in FORTRAN. You can overcome this lack of mathematical functions if you have access to C source code for the math functions you need, and you can include this code with your program.

Another alternative is to use an object code math library that is compatible with C-terp. If you already use a C compiler, this solution is easy because C-terp comes in versions that are compatible with Manx Aztec C, Lattice C, Computer Innovations' C-86, Microsoft C, and Mark Williams' C. With the Microsoft-compatible version of C-terp that I tested, I was able to create a version of C-terp that used all the Microsoft compiler's library functions. This feature allows you to use the interpreter to test programs that you can later generate into executable files with your compiler. [Editor's note: *The latest version of C-terp is 2.30B. It now supports the keyword* `far` *for large memory-model addressing, as well as ANSI extensions to the C language, such as function prototyping.*]

Instant-C

Instant-C is unique among the interpreters in that it gives you the capability to create stand-alone executable files. However, the executable module produced includes the entire Instant-C library code (which is larger than 32K bytes), along with all the storage allocated by Instant-C to your source code during the debugging session, unless you follow special procedures.

Instant-C has a variety of debugging aids and offers elaborate control over execution of your program. As with C-terp, you can set breakpoints anywhere in the source code with a `__( );` function, and you can examine and change variables once the debugger is entered.

This interpreter was difficult for me to understand and use, compared to the others. First, most of the user's manual is hard to read; for example, it informs you that you can edit a *function* (e.g., `main( )`) only after you have loaded the *file* into memory. However, this vital bit of information lurks in a section entitled "Style Differences," which follows the introductory section on the editor. Furthermore, the description of the `#load` command is located in yet another section entitled "The Instant-C Work space."

Second, the program itself is distinctly user-unfriendly and has no on-screen help; you must rely on the manual and your memory. After the program is invoked, the only thing that appears on the screen is a # prompt, not a menu. Instant-C has a total of 65 valid commands that you can enter from the interpreter mode. This gives you a great deal of control over editing, testing, and debugging your code, but it also gives you a great number of keywords to remember.

The program is particularly powerful for debugging and provides the programmer with a good set of tools for unraveling problems. However, Instant-C's true forte is the speed at which it executes C programs; it is clearly in a class by itself in this category. Instant-C also includes the largest library of built-in functions for programmers to take advantage of. For example, it has a number of commands for displaying memory in hexadecimal, octal, or decimal format. The math library provides exotic transcendental functions, such as hyperbolic cosine. A `setjmp` function is provided, as well as interrupt support to invoke MS-DOS functions.

Performance Evaluation

Because both C-terp and Instant-C compile the complete source code into an internal form before executing the program, I expected to see large differences in execution time between them and Run/C, which operates as a true interpreter. Clearly, as shown in the benchmark results in table 1, Instant-C is the fastest of the four, with C-terp in second place. Run/C was the slowest in all the benchmarks. Run/C and Run/C Professional performed virtually identically in the tests. To give you an idea of how the C interpreters compare to a standard C compiler, I have included the results of the benchmarks run on version 4.0 of the Microsoft C compiler. I ran the benchmark programs as functions called from within a `main( )` function. The `main( )` function included code to start a system timer, run the benchmark program, and then read and display the elapsed time.

However, the times shown in table 1 aren't the only factors to consider when choosing an interpreter. Programmers spend much of their time editing and debugging source code that they have created; usually only the end user gets the benefits of fast execution. The speed with which a programmer can produce C language code that executes flawlessly is clearly the important benchmark to consider if you choose to operate in an interpretive environment. On the other hand, the speed with which you can learn C syntax and instructions and begin to write real C source code is a key benchmark measurement for someone trying to learn the intricacies of the C language. In this sense, Instant-C is not as clear a winner, nor is Run/C as clear a loser, as the benchmark times would indicate.

As an attempt to compare the four interpreters in a fair way, I used the scenario in which you finish entering a program using the interpreter's editor and then proceed to debug and run the source code. First, I took a simple C program (see listing 1) that ran identically on the four interpreters. I then introduced two types of common errors into the source code. Table 2 gives an explanation of the error types used. I will refer to code in the program by line number and to error types by their code names, for example, S2 for syntax error number 2.

For error S1, Run/C prints out an error code, the offending line number, and the message `Variable or function fohr not found`. The compile-time errors S2 and S3 frequently are not obvious to compilers or interpreters until a line or so later or, in the case of a missing left brace, until the end of the program. Run/C did not handle error S2 well at all,

continued

Table 1: *Benchmark times for the interpreters and the Microsoft C compiler (for reference). Times are in hours:minutes:seconds. The Sieve test measures how long it takes to run ten iterations of BYTE's Sieve of Eratosthenes prime-number benchmark on an array of 8190 numbers. Sort tests each compiler's handling of pointers during a sorting operation. Fib tests the efficiency of each compiler's recursion while computing a Fibonacci series. Float measures the time it takes to do 140,000 floating-point multiplication and division operations. Fileio exercises the I/O functions of the compiler by reading and writing to a 65,000-byte disk file. An IBM PC with 512K bytes of RAM was used for the benchmark tests.*

	Microsoft C 4.0	Instant-C 2.20	C-terp 2.22	Run/C 2.03
Sieve	11	40	59:37	5:29:33
Sort	20	1:10	46:27	3:53:26
Fib	1:16	3:49	3:04:52	15:15:28
Float	1:33	8:09	16:29	51:58
Fileio	7:20	10:49	23:10	1:21:58

Table 2: *Type and location of the errors that were introduced into the program in listing 1. The first three, S1, S2, and S3, are simple syntax errors. The latter two, L1 and L2, are errors that can be detected only while the program is running.*

Error code	Type of error	Location in listing 1
S1	misspelled variable name	fohr for fahr—line 11
S2	missing ;	end of line 11
S3	missing { and }	lines 10 and 13
L1	integer instead of floating-point division	line 11
L2	integer instead of floating-point format in printf statement	line 12

Listing 1: *A simple program that demonstrates how the three interpreters handle syntax and run-time errors. The lines have been numbered for referencing in the text.*

```
 1  #define UPPER    300.0    /* upper limit of Fahrenheit degrees */
 2  #define LOWER    0.0      /* lower limit of Fahrenheit degrees */
 3  #define STEP     20.0     /* step to increment Fahr. degrees   */
 4
 5  main( )
 6  {
 7      float fahr, celsius;
 8      printf("\t\tFAHRENHEIT    CELSIUS\n\n");
 9      for (fahr = LOWER; fahr <= UPPER; fahr += STEP)
10      {
11          celsius = (5.0/9.0) * (fahr-32.0);
12          printf("\t\t%4.0f\t\t%6.1f\n", fahr, celsius);
13      }
14  }
```

and it produced the message `Required lvalue not found`. S3 produced the message `**Error: unmatched braces**` with the line number of where the last left and right braces were found. The only way to untangle run-time bugs like L1 and L2 is with a debugger or by sprinkling calls to the `printf` function throughout the source code to examine the values of variables. With the L1 error, the program compiled well and listed the values of `fahr` correctly, but the output of the variable `celsius` was all 0.0s. For L2, the output of both variables was 0. Run/C has two ways of debugging run-time errors: You can turn on a `TRACE` command toggle, or you can interrupt the program with a Control-C. This will interrupt Run/C and print the menu `Continue, dump, ldump, interactive, help or end (C/D/L/I/H/E)?` on the screen. The dump option lists all the variables and their values from three different areas: automatic variables declared in the function where the break occurred, automatic variables from the calling function, or all global and static variables. Prior to running the program, you can turn on the `TRACE` toggle, which prints out the current values of all the variables each time they are referred to as the program runs. Run/C also has a `TRON/TROF` command pair, which is similar to BASIC, to trace program logic and aid in debugging. These error messages and debugging aids are adequate, but a cut below what C-terp and Instant-C offer.

Both C-terp and Instant-C handled syntax errors easily and in a similar manner. For the S1 error, C-terp prints an error code and the message `Undeclared identifier`. Hitting any key puts you back into the screen-oriented editor with the cursor at the first letter of the word that caused the error.

C-terp handles both the S2 and S3 errors very well. For the S2 error, it gives the message `Expecting ';'` and places the editor cursor at the first character in the line following the omitted ;. For the S3 error, it gives the message `expecting 'identifier'` for a missing left brace and the message `expecting '}'` for a missing right brace . As with Run/C, the L1 error compiled but produced 0.0s for the variable `celsius`; I obtained similar results for L2. This was a good place to try C-terp's `breakpt( );` function, so I inserted it within the `for` loop after line 12. The effect of having this function in the loop was that execution stopped just after the `printf` function was called, and the program automatically exited to the debugger screen.

In C-terp's debugger environment, the top two-thirds of the screen contains a listing of the program's source code where the `breakpt( );` function is set. The bottom part of the screen displays a menu of commands for displaying variables, tracing, or stepping through code. You can select the appropriate command by typing its first letter. For example, hitting the *D* key selects the `Display` option and causes the debugger to print the prompt `expression:`, to which you can respond with the name of any valid variable and press Return. The current value of that variable is then displayed.

Instant-C makes it virtually impossible for you to leave the editor without compiling your source code function in its most recently edited form. The compiler handles all syntax errors logically and with somewhat more chatty messages than the other interpreters. For example, error S1 results in the message `I'm sorry but I don't know the word ''fohr''` and the cursor was placed at the first character in `fohr`. For error S2, it displays the message `Missing semicolon (;) before ''printf''` and the cursor was placed at the first character in `printf`. S3 produces the message `I'm sorry, but I don't know the word ''{''` for a missing left brace and `Missing closing brace (})`; `possible unterminated remark` for a missing right brace. As with Run/C and C-terp, errors L1 and L2 generated 0.0s for the output.

You can track down run-time errors such as L1 and L2 with a variety of debugging tools. Instant-C's special `__( );` function serves as a breakpoint to halt program execution and is similar to C-terp's `breakpt( );` function. When you break out of a running program in Instant-C, the fragment of source code surrounding the `__( );` appears in a window at the top half of the screen, and you are back in the command mode with a # prompt in the bottom half. From this prompt, you can issue special commands to view and change variable values and list the source code or reenter the editor mode and view the source code. But you cannot browse through the source code, as is possible in C-terp's split-screen debugger, nor do you have any on-screen aids to help you choose commands.

Technical Support

The support I have received from Lifeboat Associates, Gimpel Software, and

continued

Rational Systems has been excellent. All my questions have been answered quickly, accurately, and, in many cases, by the person who actually wrote the program. None of the companies has a toll-free number for technical help, but, instead of that convenience, you get a much more important one: no long waits or being told that someone will call you back. It is simply courteous, quick service. This situation may change as the number of users for the software grows, but I hope not.

Rational Systems has a service for Instant-C that I've never seen before with any other software I've used. It includes a stamped, self-addressed envelope with the documentation for you to use to send back bug reports or questions. Also, Rational Systems offers a money-back guarantee for the first 31 days you own the software.

Which to Choose?

Without a doubt, the best C interpreter I can imagine would combine features found in all four software packages reviewed here. First, it would have the documentation and editor of Run/C. Second, it would present you with an easily accessed menu-driven environment and an easy-to-use symbolic debugger like C-terp's. Third, it would have the speed, sophisticated features, and extensive library of Instant-C.

Of the four interpreters, C-terp comes closest to this ideal right now. Its only major shortcomings are the lack of an in-program math-function library and the terseness of some of its documentation.

Run/C is an excellent package for beginning C programmers. I would recommend it particularly to those who are familiar with BASIC and who won't be put off by the slow execution speed. Run/C's low cost makes it especially enticing. Run/C Professional, with its ability to load external object libraries and code, adds more power to the basic Run/C package but does not improve on its execution speed.

The performance of Instant-C is excellent. However, the problems I had understanding how the software worked and its awkward documentation detract from the usefulness of the package. I cannot recommend Instant-C to beginning programmers, but experienced C programmers would appreciate and utilize its sophisticated features and power.

A big plus for C-terp and Instant-C is their ability to create versions of the interpreter that are completely compatible with the libraries of popular C compilers. In the case of Run/C Professional, you at least have the option to load and include different object libraries with the programs you are creating. ■