Programming in C: Speaking the Amiga's Language

By Sheldon Leemon

You might say that the Amiga was responsible for introducing me to the C programming language. In 1984 I was fortunate enough to see an early prototype of what was then being called the Amiga Lorraine. I was of course impressed with the graphics and sound demonstrations. But I was even more impressed with the fact that those programs had been written in a high-level language called C. Up until that time, I had assumed that you just couldn't get that kind of performance out of a microcomputer without resorting to tedious assembly-language programming. Although I had heard a little about C in the past, I suddenly wanted to know a lot more.

Unix

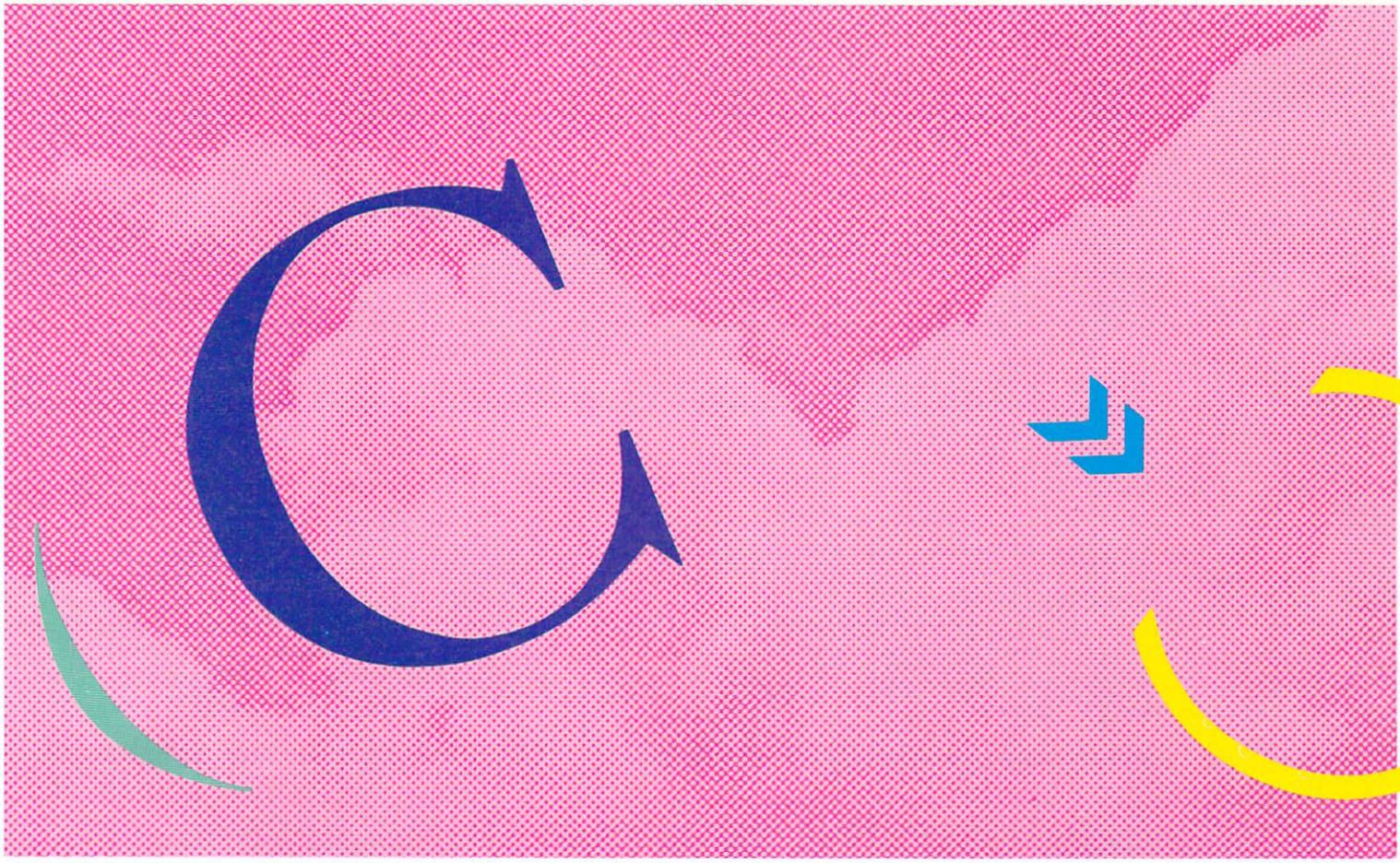
Even as late as a year and a half ago, however, there was not much information available for programming in C on microcomputers, at least not in books that appeared in my local bookstore. That's because C was designed as part of the Unix operating system, which is used mostly on mainframe and minicomputers. (Unix itself is written in C.) Though Unix is a very powerful and flexible operating system (developed by AT&T's Bell Laboratories subsidiary), it has not gained widespread acceptance for use on microcomputers as of yet. For one thing, Unix has a reputation for being so complex that only programmers who work with it day after day really get to know and like it. For another, the Unix operating system usually includes a host of utility programs, making the size of the complete system far larger than those commonly associated with microcomputers.

Language of Choice

While microcomputers have not advanced quite far enough to reach the point where they comfortably support Unix, they have certainly reached the point where they are ready for C. With older 8-bit micros, there are two factors that make assembly-language programming almost a necessity. First, these processors are relatively slow, so only machine-language programs run quickly enough to achieve acceptable performance. Second, these machines are limited to 64K of memory (without using tricky bank-switching techniques). Thus, they require the compact coding that can only be achieved through the use of assembly language.

When the IBM PC was introduced, it may not have represented a startling technological breakthrough, but it did at least present a system that used a slightly faster microprocessor and that allowed for more RAM and larger mass storage than was previously available. On the software front, this meant that developers had a more fertile environment in which to develop programs that were more powerful and easier to use.

Writing large, full-featured programs on the order of Lotus 1-2-3, however, is a massive undertaking. While assembly language affords good performance and com-



pact code size, it is very difficult and time-consuming to develop extremely large machine-language applications, and once such applications are developed, it is difficult to maintain and update them. What was needed was a language that offered some of the advantages of a high-level language, such as shorter development time and ease of maintenance, and yet produced programs that could fit in a 256K machine and offer good speed. For an increasingly large number of software developers, the C language seemed to offer the best range of features. As a result, C has become the language of choice for software development on the IBM PC and similar microcomputers.

Features

The C language has many features that make it a suitable language for software development. It is a modern, structured language. Its design philosophy is based on the use of subprograms called functions. Each function that you create is a small, self-contained program that performs a particular task. This allows the programmer to break down the overall task into manageable modules. Each module can be independently tested and debugged; when perfected, it can be incorporated into larger functions that perform more complex tasks. This modularity not only makes it easier to write and maintain programs, but it also helps to eliminate duplication of effort. If you are writing several programs that each require the user to enter some specific kind of information (such as an amount expressed in dollars and cents), you can create one general module that will prompt the user and accept the input, and use the same module in each of your programs. In fact, commercial libraries of such commonly-used modules can be purchased.

Another feature of C is the kind of output it generates. Most C compilers create machine-language code that uses the same format as assembly-language programs. Such programs may run faster than compiled languages that generate semi-interpreted code, and they do not require any special support programs to run them, making them easy to operate and easy to distribute. The programs created by a C compiler can be relatively compact. For one thing, C is not a big language. At the core level, C has only about a dozen keywords that could be considered commands. All of its input/output functions, such as printing to the display screen, are included as a standard library of functions. In some cases, it is possible to compile a program using only those functions that are actually used in the program, thus reducing the overhead requirements of the final program.

Although C is a high-level language, it works closer to the machine level than many such languages. It has a

number of operators that manipulate individual bit fields of data, which makes hardware-intensive programming easier. It also has good facilities for integrating machine language into a program to speed up those portions where time-intensive computation occurs.

Finally, C offers a fair degree of portability. Though in theory, there is not an "official" standard version of the language yet, in practice, versions of the language that are available for a wide range of computers, from micros to mainframes, are very similar. Of course, programs that include any kind of graphics generally use very hardware-specific display methods, making it hard to convert them for use on computers with different types of display hardware. Still, by isolating these display routines into a small group of distinct functions, C programmers need only to convert these functions to enable their programs to operate on another machine. This makes it much easier to convert a C program written for the IBM PC to the Amiga than it would be to convert a similar program from IBM Basic to the distinct Basic dialect used on the Amiga.

Amiga and C

In addition to the inherent virtues of the C programming language, the features of the Amiga are such that the computer lends itself particularly well to the development of software written in C. It has plenty of memory, both for creating and executing large C programs. Its use of large, special-function coprocessors to take care of time-intensive tasks, such as graphics display management and animation, sound and disk I/O, frees the main microprocessor to run at a high rate of speed. This means that some C programs which would execute too slowly on other microcomputers, will perform very well on the Amiga.

Another strong factor in its favor is that the Amiga operating system is designed to work with C programs. In fact, the operating system itself is partly written in C. One of the main features of the Amiga operating system is that it offers support libraries of functions that allow programmers to easily take advantage of its power to display and animate graphics and text, or produce sound and speech, etc. These functions are set up to mesh perfectly with applications programs written in C. In effect, C programmers can use the Amiga operating system as if it were a library of functions that are part of the C language.

Compiling C

Despite these many advantages, C is still used mostly by experienced programmers. There are several reasons why newcomers might feel intimidated by the language. For one thing, it is a compiled, rather than an interpreted, language. Using an interpreted language like Basic is a very interactive experience; it has a built-in

Lattice: The Developers of Amiga's C

When Commodore-Amiga set out to acquire a C compiler for the Amiga, they turned to Lattice Inc. While the name Lattice may not ring any bells with you, it is a familiar one to software developers. Hundreds of the best known programs for the IBM PC were written in Lattice C. Among them are dBase III, Wordstar 2000, the Smart Software System, the Perfect family of programs and the Sorcim/IUS line of programs, which includes SuperCalc3.

Beginnings

Lattice was founded in 1981, but its three principals, Steve Hersee, Francis Lynch and Dave Schmitt, had known each other for several years prior to its inception, having worked together in the software field. The company started with a compiler for minicomputers, and in 1982 released its compiler for the IBM PC. Though Lattice was itself a small company, the product was soon picked up by both Lifeboat Associates and Microsoft, who were able to give it widespread distribution. The IBM PC version of the compiler has won broad critical acclaim from the computer press, and its users currently number more than 30,000.

Steve Hersee, Director of Marketing for Lattice, attributes its success to what he calls "ego-less programming." "We're not interested in proving that we can write better code than anyone else," he says. "Our primary concern has always been meeting the needs of our users."

Although best known for its IBM PC compiler, Lattice has also done specialized software development for Tandy, Sony, Texas Instruments and IBM, among others. Nor is it a stranger to 68000-based systems such as the Amiga, having already developed C compilers for computers running under the CP/M-68K operating system, and for the Sinclair QL, a 68008 system popular in Europe. The compiler that Lattice developed for the Amiga is thus a mature product, even though it was written for such a revolutionary new computer.

The Family that Works Together

Hersee emphasizes that Lattice has developed a family of products that work together, numbering 38 in all. He is quick to point out that Commodore commissioned Lattice not only to write the native Amiga version, but also to develop Amiga compilers for the Sun Microsystems and Stride 68000 development systems and a cross-compiler that runs on the IBM PC. "This is significant," he said, "because it shows that Commodore is interested in supporting the whole spectrum of developers, from the smallest to the largest."

All of these versions of the Lattice compilers function identically, so developers who are familiar with any of them will have no problem using the Amiga version. Lattice has also made efforts to ensure that besides being internally consistent, its line of compilers

editor, and after you enter a line of code you need only type RUN to see the program execute. Some Basics even provide syntax checking on entry, so that you get instant feedback if you make a typing mistake. After running the program, if it doesn't function properly, you just list the bad lines, make some changes and run the program again.

With a compiled language, you must first compose the source code using a text editor. Then, you use the compiler to change the source code to object code. Finally, you use a linker program to convert the object code to an executable format. If an error occurs at any stage (due to a typographical error in the source code, for instance), the whole procedure must be repeated again until the compilation process is successful. Only then can you run the program to determine whether or not it does what you want it to do. If it doesn't, you've got to load your text editor and try again. This is a far cry from Basic, where you type PRINT "HELLO", and the computer simply does it.

Newer is Nicer

Lately, however, C has been made a lot more friendly for the less-experienced programmer. Very sophisticated program text editors are now available, some of which can even perform syntax checking, so that you don't have to wait until compiling time to discover syntax errors. Some interpreted versions of C have also appeared. These allow a programmer to develop programs in an interactive environment that is much more like Basic. The difference is that once programs have been written using the interpreter, they can then be compiled and run with a speed that Basic can't touch.

The atmosphere provided by the Amiga is much friendlier to such development than that of most microcomputers, even if you choose to use a C compiler instead of an interpreter. For one thing, the Amiga is a multitasking system. So instead of loading a text editor, a compiler and a linker in sequence every time you want to make a program change, it should be possible (if you have enough memory) to have each of these programs functioning *simultaneously*. Although compiling and linking is usually a multistage process, AmigaDOS allows you to set up batch jobs that will perform the whole process automatically. Finally, if you have enough memory, you can use the AmigaDOS RAM device to put your files in a super-fast RAM disk, which will speed compiling time dramatically.

"C"ing is Believing

Despite its reputation as a language for skilled programmers, C is not much more difficult to master than a language like Basic. Perhaps the best way to convince you is to show you an example of a C program. The following are listings of two programs, one in C and one in Basic. Each produces a temperature conversion table from Fahrenheit to Celsius in steps of 20 degrees Fahrenheit, which looks like this:

0	−17.8
20	−6.7
. . .	. . .
280	137.8
300	148.9

The C program is taken from page 8 of the classic text, *The C Programming Language*, by Kernighan and Ritchie.

```
/* print Fahrenheit-Celsius table
    for f = 0, 20,. . ., 300 */

main( )
{
  int lower, upper, step;
  float fahr, celsius;

  lower = 0;      /* lower limit of tem-
                     perature table
*/
  upper = 300;    /* upper limit */
  step = 20;      /* step size */

  fahr = lower;
  while (fahr < = upper) {
    celsius = (5.0/9.0)*(fahr − 32.0);
    printf("%4.0f %6.1f\n", fahr, celsius);
    fahr = fahr + step;
  }
}
```

conform as closely as possible to industry standards, so as to afford maximum portability. They have made sure that their version of the language is in close conformity with the Unix System V standard. Hersee himself is a member of the ANSI committee that is currently working on standardization of the C language.

In addition to its C compilers, Lattice offers a number of programmer's utilities and function libraries, many of which should be available for the Amiga shortly after its release. Nor is Lattice alone in supporting its compilers with auxiliary programs. Outside vendors distribute dozens of such products, like "smart" editors that can perform program syntax entry, C interpreters that provide an interactive development environment for programs that will later be compiled and function libraries that provide the "building blocks" for applications like custom databases.

Fulfilling a Promise

While you may not be interested in programming at all, the availability of Lattice C and other C development tools, including cross compilers, for the Amiga is still very significant. Such widely used, well supported software development tools that encourage portability of programs from one microcomputer to another will spur the early production of the kind of software that fulfills the promise of the Amiga's powerful hardware.

◄ Here is the equivalent program in the dialect of Microsoft Basic found on MS-DOS machines like the IBM PC:

```
10 REM print Fahrenheit-Celsius table
20 REM
30 DEFINT L,U,S
40 DEFSNG C,F
50 REM
60 LOWER = 0
70 UPPER = 300
80 STIP = 20
90 REM
100 WHILE (FAHR < = UPPER)
110 CELSIUS = (5/9)*(FAHR – 32)
120 PRINT USING "####" ;FAHR;
130 PRINT USING "######.#" ; CELSIUS
140 FAHR = FAHR + STIP
150 WEND
```

Basic Comparison

As you can see, the two programs are not all that different. Let's compare them line by line. To begin with, you will notice that the C program has no line numbers. The format of C is very free, and a single statement can take up one line, or many lines. This allows the programmer to make the program neat and readable. The first statement, which starts with the characters /* is a remark, corresponding to the REM statement in line 10 of the Basic program. In C, the remark can extend over many lines until the closing */ characters.

Next comes the line Main( ). This defines the function named Main. C programs are made up of functions, which are small subprograms. Every C program has at least one function, called Main, with which the program starts its execution. The parentheses after the function name show that it is a function. Some functions contain the names of variables that the function operates upon (called parameters) within these parentheses, but Main( ) doesn't use any, and thus is said to have an empty parameter list.

After the name of the function comes a { (brace) character. These braces are plentiful in C programs; they are used to mark the beginning and end of functions and the beginning and end of compound statements within a function. As shown here, most programmers use different levels of indentation to help group the various pairs of braces together.

After the initial brace come two strange looking statements:

```
int lower, upper, step;
float fahr, celsius;
```

These are roughly equivalent to the DEFINT and DEFSNG statements in lines 30 and 40 of the Basic program. To tell the truth, though, those Basic lines were added more as a point of reference than anything else. Basic is not a strongly typed language; you don't really have to specify to what kind of storage class a variable belongs (although most Basics give you the option to specify that it be stored as an integer and not in floating point representation). With C, these statements are not optional. Whenever you want to use a variable, you must declare ahead of time whether it's an integer, a floating point or a text character string. These declarations are made in a block at the top of the function definition.

When we compare the body of the program, we find that there are only two major differences. (One minor change in the Basic program was to change the variable name "STEP" to "STIP", because the former is a Basic keyword.) The first is that Basic uses the Wend statement to define the end of the While statement. C accomplishes this by enclosing the whole body of the While statement within braces.

The other major difference is the way in which the results are printed. The C program uses a function called Printf( ), which is not a part of the language program, but a part of the standard library of I/O routines. This is an example of a function that takes parameters; the text and variables that appear within the parentheses make up the data upon which the function operates. It performs roughly the same task as Basic's Print Using command.

The % and "f" characters are used to specify that a decimal number is to be formatted, and the numbers 4.0 and 6.1 are used to specify that the numbers are to be printed with four digits before the decimal place and none after, and six digits before the decimal place and one after, respectively. The Basic Print Using templates "####" and "######.#" do roughly the same thing. The C Printf( ) function allows for multiple substitutions, while separate Basic statements are required for each formatted column.

Not Hard to "C"

It should be clear from the above example that once you get past the formal requirements of function names, the braces and declaring variables, C is not as alien as you might have thought. This is not to say that C is just Basic in another disguise. There are a number of powerful features found in C that are quite different from Basic. However, there are enough similarities so that the beginning programmer can get started and can take in the differences little by little.

For most Basic programmers, these added features will be quite welcome. For example, C has a multitude of powerful math and logical operators. The statement Fahr + = Step; may be less recognizable than Fahr = Fahr + Step, but it is a lot easier to type. C allows you to use either form. Programmers used to Basic, where If-Then-Else statements are limited to a single line, will appreciate the fact that in C, If and Else clauses can contain as many statements as desired.

It is obviously beyond the scope of this article to acquaint the uninitiated with all of the delights of C programming. Hopefully, it will spur the interest of some readers enough to investigate further. The happy reader who does so will find much more material available than when I began a year and a half ago. Many of the larger bookstores have several C titles available; I've seen some with over a dozen.

Where to Look

The following bibliography is a somewhat random sampling of titles that I have seen recently. The Ker-