Computer Bits

By Hal Chamberlin

HIGH-LEVEL LANGUAGES

DURING the short 30-year history of computers in general and even shorter 4-year history of microcomputers, the most significant software development has been the availability and use of high-level computer languages. The term "high level" is used because these languages allow a programmer to interact with the machine on a less detailed and more meaningful basis than does a so-called "low-level" machine language. The high-level languages permit the average person to use a computer system effectively without having to acquire a large amount of specialized knowledge about binary number systems, memory mapping, character codes, and other such nonsense. Also a given high-level language tends to be the same regardless of the particular type of computer involved. This, of course, is not true with machine or assembly language, which is entirely different when going from one computer type to another.

High-level languages can be broken down into two basic groups. The first group consists of *general-purpose* programming languages. These are intended for use in writing computer programs, both casual and professional. They are called general purpose because they can be used, at least in theory, to write any kind of computer program. Some specialization does exist however. For example, FORTRAN is best suited for complex scientific calculation but has been used for business data processing. COBOL is the most widely used language for business programming but it also has scientific applications although it is very inefficient in that field.

Members of the second group are called "application languages". These are associated with particular "application package" programs. One example is ECAP which stands for Electronic Circuit Analysis Program which is used to simulate and analyze the behavior of electronic circuits. The ECAP language is used to describe the circuit of interest and to instruct the program on what to do with the circuit just described. Another is COGO standing for Civil Engineering Coordinate Geometry, which is used to aid surveyers in evaluating and mapping parcels of land. Using an application program and associated language for its intended purpose is vastly simpler than writing a program to do the same thing from scratch with a general-purpose language.

Of course, all of these advantages of high-level computer languages do not come free. A given program written in a high-level language invariably requires more computer memory and more execution time. The differences generally are not trivial either. On a large machine the difference in memory requirements can easily be 3 to 1 and execution time 5 to 1. On a microprocessor, the memory difference might actually be smaller but the time difference can be 10 to 1 to over 100 to 1.

The difference in programming effort swings to the other extreme with high-level languages requiring as little as one-tenth the effort from inexperienced programmers. Professional programmers cope better with machine-level languages but the difference is still substantial. In effect, machine language gives the programmer complete control over the details of programming thus providing the opportunity to write an *efficient* program, one that takes a minimum of memory and execution time. The situation is analogous to automatic versus manual transmissions in cars. Better gas mileage, quicker acceleration, and better handling in snow is possible with a 4-speed manual transmission but the automatic is more convenient and easier to learn.

Most hobbyists want to run BASIC on their systems and are willing to pay a premium in order to do so. BASIC is a well-known, very easy to learn, general-purpose computer language that works well on small systems. It is particularly effective for small- to medium-sized programs involving mathematics and character string manipulation.

Several other languages are now slowly being implemented on microcomputers. Probably the most widely desired is FORTRAN which is better suited than BASIC for writing large or complex programs. Accordingly, larger systems with more main memory and mass storage devices are required to run FORTRAN. Many application packages such as ECAP mentioned earlier are written in FORTRAN. Actually BASIC was modelled after FORTRAN with many of its difficult or confusing features omitted or modified.

Another language generating much interest among advanced hobbyists is APL. This is a highly symbolic language that is very adept at handling arrays of numbers and other structured data.

Inside High-Level Languages. Actually a high-level language package is nothing more than a program itself, although it is very complex. Simply stated, the "language processor program" looks at statements in the particular high-level language and translates them into equivalent machine-language operations. Such language processor programs are called "compilers" and "interpreters" according to the two distinctly different methods of translation. Incidentally, most language processors are written in machine language to maximize efficiency of the translation process which, as will be shown later, is very important.

When using the compiler type of translator program, it is really a two-step process to take a program written in the high-level language and get it running on the computer. First the compiler program takes the high-level language statements, which as a group are called the *source program*, and translates them, one at a time, into equivalent machine-language instructions. The collection of generated machine-language instructions is sent to a storage device (such as a cassette or floppy disk) and is called the *object program*. Now we have an equivalent program in machine language written on the storage device which completes step one.

Before the object program can be run, it must be loaded into memory. Along with it are loaded some utility subroutines which are called the *run-time package*. These subroutines perform generally needed functions such as binary/decimal conversion, mathematical functions, and others. Usually a special loader program is required to read the object program and run-time package into memory and get them properly linked together. After the loading process is complete, the program may actually be run just as though it had been written in machine language in the first place.

Using the interpreter type of translator is generally much simpler. The main idea is to make the programmer believe that the computer is actually executing the high-level language directly. Accordingly a portion of the interpreter is actually a text editor which aids the programmer in entering the high-level language program into memory and changing it. The source program in this case is stored in memory as ASCII character strings which is simply the original program text or a slightly modified version of it. After the program is typed in, it may be run directly by using the second portion of the interpreter.

In effect the interpreter looks at the first program statement, translates it to machine language, executes it, and then forgets the translated version. Then the second statement is processed in the same manner. If a group of statements constitutes a loop, each is translated, executed, and thrown away in turn, even though these statements may have been translated hundreds of times previously. Actually most interpreters do not generate real machine language and then execute it. Instead, they scan the high-level language statement, extract the important information from it and act directly on the basis of that information. Thus the impression is given of a machine with a very powerful instruction set that actually executes the high-level language directly.

Now what about the relative merits of the two techniques? The interpreter certainly sounds simpler and more convenient to use and indeed it is. But what is gained in ease of use is lost in execution speed. Most programs spend nearly all of their time in one or more short loops. With an interpreter, the statements of the loop are repeatedly scanned and translated. Usually the translation process for a statement takes longer than the actual work specified by the statement. With a compiler, all of the statements of the program are translated once so that, during execution, only the time necessary to perform the useful work is needed. Storage space for the user program is not significantly different between a compiler and an interpreter. However, since the compiler program is not in memory when the object program is executing, it is likely that a larger program could be accommodated with a compiler.

The choice between interpreter and compiler depends heavily on the language. BASIC is nearly always implemented as an interpreter in order to maximize its convenience. FORTRAN, on the other hand, is usually compiled. APL, due to its very structure, is always interpreted. Actually, since APL programs are so compact, the overhead associated with interpretation is much less than with a verbose language such as BASIC.

Hardware Required. Using BASIC as an example, how large must a system be to use it effectively? Usually the only input/output device required is a terminal of some sort. A mass storage device is handy for saving programs but is not necessary to run them. Thus the important measure of system size is simply the amount of memory present. Accordingly, BASIC interpreters are usually rated by the amount of memory required by the interpreter and the number of language "features" supported. Memory requirement figures often include a small area for storage of the user program but this generally amounts to only a few lines of BASIC.

The smallest BASIC interpreters can be as small as 2k bytes. Being so small, they offer only the most important language features and are often given the name *Tiny* BASIC. The most distinguishing feature of a Tiny BASIC system is that only integer numbers are allowed, usually constrained to values between −32,768 and +32,767. So-called "full-featured" BASIC interpreters can run in as little as 4k. These support normal floating point (scientific notation) arithmetic and common mathematical functions but lack the features for character string manipulation. The most common interpreter size is 8k. All useful mathematical functions are allowed, two dimensional arrays are permitted, and statements are provided for handling character strings. The ultimate is variously called Disk BASIC, 12K BASIC, Extended BASIC, and other similar names. In addition to all of the features just mentioned, these interpreters allow user programs to set up and access files of data on a floppy disk or cassette tape. If any kind of home accounting or small business applications are expected to be written in BASIC, then this is the version that is needed.

As noted above, additional memory is required for all but the most trivial programs. Although it is difficult to judge how much memory a given program will require without actually trying it, a good rule of thumb is to allow 1k per 50-line page of sparsely commented BASIC statements. If long statements with a lot of comments are more your style, 2k per page might be a better figure. Also, many interpreters allow the mathematical functions to be deleted if not needed thus freeing 500 to 1000 more bytes of memory.

Speed is even harder to pin down than memory requirements; but compared to machine language, BASIC is quite slow. Simple arithmetic operations such as addition and subtraction generally take about 5 milliseconds each while multiply and divide take a little longer. Of course the floating point arithmetic, which itself is a complicated subroutine in the interpreter, is partially responsible but the tests have shown that even integer-only Tiny BASIC's are not much faster. Thus the conclusion is that the interpretation process takes a lot of time. In many cases of course, speed is of no consequence, but if a lot of calculation is to be done the time can add up quickly. An excellent article in the June, 1977 issue of *Kilobaud* magazine compares the speed capabilities of several BASIC interpreters.

Increased use of high-level languages is definitely a wave of the future which will be spurred on by the development of microprocessors specifically designed to support such languages. ◇

DELUXE
MAGAZINE CASES
Designed Exclusively for
POPULAR ELECTRONICS

● The ideal way to save your valuable copies, keep them well protected and make it easy for you to refer to any issue at any time. Both decorative and attractive enough to enhance the decor of any room, each case holds a full year's copies. Constructed of reinforced fiberboard, these durable cases are covered in a rich-textured leatherette. The gold embossed back adds to its elegance and makes each case a welcome addition to your bookshelf or cabinet.
Cases are available for your favorite magazines.

Only $5.95 each, 3 for $15.50 in any combination of titles, including all postage and handling charges. Outside U.S.A. add $1 per case.

CHARGE YOUR ORDER TO YOUR AMERICAN EXPRESS, BANKAMERICARD, MASTER CHARGE OR DINERS CLUB ACCOUNT.

Ziff-Davis Serv. Div., Dept. JJ, 595 B'way, N.Y. 10012
Please send the Magazine Cases indicated below:

PE-65 TITLE	QUANTITY
______	______
______	______

CHECK: ☐ All Black ☐ Maroon Back, Black Sides
☐ ENCLOSED IS $______. PE-117
☐ CHARGE: ☐ BankAmericard ☐ Master Charge ☐ American Express ☐ Diners Club
Account #______ Exp. Date______
Master Charge Interbank #______
(4 numbers over your name)
Signature______
Print Name______
Address______
City______ State______ Zip______
Residents of Calif., Col., Fla., Ill., Mich., Mo., N.Y. State, D.C. and Tex. add applicable sales tax.