ED 060 620 EM 009 626

AUTHOR	Bork, Alfred M.
TITLE	Introduction to Computer Programming Languages.
INSTITUTION	California Univ., Irvine. Physics Computer Development Project.
SPONS AGENCY	National Science Foundation, Washington, D.C.
PUB DATE	Dec 71
NOTE	5p.
JOURNAL CIT	JCST; P 12-16 December 1971
EDRS PRICE	MF-$0.65 HC-$3.29
DESCRIPTORS	Computer Assisted Instruction; Computer Science Education; *Guides; *Programing; *Programing Languages

ABSTRACT

A brief introduction to computer programing explains the basic grammar of computer language as well as fundamental computer techniques. What constitutes a computer program is made clear, then three simple kinds of statements basic to the computational computer are defined: assignment statements, input-output statements, and branching statements. A short description of several available computer languages is given along with an explanation of how the newcomer would make use of basic computer software. Finally, five different versions of a simple program (for solving the harmonic oscillator numerically) are given with comparison. (RB)

Introduction To Computer Programming Languages

By Alfred M. Bork

The digital computer is a powerful, calculating, and logical device. In order to take advantage of its capabilities, the user must know how to instruct the computer in a language that it understands. Different computers have different languages available to them, and even for the same computer, the language situation is likely to change.

A Program Before reviewing the types of statements used in computer programming, it might be useful to consider the overall question of what constitutes a computer program, i.e., the instruction given to a computer. The computer program can be viewed as something like a paragraph, composed of a number of "sentences." These individual sentences are executed in the order in which they appear in the paragraph, unless a "branching" sentence explicitly changes this order and directs the machine to go to another sentence. In some languages every sentence is numbered, and these numbers determine the order of execution except for branching statements.

Usually the computer offers some easy way of changing individual sentences, so that the program can be modified and corrected for future use. It is very important to keep in mind the sequential nature of the calculation—the fact that the sentences are executed in order. For example, if a new value of some variable is calculated in one sentence, that value will be used in later sentences which refer to the same variable.

The flow of a computer program can be studied by means of a chart or diagram, sometimes called a flowchart, and books on computer languages will show many flowcharts. Although there are some conventions for flowcharts, the beginner does not really need to worry about them. It does sometimes help to think about a program in a chart form, however, because it gives an overall "picture" of what is happening in an easy-to-comprehend, graphic form.

All computer languages intended primarily for calculation purposes have similar features and structures. Preparing a beginning program in any of these languages (e.g., the type of program that might be needed in physics) is a similar process. Simple calculational computer programming involves a limited number of activities, or kinds of statements, directing the computer to do what the programmer wants it to do. For convenience, we can distinguish three of these—assignment, input-output, and branching. This does not, by any means, exhaust everything that a computer can do, but it is enough to get started.

Assignment Statements First, since the computer is a calculational tool, it is not surprising that some statements in the computer language are designed to tell the computer just what calculations the programmer wants it to perform. A standard shorthand language describing calculations algebraically already exists. A modification of this language, in one form or another, is used in most contemporary computer languages. The basis for describing a calculation in algebraic terms is a formula, such as

$$F = MA$$

which says that to determine the number F, one must multiply the numbers M and A together. Similarly,

$$X = A \sin (wt)$$

specifies a more elaborate procedure for finding X from A, w, and t.

A formula with only a single unknown on the left hand side, and a group of items describing how to calculate that unknown on the right hand side, is characteristic of a typical computer assignment statement. Here are several examples, with indications of the language.

```
20  V=VO+A*T       (BASIC)
    P=N*R*T/V      (FORTRAN or PL/1)
[7] A←B*(C+D)      (APL)
```

An assignment statement is somewhat different from a formula, and some languages (e.g., APL) replace the equal sign with some other sign, such as a left-pointing arrow. A formula does not, by itself, indicate that a calculation must be carried out but merely states the relation among the variables. Furthermore, a formula can have many variables on both sides of the equation. However, the formulas which lead to assignment statements in a computer language are all ones with a single variable on the left; they specify a collection of arithmetic operations as indicated on the right to be carried out and assign that

Dr. Bork is professor of physics and of information and computer science at The University of California, Irvine.

value to whatever variable is indicated on the left. Most languages allow free use of spaces in order to make formulas more readable.

The computer treats the variables as storage slots in its memory. Corresponding to each variable in a particular program, an appropriate storage location exists in the computer memory. The memory can be thought of as a collection of mail boxes, with the labels that identify some of the mail boxes being the same labels which identify the variables of the problem. Thus, in the relation

```
7.42      F = M*A      (JOSS)
```

in a computer program, the computer is being told to find the numbers currently stored in the two locations designated M and A, to multiply these, and to store the results in the location called F. Whatever number was previously stored in the location F will be changed in this operation: but the numbers stored in locations M and A will not be. (The process of how the computer associates variables with locations in memory is a bit more complicated than indicated here, but this description illustrates what is happening.)

The variable on the left can also occur on the right. Thus the statement in an APL program

```
COUNT←COUNT + 1
```

increases the value of the count by one.

The assignment statement, with slight variants in different languages, is one of three building blocks in computer programs. Most computer languages allow the standard arithmetic operations of addition, subtraction, multiplication, division, and exponentiation. Most use an asterisk for multiplication, although APL uses the usual multiplication sign. All computer languages in common use require an *explicit* multiplication sign; it cannot be omitted as in algebra. Thus:

```
A←B(C+D)      (APL)
```

would be an erroneous statement, because the multiplication sign following the B has been omitted.

Computational computer languages have a symbol for exponentiation, that is, raising a number to a power. In FORTRAN and PL/1 the symbol for "to the power" is a pair of asterisks in a row; in BASIC it is an up arrow; in APL it is a single asterisk. Some languages, particularly APL, contain other arithmetic operations in addition to these basic ones.

Parentheses can be used freely in assignment statements. In fact, this is highly desirable, because the language might not make the same assumption that the programmer does about the order for carrying out the various arithmetic operations. When in doubt, use parentheses.

The names that can be used for a variable also differ. Except for BASIC, computer languages usually allow, as already suggested, multiple letter or letter plus number names for variables. In FORTRAN, PL/1, and APL, the name TIME could identify a single variable. Given this free choice of variable name, it is easy to see why explicit multiplication is required. Many forms of BASIC restrict variables to single letters, or single letters followed by numbers. The variables, then, can have convenient names which remind the programmer of the intended meaning or use.

All the computational processes in a program can be described by a series of assignment statements. These statements are executed one by one, in sequence. A computer program is executed in the order of the statements that are given to the computer: although, as we will later see, exceptions to this do exist. As each statement is executed, it uses the values currently stored in the memory slots. Thus, if a PL/1 program contains these two statements,

```
DIST=ACC*TIME*TIME
Q=TIME*DIST/(DIST**3.7),
```

one following the other, the value of DIST calculated in the first statement will be used in the second statement. The same value of TIME will be used in both calculations. A corollary is that a variable must be defined, or calculated, in a program *before* it is used in another calculation. If it is used without previous definition, the results may be unpredictable. Some languages will alert the programmer when this is happening, but not all have this capability.

Input-Output Statements The second type of fundamental statement in a computer program is the input/output statement, used when it becomes necessary to enter information into the computer while the program is running or when it is necessary to run the program with different values. The programmer will almost certainly want to know the results of the calculations which the computer has been performing. The assignment statements only store the results of the calculation internally: they do not give access to those results. The programmer must tell the computer explicitly when he wants input and output. All computer languages have either simple input statements that allow the programmer to enter values and store them in some of the memory slots preliminary to doing a calculation, or they have statements which allow the programmer to read out the values contained in some of these slots in a form convenient for understanding what is happening.

Details differ. Consult the sample programs. The input statements often have words like INPUT, ACCEPT, or READ to identify them, while output statements have words like WRITE (FORTRAN), PRINT (BASIC), TYPE (JOSS), or PUT (PL/1). APL has a somewhat different procedure—variables listed by themselves will automatically be printed out. Also, the format in which the output is presented, the number of significant figures, the number of numbers on a line, and the spacing between numbers may or may not be under control in the calculation in the language that you use. Sometimes the messy details may have to be mastered in order to learn how to get the kind of format desired. This is a common source of annoyance to beginners with some languages (particularly FORTRAN), but these difficulties are only minor in most cases. It is not essential to learn everything about a language at first try: enough information to get started in working the problems is an adequate beginning.

Branching Statements The third kind of statement in a computer program, in any language, gives real power to computer programming and sometimes allows the description of a very long calculation by a short program. This is the branching statement. We have said that the statements in a program are usually executed one after the other, in the order that they are typed or entered on punched cards, or, as in BASIC, JOSS, and APL, in the order of the statement numbers. But the programmer can change this order by means of a branching statement.

A branching statement is an order to the computer to go to, and execute, some different statement in a program, *not* the next statement, and to continue executing from this point. There are two basic types of branching statements, unconditional and conditional. The unconditional branch *always* takes place; whenever the computer gets to that point in the program, it always hops to some other place and starts executing statements in order from that point.

→4	(APL—go to statement labeled 4)
→COMPUTE	(APL—go to statement labeled COMPUTE)
GO TO 44	(FORTRAN—go to statement labeled 44)
GO TO NEXT	(PL/1—go to statement labeled NEXT)
47 GO TO 22	(BASIC—go to statement labeled 22)
TO 4.3	(JOSS—go to statement labeled 4.3)

The second branching statement is a conditional branching statement. Whether the programmer goes to another place or not depends on the results of the calculations that have occurred so far. Thus, using conditional branching, the calculation can be controlled, sending the computer back to do things over again under some circumstances, but not under others.

44 IF T<10 THEN 150	(BASIC—go to 150 if T<10; otherwise, execute the next statement)
IF (DELTA .LE.6.3) GO TO 66	(FORTRAN—go to 66 if DELTA<6.3; otherwise continue)
3xI(TIME<5)	(APL—to 3 if TIME<5; otherwise continue)

The branching statement, whether conditional or unconditional, has to indicate where to go within the program. This means that there must be some way of identifying the line to which the computer goes, i.e., a label for that line. Languages differ as to the kinds of things that are acceptable as a label: some languages allow only numbers while others allow the programmer to choose convenient names. In some languages (e.g., BASIC, JOSS), every line must have a label, but in other languages only the lines that are going to be jumped to, need to have such a label (e.g., FORTRAN, PL/1). Other details about the branching statements, particularly the conditional branching statements, also differ widely from language to language. But most of them have the same general outline of allowing the programmer to test values in the calculation at that point, and make a decision to jump or not to jump based on that test.

For the Beginner We would not want to claim (because it is not true!) that these three types of statements are all that occur in programming languages. A rich language may have other types too, and this classification may not even be appropriate. But these basic ideas are enough to get started and are all that the novice need be concerned with initially.

As we have said, it is not necessary to learn all about a language at first. Rather, it makes sense to learn only what you need now, assuming that as your needs expand, you will be able to learn more about the language. For some languages it would be an almost Herculean task to learn everything, but it is a simple task of only a few hours to learn a useable subset of any common language for a physical science course.

Some general advice about learning a programming language may be useful. Details differ, depending on what type of system is being used. First, the user should maximize the benefits of working in a terminal environment, if this is the case. This means that he should go relatively quickly to the computer terminal and start running small examples of programs. One learns much more quickly by practice than by reading, and terminal facilities are ideal for quick practice. A computer-knowledgeable friend would be a definite advantage in initial sessions at the terminal.

A number of standard books are available for each language, but the user should distinguish between books designed as basic instruction in the language, such as primers, and manuals which describe all the facilities of the language. The latter, usually available from the computer manufacturer, are useful in active programming; however, as they attempt to be encyclopedic, writing of the language in a logical, rather than a pedagogical fashion, they are not the best aid for the beginner.

Available Languages Following is a brief description of common computer languages for numerical computations. FORTRAN is presently, by far, the most common calculation language, and most scientific computation is done using FORTRAN. Designed in 1957 by IBM for the 704, it is available on almost every computer. Like the other languages discussed here, it is an algebraic language describing calculations using formula-like expressions. FORTRAN has many different "dialects," the most widely-used being FORTRAN IV. which usually includes logical "IF" statements and complex variables. As it was one of the first such languages, it is in some ways cruder than newer languages. FORTRAN is used in both timesharing and batch.

ALGOL is a second generation algebraic language, after FORTRAN, and has a more logical structure. It is more widely used in Europe than in the United States: however, many American machines have ALGOL compilers. A new version of ALGOL, ALGOL 68, is coming into use. Most ALGOLs are available only in batch.

JOSS is an algebraic language developed by the Rand Corporation particularly for timesharing. It can be used either for programs or as an electronic desk calculator. Because it is intended for terminals, it provides rapid feedback for grammatical errors. Variations of JOSS exist under many names.

BASIC originated at Dartmouth College and is a widely available algebraic terminal language similar to JOSS. Various forms of BASIC differ in the way they handle alphanumeric strings. BASIC has a convenient set of matrix operations.

PL/1, planned by IBM and users for System 360, combines algebraic, business, and list-processing facilities, and allows complex programs which use both linguistic and computational modes. It is a "third generation" algebraic language, following FORTRAN and ALGOL. Like FORTRAN, it is available in both batch and terminal modes. Although it is used primarily on IBM machines, it is also available from several other manufacturers.

APL (A Programmed Language) is a highly interactive, recently-developed, time-shared language. In addition to arithmetical operations, it has many powerful numerical and string operators. While beginners can write APL programs which look much like BASIC or PL/1, the strength of the language comes from its ability to conveniently manipulate arrays and matrices. It has gained rapidly in popularity.

Operating Systems Using computers has another aspect besides the languages in which we construct the programs. Some mechanism for getting the information—both program and date—in and out of this system must exist, and this will involve both the mechanics of how this is done (the typewriter-like terminals, keypunch machines, paper tape punches, or other similar devices) and the instructions to the computer to tell it just what you want it to do. A modern computer can handle a great variety of languages and has considerable room for choice as to how the information is to be put out and where. All this information is conveyed to the computer by the user, although a "default" may exist if nothing is said.

Most computers run under the control of an operating system, which serves as an overall supervisor to manage the functioning of the computer to make it as efficient as possible. Operating systems differ widely. Most "batch" systems will require a series of job "cards" (which may not actually be punched cards) preceding the program, as well as directions for telling it what to do. Operating systems usually have commands in addition to the language commands which specify the details of the task at hand. The specification could be simple or complex, depending on the degree of choice offered to the user in a particular system. Just as with the language, there is no need to worry about learning all the details at once. You can start with some standard ways of preparing operating systems instructions, probably available at your computer installation.

Interpretation In Table 1 we present five different versions of a simple program for solving the harmonic oscillator numerically. In the present section we will analyze several of these programs in a line by line fashion, explaining what is happening, and developing, in addition, some of the features of the languages.

JOSS Note that each statement has a number, and that the numbers are all decimal numbers. The statements are executed in numerical order, except when branching statements appear. The integer part of all the numbers is one, so the statements together constitute, from the JOSS viewpoint, part one. To execute these statements in JOSS, one would type, DO PART ONE. (With most of our languages, the implementations differ, and JOSS is no exception.) Statements 1.1 and 1.2 and 1.3 set the initial conditions, and 1.4 determines the time step. These statements "initiate" the variables. Line 5 is the calculation of the new position and 1.6, the calculation of the new velocity. In each of these equations the same variable appears on both right and left hand sides of the calculation. On the left we use the old value, but we change the value in each case when we execute the statement. Line 1.7 computes the new time by adding the time step to it. Then we type out the results, and, in step 1.9, we go back to the step labeled 1.5 if T is less than 3. This last step sets up a *loop*, the principle calculation loop of the program.

BASIC In BASIC the line numbers are all integers, and the statements are carried out in the order of the numbers. Some forms, but not all, require the use of LET before assignment statements. Again, the first three lines set the initial conditions for time, position, and velocity, and the fourth line sets the time step. The next two lines are calculations of position and velocity, and the new time is calculated in line 170. Line 180 prints out the values just calculated, and 190 returns to statement 150 if T currently is less than 3. Some BASICS would not require an END statement.

FORTRAN In FORTRAN not every statement needs to be numbered: only those statements referred to by others need to have a statement number. The first three statements set the initial conditions, and the fourth one sets the time step D. (In FORTRAN one could use something like DT, but D is used here to follow the examples in the more restrictive languages.) Statement 10 and the next line are the calculations of the new position and velocity, and again the new time is computed. The WRITE statement in FORTRAN is a bit more complicated than similar statements in the other language. Six indicates the unit on which one is to write. perhaps the line printer or the user's terminal. Seventy references the FORMAT statement,

TABLE 1.
The Harmonic Oscillator

JCSS.

```
1.1  T = 0
1.2  X = 1
1.3  V = 0
1.4  D = .1
1.5  X = X + V*D
1.6  V = V − X*D
1.7  T = T + D
1.8  TYPE T, X
1.9  IF T<3, TO STEP 1.5
```

BASIC

```
110 LET T=0
120 LET X=1
130 LET V=0
140 LET D=.1
150 LET X=X+V*D
160 LET V=V−X*D
170 LET T=T+D
180 PRINT T;X
190 IF T<3 THEN 150
200 END
```

FORTRAN

```
    T=0.
    X=1.
    V=0.
    D=.1
 10 X=X+V*D
    V=V−X*D
    T=T+D
    WRITE (6, 70) T,X
    IF (T−3.) 10. 10, 14
 14 STOP
 70 FORMAT (F10.2, F12.4)
    END
```

APL

```
  ∇HARMONIC
[1] T←0
[2] X←1
[3] V←0
[4] D←0.1
[5] CALCULATE:X←X+V×D
[6] V←V−X×D
[7] T←T+D
[8] T, X
[9] →CALCULATE×T<3∇
```

PL/1

```
OSCILLATOR: PROCEDURE OPTIONS (MAIN);
        T = 0; X = 1; V =0; D = .1;
CALCULATE: X = X + V*D; V = V − X*D;
        T = T + D; PUT SKIP DATA (T,X);
        IF T<3 THEN GO TO CALCULATE;
END OSCILLATOR;
```

the next to the last statement in the program. It describes how one wants the information to come out. Thus, for the time t. ten places are to be allowed, with two places to the left of the decimal point, while for x, twelve places are allocated with four places to the left of the point. It is a bother to write FORMAT statements in FORTRAN, but it does give more flexibility. Many types of FORTRAN have simple input/output statements without format, but they are not used here because they are non-standard. The branching statement is also a little more complicated in FORTRAN than in other languages. It says if T - 3 is negative. go to statement 10, and compute some more. If it is 0, go to statement 10 also. While if it is positive—that is, T is greater than 3—then go to statement 14, which stops the calculation.

APL In APL the program is a function, here called HARMONIC. The symbol ∇ is used to show the beginning and end of the function. The lines are numbered; this numbering is provided by the APL system. rather than by the user. Note that in place of the equal signs in the other programs, we have left-pointing arrows. The first lines set initial values for T, X, and V, and for the time step D. Lines 5 and 6 compute new positions of velocity. Five has a label. CALCULATE, in front of it; the colon indicates that it is a label. One could, in APL, have referenced this statement by the line number, five, but sometimes the label is more convenient. Note that the multiplication sign is an actual multiplication sign. not an asterisk. Line 7 computes the new time, and line 8 is the output statement. Simply listing variables on a line with nothing else indicates to APL that the current values of the variables are to be displayed to the user. Nine is the branching statement, telling the computer to go to CALCULATE, the labeled statement (5). if T is less than 3, but otherwise to terminate the program. Branching statements in APL are powerful, but the beginner should probably use only a few routine ones, rather than try to learn all the tricks of the trade.

PL/1 Examining the PL/1 program, the second line establishes the initial values of the variables used. The calculations of position and velocity are in the third line. These look very much like the corresponding algebraic relations with one exception. No primes are used. The same variable x occurs on both sides of the equation. In algebraic computer languages, such as PL/1, an equation is a specification for a calculation. It tells the computer to take the variables on the right hand side from computer memory, perform the indicated computation. and put the result back in the position in memory that holds the variable X. Each variable is associated with a "slot" in memory, so the net effect of this operation is to change the value of X stored.

As already suggested, the differences among these various programs are minor. For simple calculational purposes, such as the study of the harmonic oscillator, the differences of the common computational computer languages are small. The major areas of difference are with input/output and branching statements. However, if one goes beyond this beginning level, the languages tend to diverge. ▲