[illegible]

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Technical Memorandum 33-566

A Brief Description and Comparison of Programming Languages FORTRAN, ALGOL, COBOL, PL/I, and LISP 1.5 From a Critical Standpoint

F. P. Mathur

JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

September 15, 1972

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No. 33-566	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle A BRIEF DESCRIPTION AND COMPARISON OF PROGRAMMING LANGUAGES FORTRAN, ALGOL, COBOL, PL/I, AND LISP 1.5 FROM A CRITICAL STANDPOINT		5. Report Date September 15, 1972
		6. Performing Organization Code
7. Author(s) F. P. Mathur		8. Performing Organization Report No.
9. Performing Organization Name and Address JET PROPULSION LABORATORY California Institute of Technology 4800 Oak Grove Drive Pasadena, California 91103		10. Work Unit No.
		11. Contract or Grant No. NAS 7-100
		13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address NATIONAL AERONAUTICS AND SPACE ADMINISTRATION Washington, D.C. 20546		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

Several common higher level program languages are described. FORTRAN, ALGOL, COBOL, PL/I, and LISP 1.5 are summarized and compared. FORTRAN is the most widely used scientific programming language. ALGOL is a more powerful language for scientific programming. COBOL is used for most commercial programming applications. LISP 1.5 is primarily a list-processing language. PL/I attempts to combine the desirable features of FORTRAN, ALGOL, and COBOL into a single language.

17. Key Words (Selected by Author(s)) Computer Applications and Equipment Computer Programs	18. Distribution Statement Unclassified -- Unlimited

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 13	22. Price

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Technical Memorandum 33-566

A Brief Description and Comparison of Programming Languages FORTRAN, ALGOL, COBOL, PL/I, and LISP 1.5 From a Critical Standpoint

F. P. Mathur

JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

September 15, 1972

Prepared Under Contract No. NAS-7-100
National Aeronautics and Space Administration

Preface

The work described in this report was performed by the Astrionics Division of the Jet Propulsion Laboratory.

Contents

Abstract

Several common higher level program languages are described. FORTRAN, ALGOL, COBOL, PL/I, and LISP 1.5 are summarized and compared. FORTRAN is the most widely used scientific programming language. ALGOL is a more powerful language for scientific programming. COBOL is used for most commercial programming applications. LISP 1.5 is primarily a list-processing language. PL/I attempts to combine the desirable features of FORTRAN, ALGOL, and COBOL into a single language.

A Brief Description and Comparison of Programming Languages FORTRAN, ALGOL, COBOL, PL/I, and LISP 1.5 From a Critical Standpoint

I. Introduction

Explicit procedures, namely, algorithms or programs for the solution of mathematically structurable problems, require a notation and syntactic structure for their description. Such systems of notation are called programming languages.

The solution of a problem by means of computer facility consists of explicitly delineating the steps or commands that a computer should follow in order to produce the solution. The computer must be instructed in the manner in which it must expect to receive the data to be acted upon. This data and the format of the solutions must be presented to the computer in a prescribed manner. Though the computer is incapable of formulating these procedures, its forte lies in following the commands with great speed.

The algorithm, which may be in the form of a flow chart, and the data are coded in the source language (the subject of this paper) and inputed to a compiler where it is compiled, assembled, and effectively translated into an object program that is in machine language — the binary language in which a computer basically operates. The process of compiling an object program from a source program is shown in Fig. 1.

In the early days of computers, all programming was done in machine language. From the standpoint of the computer, the machine language programming is the most efficient; however, it requires that the programmer be familiar with the internal organization and design of the computer. As computer applications became widespread, higher level languages were developed that separated the user from the designer and became more user oriented. At first these languages were oriented toward the scientist programmer and were strictly for scientific applications. As computer applications broadened toward data processing and business-type applications, business-oriented languages came into being. As a next step these two main, though basically divergent, orientations were and are being incorporated within the same language. The success of truly general-purpose languages as measured by widespread acceptance is yet to be seen.

II. Basic Philosophy of Programming Languages

A significant difference between natural languages and programming languages is, that due to the nonintelligent nature of the computer, no ambiguities in the programming language may exist. The language must be context free; hence, it is not subject to growth and spontaneous development.

The ideal problem-oriented source language designed for wide use must possess the characteristics of being easily learned, easy to use, easy for others to follow, should not require a huge compiler to produce a reasonably efficient execution of the job, must be computer independent (i.e., should be usable on other machines) and the structure must lend itself to automatic error detection. Many of the above requirements are conflicting, as is apparent in the following excerpt:

"An ideal source language should be easy to learn and simple to use, such a language must be general: a source language must apply to the widest possible range of problems to be of greatest value, must be concise with as few words as possible and using identifiers as far ranging as possible. It must do a maximum amount of work with a minimum of commands. Through this it will tend towards symbolic languages but not in such a way as to make its resulting simplified programs difficult for the layman to follow. It should obey the cardinal programming rule of absolute precision but at the same time should avoid any unnecessary complications which will make the construction of the compiler more difficult. Finally, it should be as natural and unforced a language as possible" (Ref. 1).

In subsequent pages an attempt will be made to see how close the higher level languages under consideration approach the ideal source language defined above. For a thorough treatment of language comparison, the reader is referred to Ref. 2, where some 120 languages are broadly covered and illustrative sample programs are given for some 30 languages.

III. Metalanguages

Metalanguages are languages whereby other languages may be described. One of the characteristics of natural languages (since they are, in general, unbounded) is that they are their own metalanguage. ALGOL, a widely used metalanguage to describe programming languages, is termed the Backus-Naur form of syntax description. It describes precisely what combinations of symbols are meaningful and its use results in metalinguistic formulas.

This metalanguage employs four characters which are unrelated to, and distinct from, the programming language under description. These are the metalinguistic brackets '<' and '>' which are used to enclose names of things about which the metalanguage is talking, '::=' which means "is," and '|' which means "or," Thus

<digit> ::= 0|1|2|3|4|5|6|7|8|9

is read "a digit is a 0 or a 1 or a 2 or a 3 or a 4 or a 5 or a 6 or a 7 or a 8 or a 9."

Sometimes what is being defined is mentioned on both sides of the symbol '::=,' e.g., <identifier> ::= <letter>| <identifier> <letter>|<identifier> <digit> which says that "an identifier is a letter or an identifier followed by a letter or an identifier followed by a digit." This is a recursive definition, which is equivalent to saying that an identifier is any sequence of letters and digits which starts with a letter.

The above example of metalinguistic formula is an illustration from ALGOL. In describing other languages, such as COBOL, other metalinguistic techniques have been developed. Charts also exist which describe in graphical form the syntax of the language.

IV. FORTRAN (Formula Translation)

FORTRAN is the most widely used high-level programming language. The original version of FORTRAN, FORTRAN I, was developed in 1957 by the IBM Corporation for the 704 computer, which now is obsolete. Originally all programming was done using hand-coded symbolic machine instructions. FORTRAN I programs were monolithic, and minor changes in the program required that the entire program be recompiled. Expansion of FORTRAN I into FORTRAN II in 1958 allowed programs to be separately compiled. Subprogram facility allowed data to be declared common to all programs; also, subprograms could invoke one another, thus enabling the task of programming to be divided among several programmers. FORTRAN II also had the feature that parts of the program could be written in symbolic machine code in order to increase efficiency of compilation. Input/output was also expanded for handling and editing alphanumeric information.

In 1961, FORTRAN II was augmented to FORTRAN IV (hereafter referred to as F II and F IV). F IV contains all the features of F II plus a number of additional ones. The additional features are double precision, complex number arithmetic, logical variables, logical connectives, and relational expressions. Also, probably due to the influence of ALGOL, a number of formal syntactic changes were made. A new conditional logical IF statement supplementing the arithmetic IF statement was introduced, thus

IF (Z .GT. 5 AND. Z .LT. 9) Y=Z

In F II only a single COMMON block can be defined, whereas in F IV several independant COMMON blocks may be defined.

F II had only two types of data, floating-point and fixed-point, and had very simple means of distinguishing them. All variables beginning with letters I through N were denoted as fixed-point and the remaining as floating point. Even with the introduction of three new types of data in F IV, the simplicity and convenience of this system has been retained. However, F IV may overrule this implicit declaration by explicitly declaring a variable as any one of the five types: real, integer, logical, complex, or double precision. Variables in FORTRAN may be alphanumeric, up to six characters long, and the first character must be a letter.

FORTRAN hardware representation is restricted to the 48-character set, only upper-case letters being allowed. Expressions in FORTRAN do not permit mixed modes, i.e., all variables must be of the same type or mode; however, in exponentiating, floating-point numbers may be raised to a fixed-point power. Care must be taken in assigning modes on either side of statements, since during evaluation the expression is completely computed in its basic mode, and the computed value is assigned to the variable on the right of the equal sign, converting its mode, if necessary, to that of the variable being equated to.

In evaluating arithmetic statements the hierarchy of operations has to be obeyed. The basic hierarchy of operation is from left to right. Exponentiation and parentheses operation come first, followed by multiplication and division as second, followed by addition and subtraction as last. It should not be overlooked that exponents in themselves may be expressions. Redundant sets of parentheses may be used, thus allowing complete mathematical freedom; however, the statements become correspondingly harder to read. Programs written in FORTRAN can become quite complex and are much harder to read than those written in either ALGOL, COBOL, or PL/I.

FORTRAN does not have any reserved words; e.g., the following is a DO statement: DO 7 A = 3,5. However, DO7A = 3.5 is an arithmetic statement. FORTRAN is written in fixed form, i.e., only one statement per line is allowed; thus, no punctuation is required to terminate a statement. However, a statement may be continued onto a following line by punching a special character in column six. The order of obeying statements is sequential. Comments may be inserted freely into the program by punching a "C" in column one. Blanks have no significance in FORTRAN and are ignored; thus, GO TO 3 is identical to GOTO3. The exception to the significance of blanks is when input/output is specified, using Hollerith format. Labels in FORTRAN may be numeric only and of no greater than five digits.

Subroutines and functions in FORTRAN may not invoke themselves, i.e., they may not be invoked recursively. (However, the same effect may be obtained by the use of dummy statements.) Subscripts in FORTRAN are written in parentheses, e.g., X_i is written as $X(I)$. An array may have maximum dimension of seven in F IV and three in F II. The maximum size of an array used must be initially declared. Subscripted variables have to begin with unity and are required to be stepped positively only.

A few simple control statements such as DO, IF, GO TO, specify the flow of control. FORTRAN conditional statements, though ingenious and simple, are not as sophisticated as those in ALGOL, (these are described in subsequent pages). The statement, IF (A) 5,10,15 provides three-way conditional branching; if A is negative, zero, or positive, branching takes place to statements labeled 5,10, and 15, respectively.

FORTRAN input/output statements come in pairs; namely, an input or output statement followed by the FORMAT specification, e.g.,

```
PRINT 50,K,P

50 FORMAT (10X,I5,F9.5)
```

states that K and P should be printed according to the FORMAT specification labeled 50. The format specification stated by 10X states that the first ten character positions be left blank, I5 states that the next field of five characters represents a decimal integer; and F9.5 states that the next field consists of 9 characters representing a real number having five digits after the decimal point. The Hollerith H format specification is available for transmitting characters, but suffers from the fact that the total number of characters and blanks need to be counted and specified. Other format specifications are available, and though generally adequate for scientific applications and easy to apply, they are not as extensive as the input/output facilities provided in PL/I. The input/output statements of FORTRAN cannot be compared against those of ALGOL, since none are specified in the latter.)

FORTRAN, as exemplified by its input/output statements is, in general, a compromise between simplicity of use and flexibility of format. It has proved to be such an effective compromise between generality and efficiency that it has become the most widely accepted higher language in use today.

V. ALGOL (Algorithmic Language)

ALGOL was developed by an international computer group and attempts to achieve a common programming language to serve both program input to computer and as a vehicle for communicating algorithms in published form. ALGOL 60 was defined and reported by an ALGOL committee in 1960, and further revisions were incorporated in 1962 in the "Revised Report on the Algorithmic Language ALGOL 60," which was published simultaneously in various international computer journals.

The publication language of ALGOL is used to write algorithms for man-to-man communication and permits use of various notational characters such as Greek letters, etc., which are not included in the reference language and would be unreasonable to expect in the hardware representation. The publication language has been quite successful in achieving its goal, as can be evidenced by the number of algorithms published in ALGOL in such journals as the *Communications of the ACM*.

ALGOL was designed to be machine independent. This characteristic is carried to the extent that no explicit reference to addresses of stored data nor instructions for storing is made. Input-output procedures are left completely undefined, as is hardware representation. Thus the programmer must learn the specific input-output procedures depending on the compiler he may be using.

ALGOL, like FORTRAN, is directed toward numerical computation. ALGOL possesses an explicit syntax, which is defined using the Backus-Naur notation, and together with its block and nested block structure in which each nested block may have its own local identifiers, provides the language with a high degree of elegence of expression. Though all identifiers and variables have to be declared in ALGOL, it does not compare with the sophistication of the Data division in COBOL.

ALGOL comprises 116 basic symbols, namely the ten digits, 52 letters, upper and lower case, and 52 delimiters, such as the six arithmetic operators (e.g., $+, -, /$ etc.), the six relational operators ($\leq, \neq, =$, etc.), the five logical operators ($\equiv, \wedge$, etc.), the six sequential operators (*go to*, *if*, *else*, etc.), the six brackets (' ', [ ], *begin end*, etc.), the seven declarators (*boolean*, *integer*, *real*, etc.), and the three specifications (*string*, *value*, *label*).

Numbers, variables, statements, labels, blocks, procedures, and programs are built up by using the above symbols and obeying the syntactical rules. It should be noted that in ALGOL basic symbols (e.g., *go to*) can be a word or two words and need not be a single character. As contrasted against FORTRAN, two symbols for division are defined in ALGOL, the normal "/" and the operator "$\div$" which is restricted to operands of type integer and yields the integral part of the quotient. The rules of precedence of arithmetic symbols is same as in FORTRAN, and the same left-to-right sequence is retained.

ALGOL programs are written in free form. More than one statement can be written per line and a statement is allowed to overflow to the next line. However, in FORTRAN only one statement per line is permitted and in order to continue on to the next line, a special location on the line (column 6) must be punched. However, ALGOL statements require that they be delimited by a semicolon.

The ALGOL assignment statement corresponds to the FORTRAN arithmetic statement. However, the ambiguity of the "=" symbol is overcome by the use of the assignment symbol ":="; the "=" symbol is reserved for use as a relational operator, where a relational operator can be defined as having arithmetic expressions as input and Boolean values as output. ALGOL also has the feature that in a single assignment statement more than one variable can be set equal to the same value, e.g.,

```
x:=y:=z:=0;
```

The complexity in FORTRAN of the fixed-point and floating-point designations is overcome in ALGOL by declaring each identifier as type *integer* or *real*, respectively.

The FORTRAN subroutine corresponds to the ALGOL *procedure*. However, ALGOL procedures, unlike subroutines in FORTRAN, have the property of being able to invoke themselves.

The FORTRAN DO corresponds to the ALGOL *for*; however, ALGOL has the major advantage in that the variable can be stepped by a negative quantity and not necessarily in steps of one only.

The ALGOL conditional statements are much more powerful than those of FORTRAN. In ALGOL compound statements may be used to assign values in an *if* or *for* statement, in which case *begin* and *end* are used as delimiters, e.g.,

if a = 0 then *begin* $p := 0$; $q = 1$ *end else begin*

$p := 1$; $q := 2$ *end*;

Multiple nesting of *if, then, else,* statements are also permitted and with the above feature make them a very powerful improvement over that permitted in FORTRAN.

In FORTRAN labels are numerical only, whereas in ALGOL labels may be an identifier or an unsigned integer. ALGOL syntax requires that labels be followed by a colon. Ambiguity of numerical labels is illustrated in the following designational expression:

if $XYZ < C$ *then* 13 *else* *r* [*if* $a > 0$ *then* 2 *else* *n*]

It is not obvious at first sight that 13 is a label but 2 is not.

All of ALGOL is written on a single level of a line. Subscripts and superscripts are accommodated by the use of brackets; "( )" are used for superscripts and "[ ]" are used for subscripts. Several levels of subscripts may be denoted by their use, e.g., the mathematical expression $x^{2}_{k_1, k_2^{2}} + x^{-5}_{k_1^{3}, k_2}$ is written in the ALGOL reference language as $x[k[1],k[2](2)](2)+x[k[1](3),k[2]](-5)$; The one exception to the above arises in exponentiation. In ALGOL the FORTRAN 1.238E7 is represented as $1.238_{10}+7$, the Burroughs ALGOL hardware representation uses the notation $1.238 \times 10^{*}7$. It is to be wondered that the ALGOL committee did not accept the simpler FORTRAN notation. In ALGOL exponents are required to be integers.

Unlike COBOL, ALGOL has very few reserved words (identifiers), these being the common standard functions such as abs(E), sqrt(E), sin(E), etc., where E may have a current value of x, giving abs(x), sqrt(x), etc. These functions may be available through a library of subroutines. At this point one may inquire why the special symbols such as *begin, real,* etc., could not have likewise been made reserved, thus avoiding the necessity to have to underline them or to print them in bold face.

Spaces or blanks have no significance, thus ALGOL can be arranged in a form easy to read. The only exception is in the use of strings where the symbol "␣" denotes a space. A well-organized ALGOL program is quite superior to FORTRAN or COBOL in readability. Indentation is not part of the language; however, it is used to enhance readability of program. Comments can also be used, which are ignored by the computer, to provide the program with documentation.

ALGOL is notably lacking in the provision for complex numbers and multiple precision and has been severely criticized for the lack of the former. It is to be noted that the Russian version of ALGOL does have provision for complex numbers.

Another criticism leveled at ALGOL is that (though the recursive feature is an important one) the recursive use of procedures without prior statement, whether this is needed or not in a particular case, necessitates provision for it in the compiler, with consequences of additional expense in storage space and time.

ALGOL also gives rise to side effects. "Side effects arise principally because there is no limitation on the extent to which functional procedures may bring about the modification of variables whether listed among their parameters or not. A consequence of this is that, when a general expression is being evaluated, the meaning of later terms or subexpressions may be changed by side effects occurring during the evaluation of earlier terms or subexpressions" (Ref. 3). A knowledgeable programmer may use side effects to perform programming tricks. Though this is essentially an added flexibility, it should be relegated to machine language programming and not to automatic programming languages where the ordinary user may be confounded by their occurrence.

VI. COBOL (Common Business Oriented Language)

Business-type programming languages, as opposed to scientific problem oriented languages which were designed to express numeric calculations and possess an algebraic command structure, are characterized by the handling of a great mass of data with relatively little computation required. The command structure of COBOL is similar to that of the English language. The language is based on English, using English words and certain rules of syntax derived from English; however, being a computer language, it has to be precise. The COBOL language was developed through collaboration of computer manufacturers and users, in cooperation with the United States Department of Defense. The language was first described in a report to the Conference on Data Systems Languages (CODASYL) in April 1960 and is hence known as COBOL 60. Further additions and refinements produced COBOL 61 in 1961. COBOL 61 has been defined by CODASYL as consisting of two main portions:

(1) "*Required COBOL* consists of that group of features and options ... which have been designated as comprising the minimum subset of the total language which must be implemented."

(2) "*Elective COBOL* consists of those features and options, within the complete COBOL specifications ... whose implementation has been designated as optional."

Various CODASYL group committees have been established for the maintenance, additions, and improvements to COBOL. However, since 1961 no changes have been announced.

The formulation of COBOL has with reference to the segregation of its divisions been perhaps more methodical than other programming languages. COBOL is divided into four main divisions corresponding to the four different categories of information, namely:

(1) Identification division

(2) Environment division

(3) Data division

(4) Procedure division

In the identification division the programmer and the program is identified; the environment division contains the description of the hardware used (memory size, input output units, etc.); the data division specifies the structure of the data files, records, and entries. The procedure division describes the program to be executed, and is similar in structure to FORTRAN or ALGOL. The main features of the above is that it separates the computer-dependant statements of the program from those that are computer independent. Thus, a program on one system may reasonably easily be converted for use on another by adjustment of statements in the Environment and Data divisions. The above is one of the most important features of COBOL.

The language notations of COBOL are similar to those of FORTRAN, ALGOL, or PL/I. Like FORTRAN, COBOL requires a programming form and statements have to be numbered; however, in COBOL each and every statement must be numbered sequentially. This can cause reassignment of all the numbers should one line in the program be required to change.

Statements in COBOL need to be terminated with either a period, a comma, or a semicolon, whereas ALGOL and PL/I require only semicolons to terminate their statements.

COBOL allows words to be connected with hyphens, just as in PL/I in which words are connected to form composite words by the use of the underscore.

One of the restrictions on the use of the blank that is found in COBOL, perhaps due to the noncomputational nature of the language, is that, unlike FORTRAN, ALGOL, and PL/I, it requires that blank characters must surround arithmetic operators.

In COBOL as in PL/I, key words have preassigned meanings. In COBOL, key words are reserved for their intended purpose and cannot be used for other purposes. In writing programs, key words should be underlined, analogous to boldface words in ALGOL. Both key and nonkey (noise) words are required to be spelled correctly. Indentation may be used to enhance readability of program. The use of noise words is not necessary; however, they provide the necessary documentation for the understanding of the program by a nonuser. This also leads to verbosity. Comments in COBOL are restricted to the Procedure division and written in a NOTE statement, whereas other languages allow comments to appear throughout the program. One of the hindering features of COBOL is that it has several hundred reserve words, and in writing a program one has to constantly refer to the list of reserved words. Conditional statements are easier to understand in COBOL than in any of the other languages.

COBOL requires that all data be described in the Data division along with all programmer-supplied words. Though this is a notable feature of COBOL, it contributes toward the loss of programming flexibility.

COBOL restricts the table (array) size to three dimensions. Also, there is no concatenation operator in COBOL.

In general, COBOL programs are easy to read, perhaps because programs are relatively less complex and are hard to write due to the verbosity. However, in other dialects of COBOL, such as Rapidwrite, often-used words may be coded and effectively abbreviated. This aspect of coding can be extended so that COBOL may be written in a language other than English by the compilation of suitable dictionaries or synonym tables. Most of the features of COBOL have been improved upon and incorporated in the most recent IBM language PL/I.

VII. LISP 1.5 (List Processing Language)

LISP was first developed by John McCarthy and others of the Artificial Intelligence group at MIT (1960). It underwent several stages of development and simplification and is essentially based on a scheme for representing the partial recursive functions of a certain class of symbolic expressions (Ref. 4).

Until now, programming languages have been concerned with solving problems using algebraic equations, statements, and logical operations. Generalization of techniques for processing string-type data, editing, and listing led to the formulation of LISP. Applications to which LISP is particularly suited are those involving symbol manipulation, theorem solving, game solving, integral and differential equations, and heuristic problems. Several list processing languages exist, such as IPL V, COMIT, SLIP, and others. LISP is probably the most powerful; however, it does not have as good string manipulation properties as COMIT, for example.

The primary aim of list processing languages is to locate elements of lists corresponding to certain properties and to ascertain truth or falsehood or other properties of special functions as applied to certain lists. The important operations in list processing are the following (Ref. 5):

(1) Creation of a list or stack

(2) Location of a list or stack when given its symbolic name

(3) Ability to insert a sublist or cell into a specified location in a list structure and a similar ability to delete

(4) Store or retrieve a cell in a stack ("push-down" and "pop-up")

(5) Ability to iterate through a list structure according to a specified tracing mode and to terminate the iteration by use of a relation on a cell contents

(6) Unlimited-indirect and direct positional or symbolic addressing of cells in a list structure

(7) Ability to retrieve or store the link or information portion of a list cell

(8) Ability to return list structures, sublists, cells, or stacks to the list of available space

(9) Relations on contents and structure which set indicators that can be tested for branching and other purposes

(10) Elementary arithmetic capabilities

(11) A "book-keeper" that maintains the available space list and symbolic-name-versus-address table

LISP consists of two divisions called

(1) *Data language* which describes the lists, sublists, elements, items, etc., to be operated on, and

(2) *Metalanguage*. These are operations to be performed on the Data language. To avoid confusion in semantics it would have been preferable if this section had been referred to as the Function language.

The heart of the data language is the s-expression, which is defined thus:

<s-expression> ::= <atomic symbol>|<s-exp. s-exp.>

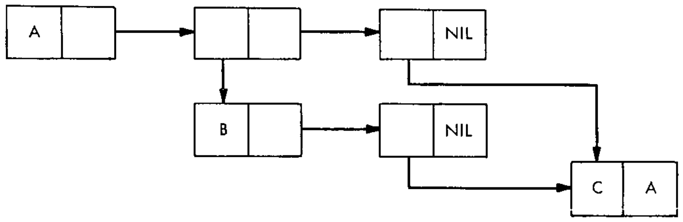
where an atomic symbol is a single element of a list or the name of a list, beginning with an alphabetic character, containing no blanks, and cannot be decomposed.

In LISP, blanks and commas are interchangeable; e.g., $(A\ B\ C)$ is equivalent to (A,B,C), which is a symbolic expression, and A, B, and C are atomic symbols. A special symbol NIL meaning "nothing" is used and is a useful function for the purpose of definition.

LISP uses extensively the functional notation, i.e., a function and its arguments. When the specific number and type of arguments are given, the function will be evaluated, and the necessary housekeeping of assigning locations to the arguments is automatically taken care of. Five basic functions are defined in LISP: *cons,* which construct a larger expression from two smaller ones; two breakup expressions *car* and *cdr,* which obtain the first part and last part of a composite expression, respectively; two predicates *eq,* which check if the two arguments are the same; and *atom,* which checks if the single argument is an atomic symbol. From these five basic functions, by means of composition conditional expression and recursion, other functions can be defined.

For example, if $\boxed{X \mid Y}$ denotes a word, the first cell being the address and the second the decrement, then *car* (X,Y) is the address of X, and *cdr*(X,Y) is decrement Y. LISP also features logical connectives e.g., $p \wedge q = (p \rightarrow q;\ T \rightarrow F)$; and $p \vee q = (p \rightarrow T;\ T \rightarrow q)$. The *cond* function $(p_1 \rightarrow e_1;\ p_2 \rightarrow e_2;\ \cdots\ p_n \rightarrow e_n)$ is analogous to saying if p_1 then e_1 else if p_2 then $e_2 \cdots$.

A possible structural representation of a list such as $(A(B(C.A))(C.A))$ is the following:



In LISP the linkages of a list structure can become very complex, and it is not feasible for the programmer to keep track of the locations of the various lists. To take care of this the language itself automatically assigns locations to the newly created lists, and makes the cells of the erased lists available. This is achieved by scanning the entire memory, after the available cells are exhausted, and then collecting those cells which are not connected to any list structure of the program.

Though LISP and similar languages offer a convenient way of dealing with such non-numeric data processing applications as pattern recognition, information retrieval, decision making, etc., they are not easy to follow, and successions of the functions *car, cons, cdr* abbreviated in the form, e.g., *caddadaadar* are not always easy to decode. Also, the use of parentheses require painstaking exactness in order to have them balanced. However, in order to perform these functions using the languages previously discussed would require voluminous programs. It is also feasible for implementing LISP, e.g., in ALGOL, and to make it an adjunct to the higher languages discussed, thereby increasing their power considerably.

VIII. PL/I (Programming Language I)

PL/I was originally called NPL (New Programming Language), but due to the pressures from the British National Physical Laboratory, IBM decided to label it PL/I. It is the latest higher level language to emerge from IBM. PL/I is a general-purpose language that enables the programming not only of scientific and commercial applications, but also of real-time and systems applications. Since PL/I is broad and comprehensive, a feature of importance is that it may be taught as well as used in subsets of the language. Though PL/I retains some of the FORTRAN features, its structure and descriptive vocabulary (key words) are essentially those of ALGOL, e.g., a subroutine is referred to as a PROCEDURE. Due to the large variety of operations PL/I contains more key words than FORTRAN. However, these key words are not reserved and may be used as variables and labels.

Commercial data processing features bear many similarities to COBOL, as have been pointed out in the prior discussion of COBOL. PL/I also possesses features of list processing, though not as extensively as LISP. Most of the procedures in LISP can be done in PL/I.

Among new features and concepts in PL/I are the provision of statements to use interrupt facility of modern computers as well as facilities for running asynchronously, tasking, multiprocessing. In calling procedures, control is passed asynchronously to the next statement before the called procedure has been executed. The control system treats the called procedure as an independent task and assigns it a priority. On interrupt the task with highest priority is given control.

In PL/I storage can be STATIC, AUTOMATIC, or CONTROLLED. By means of these the programmer has the capability to make storage allocation over complete procedure, or local to a block, or special, depending on execution of data.

The character set of PL/I is larger than that of FORTRAN, though smaller than that of ALGOL. PL/I does not use lower case letters as in ALGOL. Unlike FORTRAN and similar to ALGOL, PL/I is written in free form and may be indented for readability. Statement numbers are not used in PL/I, statements are ended by means of a semicolon as in ALGOL, and more than one statement may be written on a line. (Also, the multiple assignment statement of ALGOL $A:=C:=B$; is incorporated in PL/I as, $A,C=B$;). Comments are used in PL/I by means of the delimiter "/*" (e.g., /* THIS IS A COMMENT */). Alphabetic labels are used in PL/I, numeric labels being forbidden. Subscripted label variables for purposes of switching is also a feature of the language; e.g.,

```
DECLARE LABEL(10);
LABEL(3): A=1;
```

Identifiers can be 31 characters long as compared to 6 in FORTRAN. To improve the clarity and mnemonic significance of names the underscore "_" may be used (when using the 60-character set) as a connective within a name. The syntax of identifiers is similar to FORTRAN.

In normal reading and writing of natural languages, blanks are of significant importance and (unlike FORTRAN) this has been recognized in PL/I. Since there are no reserved words in PL/I, except for relational operators such as GE, GT, NOT, etc., (note that control words like GO TO are not reserved; the meaning is derived from context), and since key words may be used as names, it is imperative that words in PL/I be separated by one or more blanks; e.g.,

```
DOI = 1       is an assignment statement.
whereas
DO I = 1 · · ·     is a DO statement.
```

It is not necessary that operators be surrounded by blanks, unless the operators have been transliterated using characters (e.g., LE is transliterated in the 48-character set as <=).

Though identifiers in DECLARE statements of PL/I may have many attributes (e.g., base, scale, precision, range, etc.), the language has the characteristic flexibility that the user need not specify details which are not of interest. Rules such as "if data unspecified then consider it arithmetic," "if no base specified then assume decimal," etc., then take over. If however, as in FORTRAN the names of variables begin with I through N and their characteristics are undeclared, then they are assumed to be of type integer (fixed point of FORTRAN). This property is useful when using integers for indexing and counting.

Specific virtues of PL/I, such as use of prefix operators, the use of "*" in a subscript position to indicate evaluation of arrays for all values of subscripts, bit-string operators, etc., are seemingly endless. PL/I allows powerful DO statements, such as

```
DO I=10 TO .01 BY -.001;
```

It is to be noted that the commas in FORTRAN DO statement are here replaced by TO and BY. The DO statement has also been extended to control a loop from, e.g., 1 to $J-1$ and then from $J+1$ to N. Some of the other control statements and properties of PL/I are illustrated in the following compact program for evaluating the roots of the equation $ax^2+bx+c=0$ (Ref. 6):

```
/*FIND ROOTS OF EQUATION
  A*X**2+B*X+C=0*/
DECLARE (R1,R2,F)COMPLEX;
IF A=0 THEN LINEAR:  R1,R2=-C/B;
ELSE QUAD:  DO;D=B**2-4*A*C;
E=-B/(2*A);
IF D=0 THEN R1,R2=E;
ELSE IF D>0 THEN REAL:  DO;
    F=SQRT(D)/(2*A);
    R1 = E+F; R2 = E-F;
    END;
  ELSE IMAG:DO;
    F=SQRT(-DO/(2*A)*1I;
    R1=E+F; R2=E-F;
END QUAD;
```

FORTRAN input/output has been expanded upon greatly in PL/I in order to incorporate COBOL-type programming and list processing. Input/output has also been simplified to the extent that READ and WRITE statements need have no FORMAT specification at all, the format being deduced from the data. Four kinds of READ and WRITE statements exist; namely, data-, list-, format-, and procedure-directed. Format specifications are, in principle, similar to those in FORTRAN. Apart from these, input/output facilities also include report generation, editing, and buffer control.

The two criticisms leveled against ALGOL, namely that all identifiers have to be declared and that all procedures have the recursive property, whether desired or not, have both been avoided in PL/I. PL/I allows implicit declaration and all procedures in order to be recursive have to be so declared, thus minimizing potential side effects. However, PL/I may be criticized for its restricted character set which does not provide lowercase alphabetic characters as in ALGOL. Lower case characters would help to make commercial and non-numerical type outputs more conventionally readable.

Many of the PL/I features (e.g., report generation, complex arrays, etc.) are not yet fully available. Software for the IBM 360's yet remains to be completely debugged. It has been voiced by IBM that they would like FORTRAN and COBOL to be replaced by PL/I. It should be noted that the diagnostics of PL/I are perhaps more powerful than those for any other language. The underlying philosophy is that in the event of an error, the compiler should deduce the correction and make every effort to proceed with compilation.

To say the least, PL/I is a challenge to the other higher level languages, and hopefully will motivate them to accelerate their growth.

IX. Conclusion

Several powerful programming languages have been described in this paper that attempt to facilitate communication between man and machine. In the short time since computers came into their own, the growth of programming languages has been prolific. A significant if not an equal portion of the cost of development and operation of computer systems is incurred in providing software.

In order to develop an effective programming language, the program innovator must not only be familiar with broad areas of applications such as scientific, business, and symbol manipulation, but must also be familiar with the intimate inner workings of compilers and computers. His mathematical and logical disciplines must be enhanced by those of grammar, polemic, and linguistics.

The possibility of standardization of computer languages is still remote, and in the author's opinion will remain so until standardization in hardware applied to standard problems is first achieved. However, neither is the development of hardware at a standstill. Great effort is being made in the development of more effective input/output facility with a computer; effectively, an attempt is being made to give computers new "senses" in order that their input/output perception may be enhanced. Breakthroughs in character recognition and speech recognition would open new avenues of communication with the computer, resulting in hopefully simpler and more versatile programming languages. Conception of computers based on other than binary arithmetic systems, such as signed-digit arithmetic, residue numbers (or even the development of a multi-state device), would, it is conjectured, affect the evolution of programming languages to a significant degree. Finally, since man is so strenuously trying to simulate the behavior of human beings in the computer, it is hoped that he will improve his understanding of both himself and the machine.

References

1. Kilner, D., "Automatic Programming Languages for Business and Science," *Comput. Bull.*, Sept. 1962.
2. Sammet, J. E., *Programming Languages. History and Fundamentals.* Prentice-Hall, Inc., New York, 1969.
3. Strachey, C., and Wilkes, M., "Some Proposals for Improving the Efficiency of ALGOL," *Commun. ACM*, Vol. 4, pp. 488–491, November 1961.

References (contd)

4. McCarthy, J., "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I," *Commun. ACM,* Vol. 3, pp. 488–491, November 1960.

5. Chai, D., and DiGuiuseppe, J., *A Study of System Design Considerations in Computers for Symbol Manipulation,* Final Report to the National Science Foundation. University of Michigan, May 1965.

6. *A Guide to PL/I for FORTRAN Users,* IBM Technical Publication C20-1651-0, 1966.

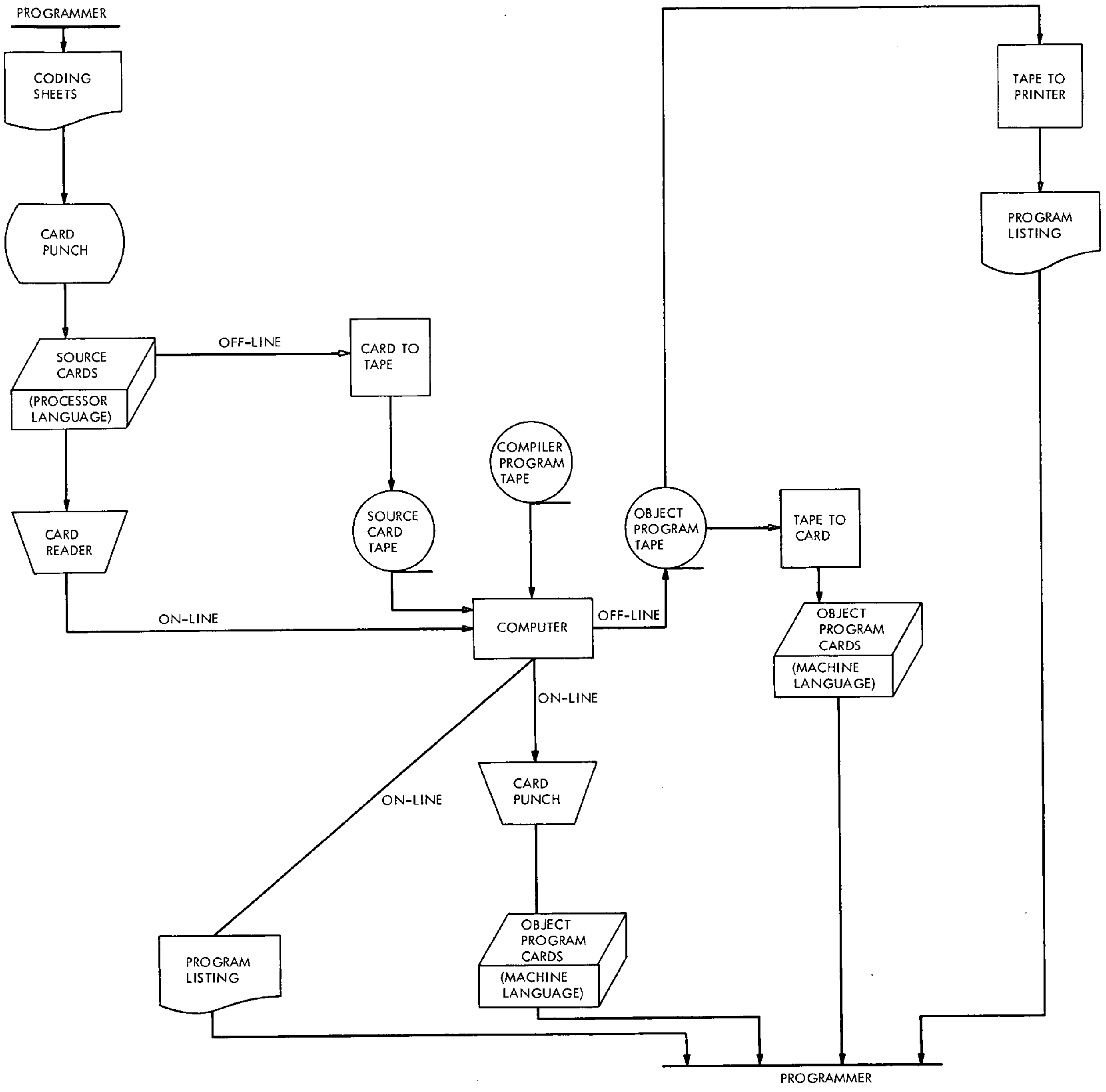


Fig. 1. Process of compiling an object program

Bibliography

General

Bromberg, H., "Standardization of Programming Languages," *Datamation*, August 1963.

Burck, G., *The Computer Age*, Academy Guild Press, Fresno, Calif., 1965.

Pearcey, T., "Aspects of the Philosophy of Computer Programming," *Comput. Bull.*, March 1964.

Shaw, C., "The Language Proliferation," *Datamation*, May 1962.

Wegner, P., *Introduction to System Programming*, Academic Press, Inc., New York, 1964.

FORTRAN

Backus, J., and Heising, W., "FORTRAN," *IEEE Trans. Elec. Comput.*, Vol. EC-13, 1964.

Backus, J. W., et al., "Revised Report on the Algorithmic Language ALGOL 60," *Commun. ACM.*, No. 6, pp. 1–17, January 1963.

"FORTRAN IV Language for IBM 7090/7094 IBSYS Operating System," IBM Systems Reference Library, C28-6390-3.

Golde, H., "FORTRAN II and IV for Engineers and Scientists," The Mac-Millan Company, New York, 1966.

McCracken, D., "A Guide to FORTRAN IV Programming," John Wiley & Sons, Inc., New York, 1965.

ALGOL

Clippinger, R., "ALGOL—A Simple Explanation," *Comput. Automat.* Vol. 11, 1962 pp. 17–19.

Dijkstra, E., "Operating Experience With ALGOL 60," *Comput. J.*, Vol. 5, 1962.

Higman, B., "What Everybody Should Know About ALGOL," *Comput. J.*, Vol. 6, 1963.

McCracken, D., *A Guide to ALGOL Programming*, John Wiley & Sons, Inc., New York, 1962.

Naur, P. (Ed.), "Revised Report on the Algorithmic Language ALGOL 60," *Computer J.*, Vol. 5, 1963.

COBOL

Clippinger, R., "COBOL," *Comput. J.*, Vol. 5, No. 3, Oct. 1962.

Cunningham, J., "Why COBOL," *Commun. ACM*, Vol. 5, p. 236, May 1962.

COBOL, Department of Defense, Washington, D.C. 1965.

Bibliography (contd)

COBOL (contd)

McCracken, D., *A Guide to COBOL Programming,* John Wiley & Sons, Inc., New York 1963.

Sammet, J., "Basic Elements of COBOL 61," *Commun. ACM*, Vol. 5, pp. 237–253, May 1962.

LISP 1.5

Weizenbaum, J., "Symmetric List Processor," *Commun. ACM,* Vol. 6, September 1963.

PL/I

A PL/I Primer, IBM Technical Publication C-28-6808-0, 1965.

A Guide to PL/I for Commercial Programmers, IBM Technical Publication C20-1651-0, 1966.

PL/I Language Specifications IBM Operating System/360, IBM Systems Reference library, C28-6571-2.