BULLETIN 5000-21001-D
Revised December, 1961

AN INTRODUCTION TO ALGOL 60

for the B 5000 Information Processing System

Equipment and Systems Marketing Division
Sales Technical Services

Burroughs Corporation
Detroit 32, Michigan

TABLE OF CONTENTS

APPENDIXES

PREFACE

One of the languages which can be used to program the BURROUGHS B 5000 Information Processing System is ALGOL 60 (the abbreviation for ALGOrithmic Language, 1960 revision). This manual will introduce the reader to the use of ALGOL 60 for the statement of problems to be processed by the BURROUGHS B 5000. The material presented here is introductory; it does not describe all the rules of ALGOL 60 nor itemize all the details of the rules it does cover.[1] Its purpose is to present enough information in such a way that the reader will be able to write elementary ALGOL programs for the B 5000. It is assumed that the reader has some knowledge of mathematics, though it might not be more than the high school algebra he once studied.

The manual introduces the basic elements of ALGOL 60 and specifies the fundamental rules for combining these elements into statements and the statements into complete ALGOL programs. Learning to write programs in ALGOL is similar to learning to write compositions in English. In both cases, one must know what the given basic elements are and learn the rules for using the basic elements to construct more complicated forms.

[1]Those interested in pursuing the details of ALGOL 60 will find its syntax completely described in *Communications of the ACM,* May, 1960, pp. 299 - 314.

SECTION 1
JUST WHAT IS ALGOL?

Someone has said, "Mathematics is the science of computing, but of computing as little as possible." Undoubtedly he was thinking of the enormous reduction of labor that is realized, for instance, in solving a simple expression for the area under a parabola, instead of tediously counting little squares. The same thought applies to the use of a digital computer. We often use a computer to perform highly repetitious calculations which could not be done in any other practical way.

For a long time, however, the use of these accurate, high-speed electronic devices has been bogged down in the difficult job of instructing them—the process known as programing. Since computers had to be instructed step by step on the most basic level, each programer found himself repeating in a general way many things that another programer using the same machine had to do *for himself:* getting information into the machine, operating on it in some way, and printing out the results. In spite of many similarities in their programs, there was virtually no possibility of one programer making use of another's work.

ALGOL changes this situation. It provides a person who needs to use a computer with a means of expressing himself in easy-to-understand, commonsense terms and relieves him of the need to understand the details of the computer's operation. The programing methods used previously, all of which were different from machine to machine, now become the concern only of the automatic system which turns the programer's directions into a set of instructions which the machine can "understand." Although ALGOL is not the first of such languages, the earlier ones were so closely associated with particular machines that they were not applicable to other computers, even those of the same manufacturer. ALGOL, on the other hand, is intended to be completely general and thus independent of any particular computer.

In preparing any program, there are stages of planning. The first is adapting the problem to the demands of a digital computer. This problem is much the same whether a program is written for a particular machine or in more general terms.

The second part of preparing a program is much more dependent on the nature of the machine. For any computer, however, if the programer must use the machine instructions directly to prepare his program, he must exercise the most scrupulous care to insure that he has not committed any of many possible errors: computing with values obtained from a wrong location in memory, setting up the wrong qualifications to enter or leave a portion of the program which is used repeatedly, ad infinitum. This part of the programer's job is called "housekeeping," an apt name because of its connotations of repetition and drudgery.

When the special program called a compiler (built into the B 5000) transforms an ALGOL program into one which the computer can employ, it assumes these responsibilities; housekeeping conventions have been previously established — prepackaged, if you will — and the compiler performs virtually all of these duties, relying on the dependable performance of the computer to maintain accuracy.

With this indication of some of the advantages of using ALGOL to program a digital computer, let us examine the role of the language itself in the program.

Dr. Herriot of Stanford University has made an interesting distinction between a mathematical statement of a problem and its ALGOL counterpart. The mathematical statement is static; the ALGOL statement, on the other hand, is dynamic. The ALGOL statement, however complex it may be, describes an actual step-by-step procedure for obtaining a solution to a problem, using actual numbers and achieving a number as a result. For example, when we write the equation which gives the hypotenuse of a right triangle

$$C = \sqrt{A^2 + B^2}$$

we do not qualify in any way the order in which we will arrive at the sum of the squares, the units we will use in performing the calculation, or the means of determining the square root of the sum $A^2 + B^2$. The ALGOL equivalent for this would be written as

C ← **SQRT** (A*2 + B*2)

Now there are many questions which may occur to you; after all, what we have written here in ALGOL *resembles* the algebraic equation which we wrote, but it is certainly not altogether like it.

We can easily recognize the C, A, and B from the algebraic equation; in ALGOL they belong to a class of symbols called *identifiers.* Identifiers are used in ALGOL much as in algebra, where a symbol is used to denote a value which has yet to be calculated, so that it can be distinguished from others with which we are working. A large number of such identifiers can be written in this symbolic way, drawing on letters of the alphabet, and numerals if desired; it is necessary that the first character of an identifier always be a letter, and that there be no spaces to break up the identifier.

Examples of identifiers:

A
B10
PRESSURE
A1B2C3D4E5

To go back to our example

C ← **SQRT** (A*2 + B*2)

the symbol + is used in ALGOL in an easily understandable way; it has the same significance as the plus sign in the algebraic statement

$$C = \sqrt{A^2 + B^2}$$

that is, addition. The class to which the plus sign belongs is that of *arithmetic operators.* There are, of course, other operators in ALGOL, the arithmetic operators being:

subtraction	−	(minus sign)
multiplication	×	(cross-product sign)
division	/	(solidus)
exponentiation	*	(asterisk)
division	**DIV**	

(The last division operator is a special one which will be described after the discussion of the number systems employed in the B 5000.)

From our list, we now can see the significance of the asterisk in our ALGOL equivalent. Where in mathematical notation we wrote A^2, in the B 5000 version of ALGOL we write A*2. The 2 here is called the *exponent* of A; if we were using an expression such as X^Y, we would write its ALGOL equivalent as X*Y, and so forth. More complex expressions are possible and will be discussed later.

SQRT in our ALGOL equivalent

C ← **SQRT** (A*2 + B*2)

means "square root," and is equivalent to $\sqrt{}$ or to the power of 1/2 in mathematical notation. When we use the symbol $\sqrt{}$ in mathematics, we indicate its extent by adding a vinculum (overbar) $\sqrt{\quad}$; when we use $^{1/2}$, we indicate the extent by parentheses. The parentheses in our ALGOL statement thus perform the same task as when they enclose any mathematical expression.

We are left with one symbol to be explained, the left-pointing arrow:

C ← **SQRT** (A*2 + B*2)

This arrow is called the *replacement operator.* It replaces the current value of the identifier C with the value of the expression to its right, allowing the programer to use C whenever he has need to refer to the value of $\sqrt{A^2 + B^2}$. Although the left-pointing arrow is often placed where we would find an equal sign (=) in algebra, it is important to emphasize (for reasons which will become apparent later) that *the replacement operator is not entirely equivalent to the equal sign in mathematical notation.*

To extend the discussion of **SQRT,** it is necessary to say that the method used to perform the calculation is made available from a section of the compiler called the library, and is thus termed a *library function.* The programer need only write the name of a library function in order to make use of it. There are several other such functions, some of which compute trigonometric functions, others the logarithms, and so forth. To avoid confusion, names of these functions naturally cannot be used for any other purpose, so care must be taken not to use their names to designate anything else in the program. The ALGOL class to which **SQRT**, the names of the other library functions, and certain other words belong is called *reserved words,* a list of which appears in APPENDIX D. (Reserved words appear in this text in boldface.)

Writing a program in ALGOL closely resembles the detailed problem definition required if one were to give a problem to a machine-language programer for computer solution. The language and format of the problem definition have been standardized. The computer itself produces the machine-language program, as well as doing the processing. Therefore, it is particularly important to remember that the machine-language programer (the compiler which is in the computer) is not familiar with the problem and knows about it *only* what is contained in the problem (the ALGOL program). The problem must, therefore, be complete, unambiguous, and expressed in an acceptable form and terminology. The form will approximate the well-known: "Given; To Find; Calculations; Solution." The terminology will be ALGOL.

It is hoped that this very short treatment will serve to indicate some of the properties of ALGOL, but it must be understood that the example we have discussed is quite rudimentary. As the reader continues in this manual and finds his proficiency increasing, he will likely be struck again and again by the similarity of ALGOL to a natural language such as English. Just as English serves to write either

"See the cat. The cat is on the table."

or to discuss highly abstruse subjects, so ALGOL lets us express problems ranging from the example above to problems of great complexity. When we begin learning to write in ALGOL, though, we can just as well begin on the level of "See the cat."

SECTION 2 BASIC DEFINITIONS

SYMBOLS

The symbol set which is used in this manual consists of:

the capital letters A through Z
the digits 0 through 9
punctuation symbols , : ; ( ) [ ]
operational symbols (to be listed as introduced)

The complete list of symbols used in BURROUGHS ALGOL 60 appears in APPENDIX A.

NUMBERS

A number in BURROUGHS ALGOL 60 is written as a string of from one to eleven decimal digits.[2]

ALGOL allows the use of two types of numbers, called type **REAL** and type **INTEGER.** Type **REAL** numbers are those which include a decimal point. Type **INTEGER** numbers are the whole numbers, that is, those which do not include a decimal point. Both types include positive numbers, negative numbers, and zero.

The numbers which result from calculations are of one or the other type, depending upon the type of the numbers which went into the calculation and upon the nature of the calculation itself. The rules which determine the type of these results appear in the discussion of arithmetic expressions.

IDENTIFIERS

Identifiers are used in ALGOL programs as names, for purposes of reference.

Identifiers consist of at least one letter, followed by any letters, digits, or combination of letters and digits. No spaces may appear as part of an identifier. Some examples of identifiers are:

X	X1	SUMX	A	T
Y	Y2	AVERAGEX	ALT	T1
M	M53	LC	ALTITUDE	TIME
N	N4AND5	LIFTCOEFF	B3JFG29Z	TIME1

[2]For problems requiring greater precision, the number size can be extended to 23 decimal digits.

Identifiers are commonly used to name the variables and constants which appear in mathematical formulas. Identifiers are also combined with numbers, punctuation, and operational symbols to form the statements, expressions, declarations, etc., of a complete ALGOL program.

REPLACEMENT OPERATOR

The replacement operator symbol is a left-pointing arrow (←). It indicates that the value of whatever stands to the right of the arrow is to replace the value of the variable to the left of the arrow. Thus the statement

X ← Y

tells the computer to replace the value of X with the value of Y. The complete construction—the replacement operator with its left- and right-hand parts—is called an *assignment statement;* when executed by the computer it assigns the value as indicated above.

If a problem solution requires frequent reference to a constant (such as π), the programer may wish to use an identifier (such as PI) in his calculations. To do so, he might write the assignment statement

PI ← 3.14159

after which he may employ π in his program by simply using the identifier PI, which has been assigned the value 3.14159.

A common requirement in many problems is the setting of initial values. For example, if sums (used for tallies or totals) are calculated, it is necessary to set the sums to zero before the first pass through the calculation. The assignment statement

SUMX ← 0

replaces the value of the variable identified by SUMX with zero. If several variables must be set to a common value, they may be strung together; for example, the assignment statement

SUMX ← SUMY ← TALLYT ← K ← 0

sets the values of the variables whose identifiers are SUMX, SUMY, TALLYT, and K to zero.

ARITHMETIC OPERATORS AND EXPRESSIONS

The BURROUGHS ALGOL system for the B 5000 uses the following arithmetic operators and symbols:

OPERATOR	SYMBOL
Add	+
Subtract	−
Multiply	×
Divide	/ or **DIV** (two different results)
Exponentiate	*

These operators are used with numbers or identifiers to form arithmetic expressions, such as:

T + 9.4	N − M	B/2	D*3
Z + Y	2 × A	180/PI	2*N
N − 1	BASE × HEIGHT	VEL × TIME	A*B

The sequence of performing a series of arithmetic operations is normally from left to right. However, this process is interrupted in accordance with the following priorities:

First:	operations enclosed by parentheses
Second:	exponentiation
Third:	multiplication and division
Fourth:	addition and subtraction

Since ALGOL 60 *defines* division by a term as multiplication by the inverse of the term (that is: A/B is the same as $A \times B^{-1}$), equal priorities are assigned to multiplication and division, with these operations taking place from left to right as they occur in the program.

The foregoing definition and rule give the single exception from the priorities commonly used in ordinary algebra. For instance, in ordinary algebra the expression A/B × C is usually interpreted as A/(B × C); but in ALGOL 60 it means (A/B) × C. Parentheses must be used in ALGOL 60 to indicate denominators with more than one factor, and may be used as desired to indicate sequencing of operations.

Arithmetic expressions can constitute the right-hand member of assignment statements. Thus we can write

```
   DIST ← VEL × TIME
   AREA ← (BASE × HEIGHT)/2
      Y ← A*2 + B*2
  DELTA ← PREVT − PREST
```

In each case, the number resulting from the evaluation of the expression on the right of the replacement operator is assigned as the value of the identifier on the left.

Remember that only identifiers may occupy the left-hand position in an assignment statement. It would be incorrect to write

C*2 ← **SQRT**(A*2 + B*2)

since C*2 is an arithmetic expression. The programer could write instead

CSQUARE ← **SQRT**(A*2 + B*2)

and the identifier CSQUARE may then be used to refer to

$\sqrt{A^2 + B^2}$.

TYPE OF ARITHMETIC EXPRESSION

The type (**REAL** or **INTEGER**) of an arithmetic expression is automatically determined by the types of its components. Adding, subtracting, or multiplying two operands of type **INTEGER** gives a result of type **INTEGER.** The result will be of type **REAL** if either component is **REAL** or if both components are **REAL.**

As indicated in the list above, ALGOL provides two kinds of division. The operation indicated by the solidus (/), more commonly called the "slash" or "slant," gives a result of type **REAL** for any combination of **INTEGER** or **REAL** components, or both. This operation yields a conventional quotient which may have a fractional part, with the sign of the quotient plus or minus as in ordinary algebra.

The other division symbol, **DIV,** is used only where both operands are of type **INTEGER**; the result is always of type **INTEGER.** The sign of the result is plus or minus as in algebra. The result is always truncated (cut off) to be an **INTEGER.** No rounding is done in this cutting-off process; thus, if the normal quotient would be + 12.83, it is cut off to + 12; similarly, − 12.83 is truncated to − 12. (**DIV** is one of the reserved words. See APPENDIX D.)

For exponentiation, raising a number of type **INTEGER** to a positive power which is also of type

INTEGER will give a result of type **INTEGER.** Any other defined result will be of type **REAL.**

Because a negative number raised to a fractional power may be undefined in mathematics, there are certain invalid combinations for exponentiation. To define all situations completely, a table of valid and invalid combinations is given in APPENDIX B.

It is especially important to realize that although the type (**REAL** or **INTEGER**) of an arithmetic expression is determined by the foregoing rules, the type of the result following insertion of the expression into a statement (after the replacement operator) can change.

Example:

$Z \leftarrow (A + B)/C$

Because division with the slant sign is defined as always giving a result of type **REAL,** the expression $(A + B)/C$ is of type **REAL.** If the type of Z were **INTEGER,** then *this result would be automatically converted* to type **INTEGER,** and appropriately truncated. Conversely, an expression of type **INTEGER** would be converted to type **REAL** in such a statement if the variable to the left of the replacement operator were of type **REAL.**

EXAMPLES OF ARITHMETIC EXPRESSIONS

In the following examples, the left column gives an ALGOL assignment statement which contains an arithmetic expression. The right column gives the algebraic interpretation of that statement. Extra parentheses have been used in some of the interpretations to indicate the complete meanings.

ADDITION

$Z \leftarrow A + B$	$Z = A + B$
$Z \leftarrow E + F + G$	$Z = (E + F) + G$

SUBTRACTION

$Z \leftarrow W - R$	$Z = W - R$
$Z \leftarrow W - R - S$	$Z = (W - R) - S$

MULTIPLICATION

$Z \leftarrow A \times B$	$Z = A \times B$
$Z \leftarrow A \times B \times C \times D$	$Z = \{(A \times B) \times C\} \times D$

DIVISION

$Z \leftarrow A/B$	$Z = \frac{A}{B}$
$Z \leftarrow A/B/C$	$Z = (A/B)/C$

EXPONENTIATION

$Z \leftarrow A*B$	$Z = A^{B}$
$Z \leftarrow A*(-B)$	$Z = A^{(-B)}$

COMBINATIONS OF OPERATIONS

$Z \leftarrow A+B-C+D$	$Z = \{(A+B)-C\}+D$
$R \leftarrow Z\times Y+A\times B$	$R = (Z\times Y)+(A\times B)$
$S \leftarrow A+B\times C\times D$	$S = A+\{(B\times C)\times D\}$
$T \leftarrow A\times B/C\times D$	$T = \frac{A\times B}{C} \times D$
$M \leftarrow A/B\times C-D+E/F$	$M = \{\left(\frac{A}{B}\right)\times C\}-D+\frac{E}{F}$
$Z \leftarrow A*B\times C$	$Z = (A^{B})\times C$
$P \leftarrow A\times B*C$	$P = A(B^{C})$
$V \leftarrow A*B+C$	$V = (A^{B})+C$
$W \leftarrow A*(B+C)$	$W = A^{(B+C)}$
$Z \leftarrow A*(B\times C)-D*(E/F)$	$Z = A^{(B\times C)}-(D)^{\left(\frac{E}{F}\right)}$

SECTION 3
BOOLEAN EXPRESSIONS

Boolean expressions are rules for computing the logical values **TRUE** and **FALSE.** If the condition stated is satisfied, the result is **TRUE,** otherwise the result is **FALSE.** Just as the value of an arithmetic expression is of type **REAL** or **INTEGER,** the value of a Boolean expression is of type **BOOLEAN.**

Boolean expressions employ two kinds of operators: relational and logical.

The relational operators are:

$<$ less than	$>$ greater than
$\leq$ less than or equal to	$\geq$ greater than or equal to
$=$ equal to	$\neq$ not equal to

Relational operators specify a comparison between two terms which may have any arithmetic value.

Examples:

$R > 0$	$R < +.46$
$R + 3 \times Y \leq 2 \times Z$	$Y \geq 12.34$
$R = Y$	$100 \neq Z$

In each example, the entire expression has either the value **TRUE** or the value **FALSE,** depending on whether or not the specified relation holds.

The logical operators used in ALGOL 60 for the B 5000 are:

$\wedge$	and
$\vee$	or
$\neg$	not
EQV	equivalent to
IMP	implies

Evaluating a Boolean expression containing a logical operator involves application of the rule stated for that operator to the operands, which are restricted to the values **TRUE** and **FALSE.** (**EQV** and **IMP** are, of course, reserved words.)

Logical operands must be:

(a) the result of a relational operation, or
(b) the result of a logical operation, or
(c) a declared **BOOLEAN** variable. (Declarations of type are described in SECTION 6.)

The logical operators have the following meanings. (Where the conditions stated are *not* met, the result is **FALSE.**)

$\wedge$	(and)	If *both* operands connected by this operator have the value **TRUE,** the result is **TRUE.**
$\vee$	(or)	If *either* operand connected by this operator is **TRUE,** the *result* is **TRUE.**
$\neg$	(not)	If the operand following this operator is **FALSE,** the *result* is **TRUE.**
EQV	(equivalent to)	If the operands connected by this operator have the same logical value, the *result* is **TRUE.**
IMP	(implies)	If the operands connected by this operator have the same logical value, or if the first (left-hand) operand is **FALSE,** then the *result* is **TRUE.**

A tabular form of the above definitions follows.

Let A and B be logical operands, then:

A	**TRUE**	**TRUE**	**FALSE**	**FALSE**
B	**TRUE**	**FALSE**	**TRUE**	**FALSE**
$A \wedge B$	**TRUE**	**FALSE**	**FALSE**	**FALSE**
$A \vee B$	**TRUE**	**TRUE**	**TRUE**	**FALSE**
$\neg B$	**FALSE**	**TRUE**	**FALSE**	**TRUE**
A **EQV** B	**TRUE**	**FALSE**	**FALSE**	**TRUE**
A **IMP** B	**TRUE**	**FALSE**	**TRUE**	**TRUE**

Examples using the logical operators are given

EXAMPLE	QUESTION POSED
$\neg (B < 100)$	Is it **FALSE** that B is less than 100?
$(A = 10) \wedge (B < 100)$	Is it **TRUE** that A is equal to 10 *and* that B is less than 100?
$(X > Y) \vee (Z > W)$	Is it **TRUE** *either* that X is greater than Y *or* that Z is greater than W?
(A = B) **EQV** (C ≠ 0)	Is it **TRUE** that both relational tests give the same result; i.e., are *both* **TRUE** or *both* **FALSE**?
(M ≠ N) **IMP** (R < S)	Is it **TRUE** that the first (left-hand) relational value is equivalent to the second, *or* that the first is **FALSE**?

The interpretations of the examples are stated in the form of questions because that is the way the programer is likely to use these expressions in his programs. (See Control Statements, SECTION 5.)

In the following examples, A, B, C, and D represent logical values or complete logical operations (such as the results of the preceding examples).

EXAMPLE	QUESTION POSED
$A \vee B$	Is it **TRUE** that *either* A *or* B is **TRUE**?
$\neg (A \vee B)$	Is it **FALSE** that *either* A *or* B is **TRUE**?
$(A \wedge B) \vee (C \wedge D)$	Is it **TRUE** *either* that *both* A and B are **TRUE,** or that *both* C and D are **TRUE?**
$(A \vee B) \vee (\neg C)$	Is it **TRUE** *either* that A or B is **TRUE,** *or* that C is **FALSE**?

There are standard rules for the priority of performing relational and logical operations. While the general rule for performing these operations is to proceed from left to right, this rule is subordinated to the following priority list.

First priority:	Parenthesized operations
Second priority:	Evaluation of arithmetic expressions
Third priority:	Relational operations as met from left to right
Fourth priority:	$\neg$ (not)
Fifth priority:	$\wedge$ (and)
Sixth priority:	$\vee$ (or)
Seventh priority:	**IMP** (implies)
Eighth priority:	**EQV** (equivalent to)

This priority list indicates that in many cases parentheses may be omitted. However, to avoid errors in interpretation and for easier reading, it is recommended that parentheses be used in compound expressions.

SECTION 4
STANDARD FUNCTIONS

A function designator defines a single value which is the result of a specific set of operations on given parameters. Certain frequently used functions have been designated standard functions and incorporated in ALGOL so that the programer need not write the detailed steps to compute these values.

The arguments upon which these standard functions are to operate *must be enclosed with parentheses.* A list of the standard function designators, all of which are reserved words, follows:

STANDARD FUNCTION DESIGNATOR	FUNCTION DESIGNATED
SIN	Sine
COS	Cosine
ARCTAN	Arctangent
SQRT	Square root
LN	Natural logarithm
LOG	Logarithm, base ten
EXP	Exponential function
ABS	Absolute value
SIGN	Sign. According to whether the value of E is greater than zero, equal to zero, or less than zero, **SIGN** (E) is + 1, 0, or − 1.
ENTIER	Transfers from type **REAL** to type **INTEGER** by assigning the largest integer not greater than the value.

Examples:

(1) R← (− B + **SQRT** (B*2 − 4 × A × C))/ (2 × A)
(2) TANY ← **SIN** (Y)/**COS** (Y)
(3) R ← **ENTIER** (Z)

In example (2), **SIN** and **COS** are standard function designators, but TANY is an identifier; the angle Y must be expressed in radians, and TANY will, of course, be a type **REAL** number such as 2.3456789. In example (3), the value of R is replaced by the largest integer which is not greater than the value of the type **REAL** number Z; the sign of R will be the same as the sign of Z.

SECTION 5 OPERATIONAL STATEMENTS

Statements are the sentences of this algebraic language; as in ordinary written English, the order in which they appear is very important. Statements are separated by semicolons. A group of statements may be combined to form a compound statement by preceding the first statement of the group with the word **BEGIN** and following the last statement with the word **END.** It is sometimes necessary to identify a particular statement so that it may be referenced in other statements. To do this, a statement is given a label, which is an identifier followed by a colon. Operational statements fall into one of two general categories: assignment statements and control statements.

ASSIGNMENT STATEMENTS: SUMMARY

Assignment statements have been mentioned in preceding sections. Here these references are summarized for easier comparison with control statements.

Assignment statements contain the replacement operator ← denoting the substitution of

a number, or
the value of an identifier, or
the value of an expression

on the right for the identifier on the left.

Examples:

SUMB ← 0; [Zero replaces SUMB.]

M ← N; [The value of N replaces the value of M.]

A ← B/C − V − Q × S;
[The quantity $\frac{B}{C} - V - QS$ replaces A.]

HYPTNS: C ← **SQRT** (A*2 + B*2);
[HYPTNS is a statement label. The quantity $\sqrt{A^2 + B^2}$ replaces C.]

ROOT: Y ← (− B + **SQRT** (B*2 − 4 × A × C))/(2 × A);
[ROOT is the statement label. The quantity $\frac{-B + \sqrt{B^2 - 4AC}}{2A}$ replaces Y.]

CONTROL STATEMENTS

In the normal sequence of operations, the successive statements are executed as they are encountered. It is sometimes desirable to interrupt this normal sequence, as when one or more statements are to be repeated several times, or are to be executed only under specific conditions. The interruption of the normal sequence is called *transfer of control* since, once the transfer has taken place, successive statement sequencing continues from the new point of reference.

Transfer of control in ALGOL is accomplished through use of the control statement, which may be *unconditional, conditional,* or *iterative.*

Unconditional Control Statements

Unconditional transfer of control statements are formed by following the words **GO TO** with a label which specifies the point in the program where control is to be resumed; some examples are shown below. More general **GO TO** statements are possible, but are not considered in this section.

GO TO BILL;
GO TO COEFLIFT;
GO TO SECONDSTOP;
GO TO START;
GO TO M3L75;

Conditional Control Statements

Conditional control statements cause other statements to be executed or skipped depending on the current values of specified Boolean expressions. Conditional control statements provide the ability to make decisions necessary for the completely automatic solution of a problem.

The conditional control statement may have either of the following formats:

> **IF** Boolean expression **THEN** statement; next statement
>
> **IF** Boolean expression **THEN** statement **ELSE** statement; next statement

NOTE: The statement following **THEN** may not begin with the word **IF.**

In either case, when the relational or logical expression following **IF** (the **IF** clause) is **TRUE,** the statement following **THEN** is executed and control is transferred to the beginning of the next statement, unless the **THEN** statement contains a change of control operation (as shown in examples 2 and 4 which follow). When the **IF** clause is **FALSE,** the **THEN** statement is skipped. In the first case above **(IF ... THEN)**, control is transferred to the beginning of the next statement. In the second case above **(IF ... THEN ... ELSE)**, control is transferred to the statement following **ELSE;** after that statement has been executed, control is transferred to the beginning of the next statement.

Examples:

```
(1) IF P ≤ 0 THEN P ← .5; Y ← 2 × P + 3; ...

(2) IF Y ≤ .0001 THEN
    GO TO OUT; N ← N/2; ...
(3) IF A = B THEN C ← 1 ELSE C ← 1 − A/B;
    D ← C × M; ...
(4) IF (Y ≥ 0) ∧ (Y < .0001) THEN
    GO TO OUT ELSE GO TO CONT;
```

The conditional control statement may contain more than one **IF** clause. In this case, the **IF** clauses are evaluated one after the other in sequence from left to right until one yielding the value **TRUE** is found. Only with a **TRUE** condition is the associated **THEN** clause executed, after which control is transferred to the beginning of the next statement.

Example:

```
KEN: IF A = B THEN C ← 1
     ELSE IF A < B THEN
     C ← 1 − A/B ELSE GO TO REVERSE;
```

Iterative Control Statements

The purpose of the **FOR** statement in ALGOL 60 is to facilitate writing an iterative operation. An operation is said to be iterative when the same statement is to be executed repeatedly a specified number of times or is to be executed for each one of a designated set of values assigned to a variable. The **FOR** statement contains a **FOR** clause and a **DO** statement. The **FOR** clause gives the conditions under which the **DO** statement is to be executed repeatedly zero or more times. (The **DO** statement would be executed zero times—i.e., would not be executed—if the conditions of the **FOR** clause were not satisfied.) The **FOR** statement has the following format:

> **FOR** (variable) ← (**FOR** list) **DO** (statement)

The **FOR** list is composed of one or more elements separated by commas, and gives a rule for obtaining the values which are consecutively assigned to the variable. This sequence of values is obtained from the **FOR** list elements by taking these one by one in the order in which they are written, left to right. There are three kinds of **FOR** list elements: arithmetic expression element, **STEP-UNTIL** element, and **WHILE** element. In defining these, only one-element **FOR** lists will be considered.

ARITHMETIC EXPRESSION ELEMENT

An arithmetic expression alone may be a **FOR** list element and as such indicates that the variable will take on the value of the expression prior to the execution of the **DO** statement.

$\textbf{FOR}\ \text{variable} \leftarrow \begin{array}{c}\text{arithmetic}\\\text{expression}\end{array}\ \textbf{DO}\ \text{statement}; \begin{array}{c}\text{next}\\\text{statement}\end{array}$

When the **DO** statement has been executed, control is transferred to the beginning of the next statement.

```
FOR J ← 3 DO Z ← 2 × J + J*3;
next statement

FOR S ← C + D DO BEGIN M ← S*2;
N ← M + 5;
V ← R/S + L + M × N END; next statement
```

STEP-UNTIL ELEMENT

The effect of evaluating the **FOR** list element **STEP-UNTIL** is similar to the result obtained from counting when given a starting point, a limit, and the increment by which to count. (For example: Count by 2's from 10 through 90.) This element has the form:

starting point **STEP** increment **UNTIL** limit

The starting point, increment, and limit are arithmetic expressions. The statement following the word **DO** in the example shown below is executed once for each value computed by stepping *from* the starting point *through* the limit.

```
FOR variable ← starting point STEP increment
UNTIL limit
DO statement; next statement;
```

In the above form, the following sequence takes place:

(1) The variable is replaced with the value of the starting point.
(2) The variable is compared with the limit. If the variable has *passed* the limit, control is transferred to the beginning of the next statement. If the variable has *not* passed the limit, the statement following **DO** is executed, then the variable is altered by the amount of the increment, and the sequence continues at (2) above.

Note that since the increment may be either positive or negative the limit may be approached from either direction.

Example:

```
FOR A ← 1 STEP 1 UNTIL 10
DO statement;  next statement
```

In this example, the **DO** statement will be executed ten times, after which control will be transferred to the beginning of the next statement.

```
FOR Z ← 3 × Y + 2 STEP 2 × B UNTIL
B*2 + 1 DO statement;  next statement
```

In this example, the **DO** statement will be executed repeatedly until the value Z, which is increased by 2B after each execution, exceeds the value $B^2 + 1$. At that time, control is transferred to the beginning of the next statement. Unlike the previous example, in which the **DO** statement is always executed exactly ten times, the number of times the **DO** statement in this example is executed is not fixed since it is dependent on the current values of B and Y.

Example:

```
APPROX ← 0; FOR M ← Z1 STEP 0.005
UNTIL Z2 DO APPROX ← APPROX +
0.005 × (C × M*2 + D × M + E);
```

WHILE ELEMENT

The **FOR** list element **WHILE** implies duration and may be thought of as representing "as long as." This element has the form:

arithmetic expression **WHILE** Boolean expression

A **FOR** statement containing the **WHILE** element has the following form:

```
FOR variable ← arithmetic expression WHILE Boolean expression

DO statement; next statement
```

The statement following **DO** will be executed repeatedly as long as the Boolean expression following **WHILE** is true.

The following events take place:

(1) The variable is replaced with the value of the arithmetic expression.
(2) The Boolean expression is evaluated. If the result is *not* **TRUE,** control is transferred to the beginning of the next statement. If the result is **TRUE,** execute the **DO** statement, then continue from (1).

Example:

```
FOR Q ← 2 × V WHILE V < 10
DO statement; next statement
```

In this example the value computed from the expression 2 × V replaces Q and the **DO** statement is executed as long as the value of V is less than 10. Note that an exit—i.e., change of control to another statement—from this **FOR** statement is dependent upon either a change in the value of V as a result of the **DO** statement (see the third example at the end of SECTION 6) or the presence of a change-of-control operation within the **DO** statement.

Example:

```
FOR Q ← 2 × V WHILE V < 10
DO BEGIN M ← (Q + 5 × R) × Q;
 GO TO APRIL END; next statement
```

NOTE: In this example, Q is first set equal to 2V. Then, if V is less than 10, the statement following **BEGIN** is executed, and control is then transferred to APRIL. If V becomes equal to or greater than 10, control skips to the next statement instead of being transferred to APRIL.

This series of statements is completely equivalent to:

```
FOR Q ← 2 × V DO IF V < 10
THEN BEGIN M ← (Q + 5 × R) × Q;
 GO TO APRIL END; next statement
```

THE **FOR** LIST

As stated previously, the **FOR** list may contain several elements separated by commas. The elements within a single **FOR** list may be all of the same kind or of different kinds. They are completely evaluated, individually, as they are met from left to right, and the statement following **DO** is executed repeatedly until the **FOR** list is exhausted or until control is transferred to another point in the program as a result of the execution of the **DO** statement. (See the last example in SECTION 6.)

Example:

```
FOR L ← 1, 3, 7, 11 DO statement;
next statement
```

To exit from any **FOR** statement, either of two methods is used, depending on the operation involved. When the final (rightmost) **FOR** list element has been completely evaluated, control is transferred automatically to the beginning of the next statement in sequence. The alternate method is an exit resulting from the execution of a **GO TO** control statement from within the **DO** statement. (In this case, the **DO** statement is probably a compound statement.) If the exit is caused by a **GO TO** statement, the variable retains the value which it had immediately before the exit took place. Otherwise the value of the variable after exit is considered to be "unknown," according to the rules of ALGOL 60, and should not be used.

SECTION 6
DECLARATIONS AND BLOCKS

The main body of an ALGOL program is an ordered list of statements which, when executed, produces the specified solution. As was stated at the beginning of SECTION 5, successive statements may be combined to form a compound statement by preceding the first statement with the word **BEGIN** and following the last statement with the word **END.** The statements between **BEGIN** and **END** are treated as a whole rather than as separate units.

Statements are composed of identifiers, operators, numbers, punctuation marks, and reserved words (e.g., **IF, GO TO**) combined according to the rules of ALGOL. With the exception of identifiers, the individual items represented by each of these terms have specific meanings. Identifiers, because they are arbitrarily selected for and used in one program, have meaning only within that program. Therefore, all identifiers used in a program, except those employed as labels or as the formal parameters of a procedure declaration (see SECTION 7), must be introduced prior to their use; this is done with a declaration which defines certain properties of the identifiers.

TYPE DECLARATIONS

The type declaration defines the type of the variable named by an identifier. The type may be **REAL, INTEGER,** or **BOOLEAN.** The type declaration specifies that *all values which the identifier takes on* must be of the designated type.

The type declaration has the following form:

type identifier, identifier, . . . , identifier;

Examples:

REAL M, Y, Z;
INTEGER C;
BOOLEAN A, B;

BLOCKS

A logical segment of a program is a section of coding which is considered by the programer to be a complete and primarily independent unit. Its independence derives from the fact that a section's elements may have meaning only within that particular section. An entire program is a logical segment and it may contain subprograms which are also logical segments. In ALGOL the logical segment is called a block. A block is defined as a program section which is preceded by the word **BEGIN,** includes at least one declaration and one statement, and is followed by the word **END.** A block has the following form:

BEGIN declaration; statement; . . . ;
statement **END**

A declaration is valid only for the block in which it appears, and has effect throughout that block. All declarations for a block must immediately follow the word **BEGIN,** and any entry to that block must be made at the word **BEGIN.**

Exit from the block, as a result of encountering the word **END** or a transfer-of-control statement (**GO TO**), cancels the declarations made within the block. The identifiers declared in the block, then, have no significance outside the block and may be used for other purposes. If an identifier is further declared with the word **OWN,** upon re-entry to the block it will assume the value it had at the last exit. Except for the values of these **OWN**-declared identifiers upon re-entry to a block, the value of each variable declared within a block is unknown until the variable has appeared to the left of the replacement operator in an assignment statement.

Blocks may be labeled by preceding the word **BEGIN** with an identifier followed by a colon.

Blocks written within blocks are allowed as long as all the preceding rules are strictly followed. For example:

```
Label:  BEGIN declaration; statement;
        statement; ... BEGIN declaration;
        statement END; statement END
```

In this example, the *inner* **BEGIN-END** pair establishes a block within a larger block.

SWITCH DECLARATIONS

A **SWITCH** declaration names a group of alternative points in a program to which control may be transferred as the result of a single **GO TO** statement. The selection of the actual point to which control is transferred depends on conditions existing at the time of the transfer. The declaration contains a list of the labels of the statements to which control may be transferred, a replacement operator, and a separate label by which the **SWITCH** declaration may be referenced. The **SWITCH** declaration has the following form:

SWITCH name ← Label*1*, Label*2*, ..., Label*n*;

Example:

SWITCH BENNY ← ERRORLOOP, GOODRESULT, ALLTHRU;

In order to transfer control to one of these three points by means of the switch, the program must encounter a change-of-control statement such as:

GO TO BENNY [Y − 2]; next statement

The expression in brackets is evaluated. If the value is 1, control is transferred to ERRORLOOP; if the value is 2, control is transferred to GOODRESULT; and if 3, to ALLTHRU.

If the value of the expression is not a whole number, it will be rounded and then truncated to a whole number for purposes of selecting the transfer. If the value is either zero or outside the range of the number of labels given in the list, no transfer of control results and the program continues with the statement following the **GO TO** statement (indicated above by "next statement").

ARRAY DECLARATIONS

An array is a group or set of items arranged in such a manner that each item may be identified by its position within the group. A familiar array is a standard classroom seating plan with desks arranged in uniform rows and columns. Each desk in the room may be uniquely named in terms of its row and column. Thus, classroom A, row 5, column 3 would locate one particular desk in the designated room. Also, just as different students could occupy one particular desk from time to time, different values can be assigned to one position in an array.

In ALGOL an array declaration is used to define a fixed arrangement of items; it names the array, specifies its dimensions, and states the range within each dimension. The form of the array declaration follows.

ARRAY name [dimension*1*, dimension*2*, ..., dimension*n*]

Each dimension has the form:

lower limit: upper limit

Classroom A, with five rows and six columns, would be defined in ALGOL as follows:

ARRAY A [1:5, 1:6]

VARIABLES WITH SUBSCRIPTS

To name a single item within an array, the programer uses the identifier of the array, followed by the appropriate list of subscripts. The subscripts in a list are separated by commas, and the entire list is enclosed in brackets. Thus, to refer to the value of the variable in the fifth row and third column of array A, the programer would write:

A [5, 3]

The subscript list may contain arithmetic expressions, variables, and variables with subscripts. Some examples are:

```
V [J, K]
RATE [2 + X]
V [J, K [3, N]]
```

TYPES OF ARRAYS

In mathematical problems the items in an array normally are numerical values. Therefore, in ALGOL it is necessary to precede the **ARRAY** declaration with a type declaration—**REAL, INTEGER,** or **BOOLEAN.** If the type declaration is absent, type **REAL** is understood.

More than one array may be defined within one **ARRAY** declaration, and, if several have the same number of dimensions and the same ranges within each dimension, these need only be given once.

Every array defined within a single declaration must be of the same type. Example:

```
REAL ARRAY M, N, Q [1:10, 3:7],
S [1:5, 1:30, 2:19], T [1:4]
```

Five arrays of type **REAL** are defined by the above **ARRAY** declaration. Three of the arrays—M, N, and Q—have the same dimensions and ranges. The terms of the array dimension (lower limit: upper limit) may be arithmetic expressions involving identifiers if those identifiers have been declared and given a value in a block that contains the block in which the **ARRAY** declaration appears.

```
INTEGER ARRAY MAC [1:P + 2, K:L]
```

The identifiers P, K, and L must have values at the time the **ARRAY** declaration is encountered, since otherwise the declaration is meaningless.

The subscripts used with the **ARRAY** name, to indicate a single item within the set, follow the ALGOL conventions for subscripts and may be defined within the block in which they appear. Examples:

```
MAC [2, R]

COM3 [I, J]

FOR Q ← 2 × V WHILE V < 10 DO BEGIN
M [V] ← (Q + 5 × R) × Q;
V ← V + 1 END; next statement

FOR M ← 1 STEP 3 UNTIL 19, 20, 3 × N × A
WHILE A > 1 DO BEGIN
F [M] ← Z [M] + 5 × G; A ← A − 5;

    R ← A × F [M] END; next statement
```

SECTION 7
PROCEDURES

GENERAL NATURE OF PROCEDURES

A procedure is a section of coding which is to be executed at several points throughout the same program, or used without alteration in more than one program. It is because of this multiple usage that a section of coding is made into a procedure; as such, it can be incorporated into any program exactly as it was first written. Also, for multiple use in one program, it need be written only once, with each execution being called for by a simple notation. It is characteristic of a procedure that the operations to be executed are fixed, while the values of the variables, or the variables themselves, may be different when each point is reached from which the procedure is entered.

The section of coding to be used as a procedure is written once in a procedure declaration which has the format shown below. The parts of the procedure declaration must appear in the order indicated here.

> NOTE: Words and symbols enclosed by parentheses *in the examples* of this section represent quantities within actual parentheses, and are not merely remarks to the reader.

PROCEDURE HEADING

PROCEDURE name (list of formal parameters);

VALUE list of formal parameters to be replaced by the values of the actual parameters;

Specifications giving information about the formal parameters;

PROCEDURE BODY

BEGIN declarations for identifiers which have meaning only within this procedure;

statement; ... ; last statement **END**

The procedure heading begins with the reserved word **PROCEDURE** which indicates that what follows is a procedure declaration. Each procedure is given an identifier (name) by which it may be referenced. Formal parameters are names (identifiers) given to the variables of the procedure which obtain actual values when the procedure is used in a program. They represent quantities obtained from the main program which may be used in calculations of the procedure or which may be assigned new values through the execution of the procedure. These names are chosen when the procedure is first written and have no connection with a particular program. They take on actual meaning only when the procedure is called upon for execution by a program.

When a program makes use of a procedure (i.e., calls for its execution) the formal parameters are replaced by actual values or names actually being used in the main program.

The **VALUE** part is used if one or more formal parameters are to be replaced by an actual value before a procedure is executed. Since formal parameters which are to be replaced by different names are not contained in the **VALUE** list, a procedure declaration need not have a **VALUE** part.

Specifications for formal parameters are optional and their inclusion is suggested as an aid to persons using the procedure.

The procedure body may be a compound statement, a block, or even a single statement. All the rules pertaining to those segments of an ALGOL program apply. The following example will help clarify the concept of a procedure.

Assume that a procedure is required which will calculate the factorial of any whole number. The problem to be solved is: For any whole number N, compute $N! = 1 \times 2 \times 3 \times \ldots \times N$. For example, if $N = 6$, the problem becomes $6! = 1 \times 2 \times 3 \times 4 \times 5 \times 6$.

The following procedure accomplishes this:

```
PROCEDURE FACTORIAL (N, F) ;
VALUE N; INTEGER N;
BEGIN INTEGER I; F ← 1;
FOR I ← 2 STEP 1 UNTIL N DO
F ← I × F END
```

The effect of this procedure is that F is replaced by N!; it can be seen then that when this procedure is actually used, N represents some number which has been previously determined by some other portion of the program. The factorial of N is to be calculated in the procedure; therefore, N is listed after **VALUE.** On the other hand, F represents a variable which is, as a result of the procedure, to be assigned a new value; therefore, F is not listed after **VALUE.** It can be seen that the procedure body is a block, hence the variable I has meaning only within the procedure.

At this point it is well to remember that a **PROCEDURE** declaration is merely another declaration like type or **ARRAY** and, when incorporated in a program, appears in the head of a block.

When a program requires the fixed set of steps outlined in some **PROCEDURE** declaration, a procedure statement is written which supplies the values of the variable in the procedure, or assigns new variables to replace those named in the **PROCEDURE** declaration, and causes the procedure to be executed.

A procedure statement has the following format:

Procedure identifier (list of actual parameters)

The procedure identifier is the name assigned to the procedure in the **PROCEDURE** declaration. The list of actual parameters must contain all the identifiers, expressions, and constants which are to be substituted for the corresponding items in the list of formal parameters found in the **PROCEDURE** declaration. *The number of actual parameters must thus agree with the number of formal parameters.*

A procedure statement which would call for the execution of the above example (**PROCEDURE** FACTORIAL) is:

FACTORIAL (BOB, JOE)

When a procedure is called upon for execution, three operations take place, in effect:

(1) All formal parameters of the procedure which are listed after **VALUE** are replaced by the values of the corresponding actual parameters when the procedure statement is encountered in the program.

(2) The other formal parameters are replaced by the names of the corresponding actual parameters.

(3) The procedure body is then inserted into the program, taking the place of the procedure statement.

The example discussed above will be used to illustrate the process.

Given: A program which includes the **PROCEDURE** declaration FACTORIAL and the associated procedure statement, as well as other declarations and statements.

```
BEGIN   DECLARATION;
        DECLARATION;
        PROCEDURE FACTORIAL (F, N) ;
        VALUE N; INTEGER N;
        BEGIN INTEGER I; F ← 1;
        FOR I ← 2 STEP 1 UNTIL
        N DO F ← I × F END;
        DECLARATION;
        STATEMENT;
        STATEMENT;
        . . .
        STATEMENT;
        FACTORIAL (BOB, JOE);
        STATEMENT;
        . . .
        STATEMENT

END
```

Assume: JOE = 5 when the procedure statement is encountered.

Operations preparatory to execution: The first operation inserts the value of JOE, 5, in place of N in the procedure body. The latter then looks like this, in effect:

```
BEGIN INTEGER I; F ← 1;
FOR I ← 2 STEP 1 UNTIL 5 DO
F ← I × F END
```

The next operation inserts the identifier BOB in place of F throughout the procedure body:

```
        BEGIN INTEGER I; BOB ← 1;
        FOR I ← 2 STEP 1 UNTIL 5 DO
        BOB ← I × BOB END
```

Finally the altered procedure body replaces the procedure statement and is executed as if the following were the program:

```
BEGIN   DECLARATION;
        DECLARATION;
        PROCEDURE FACTORIAL (F, N);
        VALUE N; INTEGER N;
        BEGIN INTEGER I; F ← 1;
        FOR I ← 2 STEP 1 UNTIL N
        DO F ← I × F END;
        DECLARATION;
        STATEMENT;
        ...
        STATEMENT;
        BEGIN INTEGER I; BOB ← 1;
        FOR I ← 2 STEP 1 UNTIL 5
        DO BOB ← I × BOB END;
        STATEMENT;
        ...
        STATEMENT
END
```

Another example of a **PROCEDURE** declaration and possible procedure statements making use of it is given below.

PROCEDURE declaration example:

```
    PROCEDURE FALLINGBODY (T, V, S);
    VALUE T; REAL T, V, S;
    BEGIN REAL G; G ← 32.172;
    S ← G × T*2/2; V ← G × T END
```

The name of this procedure is FALLINGBODY. The variables T, V, and S are the formal parameters which correspond to identifiers appearing outside the procedure. The procedure body is a block. The variable G is declared within the body of the **PROCEDURE** declaration and therefore *has meaning only within the procedure.*

Procedure statement examples:

(1) FALLINGBODY (FINALTIME, SPEED, DISTANCE)
(2) FALLINGBODY (5.88, VEL2, DIST2)

Either of these two example procedure statements could be used to call out the procedure named FALLINGBODY. The first example causes the value of the variable FINALTIME to be used in place of T, and the results to be SPEED and DISTANCE instead of the variables (formal parameters) V and S. The second example causes the numerical value 5.88 to be used for T; in this case the actual parameter is the number itself since it is not an identifier. The results are to be VEL2 and DIST2 in place of the parameters V and S.

PROCEDURES AS FUNCTIONS

In SECTION 4 function designators were discussed, and a list was given of those which are considered standard in ALGOL It is possible for the ALGOL programer to create more functions. Two additional things are required of a **PROCEDURE** declaration in order for it to define the value of a function designator:

(1) An assignment statement must appear, in the procedure body, which has the procedure identifier to the left of the replacement operator.
(2) Since the **PROCEDURE** declaration is defining a single value, its type must be declared by preceding the word **PROCEDURE** by one of the three reserved words **(REAL, INTEGER, or BOOLEAN)** which indicate type.

Earlier in this section an example of a **PROCEDURE** declaration was shown which used the identifier FACTORIAL. It was constructed in the normal way. The following shows how this same **PROCEDURE** declaration might look if it were made into a function which (provided it has first been declared) can be used just as the standard functions can be used.

```
    INTEGER PROCEDURE FACTORIAL (N);
    VALUE N; INTEGER N;
    BEGIN INTEGER I, F; F ← 1;
    FOR I ← 2 STEP 1 UNTIL N DO
    F ← I × F; FACTORIAL ← F END
```

In order to make use of this procedure in a program, a notation called "function designator" is used. The function designator and the procedure statement have the same format:

Procedure identifier (list of actual parameters)

They differ in use only. The procedure statement stands alone, whereas the function designator is used as part of either an arithmetic or a logical expression. For instance, the following statements include function designators which refer to the above procedure.

Assignment statement:

```
BOB ← FACTORIAL (JOE)
```

Conditional statement:

```
IF P > FACTORIAL (Q) THEN
GO TO TOWN; next statement
```

SECTION 8
SAMPLE PROBLEMS

For computer solution of a problem, it is necessary to have data (parameters of the problem) entered into the computer by some data input process, and to have results given back by means of a data output process. Because the formal ALGOL 60 language does not deal with input or output, no formal input or output symbolism will be given here. Sentences in the example programs which follow indicate which items are to be read into the computer or written out by the computer, to illustrate the sequence of such processes in a complete computer program.

The formal definitions for data input and data output may be found in the manual describing the extensions to ALGOL which form a part of the programing language for the B 5000.

EXAMPLE 1

Given:

A curved line described by the equation:

(1) $Y = X^5 - \frac{X^3}{4} - 4X^2 + 21X - 5.238$

The equation for the slope (Y′) of the same curve:

(2) $Y' = 5X^4 - \frac{3}{4} X^2 - 8X + 21$

Find:

(a) The height of the curve above the X axis (value of Y) from equation (1) for values of X equal to 0, ½, 1, 1½, 2, 2½, . . . , 11½, 12.

(b) The values of Y′ for those same values of X.

ALGOL PROGRAM / EXPLANATORY REMARKS

BEGIN

The entire program is a block, made up of a type declaration and a **FOR** statement. The **FOR** statement includes an output statement.

REAL X, Y, YPRIME;

This type declaration indicates that the variables X, Y, and YPRIME may be assigned any value within the set of real numbers.

FOR X ← 0 **STEP** 1/2 **UNTIL** 12

The rest of the program (except for the final **END**) is a **FOR** statement. This is the **STEP-UNTIL** element with:

0 as the starting point,
1/2 as the increment, and
12 as the limit.

DO BEGIN	This begins the **DO** statement portion of the **FOR** statement. It is in itself a compound statement, made up of two assignment statements and an output statement.
Y ← X*5 − (X*3)/4 − 4 × X*2 + 21 × X − 5.238;	This is the first assignment statement, which assigns to Y the value of the arithmetic expression on the right.
YPRIME ← 5 × X*4 − (3 × X*2)/4 − 8 × X + 21;	This is the second assignment statement, which assigns to YPRIME the value of the arithmetic expression on the right.
{Print out the values of X, Y, and YPRIME}	This is where the output statement would be placed. It would result in printing three values each time the **DO** statement portion of the **FOR** statement is executed, or 25 times altogether.
END	End of the **DO** statement.
END	End of the block.

EXAMPLE 2

Given: (a) The series of numbers 17, 24, 31, 38, 45, 52, . . . , where each successive term is found by adding 7 to the previous term, and where 17 is defined as the first term;

(b) Equations

(1) $L = A + (N - 1)D,$

(2) $S = \frac{N}{2}(A + L)$

The following are the symbols and meanings:

A	is the first term of such an arithmetic series.
D	is the difference between two successive terms.
N	is the number of terms in the series up to the point in question.
L	is the Nth term of the series.
S	is the sum of the first N terms of the series.

Find: the value of the 50th term of the series by equation (1), and the sum of the first 50 terms by equation (2). Similarly, find the values of the 75th, 100th, 125th, 150th, etc., to the 300th term, and the sums of the terms to each of these points.

ALGOL PROGRAM	EXPLANATORY REMARKS
BEGIN	The entire program is a block made up of two type declarations, an output statement, and a **FOR** statement which includes another output statement.
REAL L, S;	This type declaration indicates that the variables L and S may be assigned any value within the set of real numbers.

INTEGER N;	This type declaration indicates that values assigned to the variable N must always be restricted to those numbers in the set of integers.
{Print column headings "N," "L," and "S"}	This represents an output statement which would result in printing the letters N, L, and S at the top of a page.
FOR N ← 50 **STEP** 25 **UNTIL** 300	The rest of the program (except for the final **END**) is a **FOR** statement. This is the **STEP-UNTIL** element, with: 50 as the starting point, 25 as the increment, and 300 as the limit.
DO BEGIN	The **DO** statement portion of the **FOR** statement starts here. It is in itself a compound statement, made up of two assignment statements and an output statement.
L ← 17 + (N − 1) × 7;	The first assignment statement.
S ← N/2 × (17 + L);	The second assignment statement.
{Print out the values of N, L, and S}	This represents an output statement which would print the values of N, L, and S under the appropriate column headings. Eleven such printouts would result.
END	End of **DO** statement.
END	End of block.

EXAMPLE 3

Given:

(1) $A = 2 \times W \times L + 2 \times W \times H + 2 \times L \times H$

(2) $R = \sqrt{\frac{A}{4\pi}}$

(3) $r = \frac{A}{4R\pi^2}$

Find:

(a) The total outside surface area of a rectangular box, where the width (W), length (L), and height (H) are 12 inches, 27 inches, and 14 inches, respectively. Use equation (1).

(b) The radius of a sphere which has the same surface area as the box. Use equation (2).

(c) The radius (r) of the cross section of a torus (doughnut) such that the torus will have the same surface area as the box and sphere, using the radius of the sphere (R) as the major radius (R) of the torus. Use equation (3).

ALGOL PROGRAM	EXPLANATORY REMARKS
BEGIN	The entire program is a block made up of two type declarations, six assignment statements, and an output statement.

```
REAL A, RS, RT;
INTEGER W, L, H;
  W ← 12;
  L ← 27;
  H ← 14;
  A ← 2 × W × L + 2 × W × H + 2 × L × H;
  RS ← SQRT (A/(4 × 3.14159) ) ;
  RT ← A/(4 × RS × 3.14159 * 2) ;
```

{Print out values of A, RS, and RT} — This represents the output statement. This program would result in one printout.

END — End of block.

EXAMPLE 4

Given: (1) Mean value = A = $(X_1 + X_2 + X_3 + \ldots + X_n)/n$

(2) Standard Deviation (Σ) = $\sqrt{\dfrac{X^2_1 + X^2_2 + \ldots + X^2_n}{n} - A^2}$

where $X_1, X_2, \ldots, X_n$ is a set of values, and n is the number of values.

Find:

(a) The solution of equation (1) for the mean value of the set.

(b) The solution of equation (2) for the standard deviation.

(c) The identification and value of the term which differs most from the mean value. (Identify by subscript number.)

(d) Which (if any) of the terms (values) are exactly equal to the mean value. (Identify by subscript numbers.)

Method:

1. Solve the first equation for the value of A.
2. Solve the second equation for the value of the standard deviation.
3. Find the maximum of the absolute values of the following expressions, and record the subscript number.

   $(X_1 - A), (X_2 - A), (X_3 - A), \ldots, (X_n - A)$.
4. Compare $X_1, X_2, \ldots, X_n$ successively with A, and record the subscript numbers of the terms which are equal to A.

The ALGOL statements shown below result.

(This section of coding computes the mean value.)

```
SUM ← 0
FOR I ← 1 STEP 1 UNTIL N DO SUM ← SUM + X [I] ;
A ← SUM/N;
```

(This section computes the standard deviation.)

```
SUMSQ ← 0;
FOR I ← 1 STEP 1 UNTIL N DO SUMSQ ← SUMSQ + X [I] * 2;
STANDEV ← SQRT (SUMSQ/N − A * 2) ;
```

(This section computes the maximum deviation.)

```
MAXDEV ← ABS (X [1] − A) ; INDEX ← 1;
FOR I ← 2 STEP 1 UNTIL N DO BEGIN
Z ← ABS (X [I] − A) ; IF Z > MAXDEV THEN BEGIN MAXDEV ← Z; INDEX ← I END END
```

(This section finds the subscript numbers of the terms equal to A.)

```
J ← 1;
FOR I ← 1 STEP 1 UNTIL N DO
IF X [I] = A THEN BEGIN Y [J] ← I; J ← J + 1 END
```

This program in ALGOL as presented is almost complete, but a few additional constructions are required.

A. Because every ALGOL program is considered a block in itself, it is necessary to place **BEGIN** and **END** around the program.

B. Because all identifiers of a program must be declared, the following declarations must be included, and placed at the beginning of the block:

```
INTEGER N, I, J, INDEX;
INTEGER ARRAY Y [1:1000];
REAL A, SUM, SUMSQ, STANDEV, MAXDEV, Z;
REAL ARRAY X [1:1000];
```

Note that these declarations of arrays with a subscript range of 1 to 1000 allow up to 1000 values of X to be used, and similarly up to 1000 values of Y (in case all values of X were equal and therefore all equal to A).

C. As pointed out earlier, data input and data output statements are also required to perform the following:

(1) Read data input values of $X_1, X_2, \ldots, X_n$, and n.

(2) Print an output page heading which reads: COMPUTATION OF MEAN VALUE, STANDARD DEVIATION, AND GREATEST DEVIATION.

(3) Print the values found for A, STANDEV, MAXDEV, and INDEX.

(4) Print a heading which reads:

SEQUENCE NUMBERS OF THE TERMS EQUAL TO THE MEAN VALUE.

(5) Print out the (J − 1) values in the set Y [J] (these being the sequence numbers of (4) above).

(6) If there are no terms equal to the mean value, then, instead of (4) and (5) above, print out a line which reads:

NO VALUES OF X [I] ARE EQUAL TO A.

ALGOL PROGRAM / EXPLANATORY REMARKS

```
BEGIN
```

The entire program is a block made up of two type declarations, two array declarations, an input statement, seven assignment statements, four **FOR** statements, an **IF** statement, and two output statements.

```
  REAL A, SUM, SUMSQ, STANDEV, MAXDEV, Z;
  INTEGER N, I, J, INDEX;
  INTEGER ARRAY Y [1:1000];
  REAL ARRAY X [1:1000];
```

```
{Read data input values of X1, X2, X3, . . . , Xn, and n} ;   This represents the input statement which
                                                              would cause n + 1 values to be read.

SUM ← 0;

FOR I ← 1 STEP 1 UNTIL N DO SUM ← SUM + X [I] ;               The first FOR statement.

A ← SUM/N;
SUMSQ ← 0;
FOR I ← 1 STEP 1 UNTIL N DO                                   The second FOR statement.
  SUMSQ ← SUMSQ + X [I]*2;

STANDEV ← SQRT (SUMSQ/N − A*2) ;

MAXDEV ← ABS (X [1] − A) ;

INDEX ← 1;

FOR I ← 2 STEP 1 UNTIL N                                      The third FOR statement starts here.

  DO BEGIN Z ← ABS (X [I] − A) ;                              This is the DO statement portion, which is
                                                              itself a compound statement made up of an
                                                              assignment statement and a conditional state-
                                                              ment.

    IF Z > MAXDEV THEN                                        This is the conditional statement, of which
                                                              another compound statement is a part.

      BEGIN MAXDEV ← Z;
      INDEX ← I END

END;                                                          This terminates the third FOR statement.

(Print heading which reads: COMPUTATION OF )                  The first output statement, which results in
{MEAN VALUE, STANDARD DEVIATION, AND       }                  printing a heading.
(GREATEST  DEVIATION                       )

{Print values of A, STANDEV, MAXDEV, and INDEX}               The second output statement, which results in
                                                              printing four values.

J ← 1;

FOR I ← 1 STEP 1 UNTIL N                                      The fourth FOR statement.

  DO IF X [I] = A THEN                                        The DO statement portion.

    BEGIN Y [J] ← I ; J ← J + 1 END;

IF J = 1 THEN   (Print out a line which reads: NO )           This is the conditional statement, and the out-
                {VALUES OF X [I] ARE EQUAL        }           put statement which will be executed if J = 1.
                (TO A                             )

  ELSE  BEGIN                                                 Compound statement (made up of two output
                                                              statements), which will be executed if J ≠ 1.

{Print a heading line which reads: SEQUENCE NUM-}
{BERS OF TERMS EQUAL TO THE MEAN VALUE          }

{Print out the sequence numbers, which are}
{the (J − 1) terms of the Y array.        }

  END                                                         End of conditional statement.

END                                                           End of block.
```

APPENDIX A

BURROUGHS B 5000 ALGOL 60 SYMBOLS

		BURROUGHS B 5000 ALGOL 60 HARDWARE REPRESENTATION	FORMAL ALGOL 60 REFERENCE LANGUAGE SYMBOLS
	Blank		
	Decimal Point	.	.
	Comma	,	,
	Colon	:	:
	Semicolon	;	;
	Left Parentheses	(	(
	Right Parentheses	)	)
	Left Bracket	[	[
	Right Bracket	]	]
	Replacement Operator	←	:=
Relational Operators	Less	<	<
	Less or Equal	≤	≦
	Equal	=	=
	Greater or Equal	≥	≧
	Greater	>	>
	Not Equal	≠	≠
Logical Operators	And	AND	∧
	Or	OR	∨
	Not	NOT	¬
	Equivalent	EQV	≡
	Implies	IMP	⊃
Arithmetic Operators	Add	+	+
	Subtract	−	−
	Multiply	×	×
	Divide	/	/
	Integer Divide	DIV	÷
	Exponentiate	*	↑
Alphabetic Characters	A through Z		A, a through Z, z
Numeric Characters	0 through 9		0 through 9

APPENDIX B

DEFINITIONS OF EXPONENTIATION OPERATIONS

These definitions of results are for the operation A^n, where A is to be raised to the nth power. A represents a number of either type **REAL** or type **INTEGER**.

(I) Where n is a number of type **INTEGER**:

(1) If n is greater than zero, and A is not equal to zero, then $A^n = A \times A \times A \times \ldots \times A$ (n times). The result is of type **REAL** if A is type **REAL**, or type **INTEGER** if A is type **INTEGER**.

(2) If n is greater than zero, and A equals zero, the result is zero.

(3) If n equals zero, and A is not equal to zero, then A^n equals 1, and the type of the answer is the same as the type of A.

(4) If n equals zero, and A also equals zero, then the result is undefined.

(5) If n is less than zero, and A equals zero, then the result is undefined.

(6) If n is less than zero, and A is not equal to zero, then $A^n = 1/(A \times A \times A \times \ldots \times A)$ where the denominator has n factors, and the result is always of type **REAL**.

(II) Where n is a number of type **REAL**:

(1) If A is greater than zero, then $A^n = (e)^{n \times \ln(A)}$, and the result is always of type **REAL.**

(2) If A equals zero and n is greater than zero, the result is zero.

(3) If A equals zero, and n is equal to zero or less than zero, the result is undefined.

(4) If A is less than zero, the result is undefined.

APPENDIX C

GLOSSARY

Algorithm:	A statement of the steps to be followed in the solution of a problem.
Argument:	One of the independent variables upon whose value that of the function depends.
ARRAY:	An ordered arrangement of items, such as a determinant, matrix, or vector.
Function:	(1) An element whose value is so related to some other element (or elements) that it is dependent upon those secondary elements (arguments).
	(2) The relationship which defines the value of a dependent variable based on the value(s) of the independent variable(s) (arguments).
Instruction:	In conventional computer usage, an instruction is a symbol (or group of symbols) recognized by the computer as an order to perform an operation. Typical instructions include: add, multiply, read a punched card, write on magnetic tape, store in memory, and change control to another part of the program if a certain condition (such as a minus sign) exists. Under older methods, a programer had to learn from 30 to 150 different instruction symbols, and all their individual variants and peculiarities.
INTEGER:	An ALGOL reserved word indicating that the numbers or variables so designated may have only numerical values which are whole numbers or zero.
Integer:	A whole number, either positive or negative, containing no fractional or decimal part. Zero may be an integer.
Integral:	An adjective indicating the integer or whole-number portion of the modified term.
Inverse:	The inverse of N is 1/N; the inverse of B/A is A/B. "Reciprocal" is used with the same meaning.
Machine Language:	The system of codes by which instructions and data are represented internally within a particular data processing system.
Operand:	Any of the quantities entering into an operation.
Operator:	A symbol which specifies that a defined action is to take place.
Power:	Raising a number to the power of n means multiplying the number by itself n times; thus, in ALGOL notation, 10*3 means $10 \times 10 \times 10$, or 1000. This process is also called exponentiation; in the above example, 3 is the *exponent* of 10.
Problem language:	The words and symbols used in the original formal statement of a problem, as for hand computation.
Program:	An ordered list of instructions which directs the computer to perform certain operations in a specified sequence to solve a problem.

Programer:	One who designs a set of computer operations to solve a problem (see Program).
Pseudo-Language:	An arbitrary system of codes, independent of the hardware of a computer, which is used to express the steps in a computer program. It usually will be converted (translated) into an equivalent set of machine language codes which are actually used to perform the program on a computer.
REAL:	An ALGOL reserved word indicating that the numbers or variables so designated may take on any real-number value.
Real Numbers:	The set of *all* positive and negative numbers, including integers and zero, but excluding any imaginary or complex numbers.
Reciprocal:	See Inverse.
Scale Factor:	A factor by which a quantity in a problem is multiplied or divided to determine the location of the decimal point.
Subscript:	In algebra, a subscript is a number or letter, appearing below the center of a line of other symbols, which represents the position of an element in an array. For example: A_5 is the fifth element in an array called A. In ALGOL, subscripts are enclosed in brackets; the above example would be written A[5] in ALGOL. In a two-dimensional array (i.e., one with rows and columns), two subscripts are used to select the row and column. For example: $B_{5,3}$ names the element in the fifth row, third column, of array B; in ALGOL we would write B[5,3].

APPENDIX D

RESERVED WORD LIST

The following reserved words, which form a major portion of the B 5000 ALGOL vocabulary, may be used *only as shown in this manual.*

ABS
ARCTAN
ARRAY
BEGIN
BOOLEAN
COS
DIV
DO
ELSE
END
ENTIER
EQV
EXP
FALSE
FOR
GO TO
IF
IMP
INTEGER
LN
LOG
OWN
PROCEDURE
REAL
SIGN
SIN
SQRT
STEP
SWITCH
THEN
TRUE
UNTIL
VALUE
WHILE

Burroughs Corporation

Detroit 32, Michigan

In Canada: Burroughs Business Machines Ltd., Toronto, Ontario