CHAPTER 2

Notes on ALGOL at Carnegie Tech

INTRODUCTION

ALGOL-60 has been designed to be both a universal language for describing and publishing numerical algorithms, and a programming language for executing algorithms on computing machines. The "reference language" ALGOL-60 has been precisely and elegantly defined in the Revised ALGOL-60 Report (Communications of the ACM, 6, 1 (Jan. 1963)). When ALGOL is actually implemented on a particular computer, however, some changes of notation and some restrictions are usually added to this definition.

The ALGOL translator which has been written at Carnegie Tech for the CDC G-20 computer accepts a language which we call ALGOL-20 to distinguish it from ALGOL-60 when we need to be purists. As a matter of fact, most of the differences between ALGOL-20 and the reference language ALGOL-60 are minor; however, a knowledge of them is needed to use the CIT ALGOL system successfully. In this document, a reference to simply "ALGOL" will always mean ALGOL-60, the reference language.

This chapter describes those aspects of ALGOL-20 which differ from ALGOL-60. As such, it is the primary documentation of our ALGOL system. It is keyed to both the Revised ALGOL-60 report and to the text, A Guide to Algol, by D. D. McCracken. References to the former are by section numbers given in square brackets, and to the latter by section numbers given in round brackets. Thus the paragraph at the top of the next page relates to section 2.3 of the Revised Algol Report and to section 1.4 of McCracken.

SYMBOLS (1.4) [2.3]

The G-20 accepts all of the special symbols of ALGOL-60 except for those shown in the following table:

ALGOL-60	ALGOL-20
⊃ ("implies")	Not available, but "¬" may be used with "∨" to obtain the same effect. See page 2.8.
≡ ("is equivalent")	Use "=". See page 2.8.
× (multiplication)	Use "*".
÷	Not available, but ":" may be used with "/" with the same effect. See page 2.7.
≯	Use "¬>".
≮	Use "¬<".
' ' (string quotes)	Both represented by '. See page 2.3.

In four cases, ALGOL-20 uses a pair of adjacent symbols to stand for a single symbol of ALGOL-60. For example, the ALGOL-60 assignment operator ":=" is strictly a single symbol, but it must be punched into an ALGOL-20 program card as a colon and an equal sign in adjacent columns. There must be no blanks separating the symbols of the pair, and they must both be on the same card. Note: punching them into the same column will give a "hash" of holes which the G-20 will interpret as some other (erroneous) character. The four double-symbol characters are

ALGOL-60 character	ALGOL-20 character pair
:=	:=
≯	¬>
≮	¬<
≠	¬= ("≠" is also allowed)
non-existant	8R
non-existant	8L
non-existant	8F

NUMBERS (2.1) [2.5]

(a) A number, N, in an ALGOL-20 program must either be zero (which may be punched with or without a decimal point) or else its absolute value N must satisfy:

$$1.275_{10}-57 \leqq N \leqq 3.450_{10}+69$$

(b) Because of the nature of the G-20 computer, the distinction between real and integer numbers is unimportant. The programmer may write an integer-valued constant with or without a decimal point (e.g., "34", "34.", or "34.0") without changing the type of arithmetic performed with the constant.

Numbers are represented in the G-20 in "floating point" form with a maximum of 42 binary digits of mantissa, corresponding to approximately 12 decimal digits of precision. If more than 12 digits are written, the extra (least significant) digits will be ignored. (The number is rounded at the 14th octal digit.)

(c) In ALGOL-20, the last character of a real number may be a decimal point; thus, the number "6." is legal.

(d) Octal numbers may be written in ALGOL-20. See Chapter 6e.

STRINGS [2.6]

(a) A string cannot contain a string since ALGOL-20 has no way of distinguishing between the left and right string quotes.

(b) Strings of four characters or less may be used as logic constants and assigned to logic variables. If more than four letters appear in such a string, only the leftmost four are used. Strings of less than four characters are stored right-justified.

IDENTIFIERS AND VARIABLES (2.2) [2.3, 3.1]

(a) Only upper case (capital) letters are available in ALGOL-20.

(b) In ALGOL-20, certain identifiers have special meanings and are therefore reserved. The programmer may never use these reserved ALGOL identifiers as variables or, indeed, for any purpose other than their reserved meanings. These reserved identifiers must be separated from adjacent identifiers

by at least one blank. For example, if the blank between the reserved identifier IF and the identifier "X" were omitted in "IF X > 0", then the ALGOL translator would interpret "IFX" as a single variable identifier; as a result, the statement would have no meaning at all.

The reserved identifiers in ALGOL-20 are

ABS	GO TO	PRINT
ARCTAN	HALF	PROCEDURE
ARRAY	IF	PUNCH
BEGIN	INDEX	READ
BOOLEAN	INPUT	REAL
COMMENT	INTEGER	SIGN
COS	LABEL	SIN
DO	LIBRARY	SQRT
ELSE	LN	STEP
END	LOGIC	STRING
ENTIER	MAX	SWITCH
EXP	MIN	THEN
FALSE	MOD	TRUE
FOR	MONITOR	UNTIL
FORWARD	NAME	VALUE
GO	OUTPUT	WHILE
GOTO	OWN	

Some of these reserved identifiers have no ALGOL-60 equivalent; in particular:

HALF, INDEX, LOGIC (see page 2.5 below)

MAX, MIN, MOD (see page 2.9 below)

NAME, INPUT, OUTPUT, PRINT, PUNCH, READ
(see Chapter 3 - Input/Output)

LIBRARY (see Chapter 5)

FORWARD

MONITOR

FORWARD and MONITOR have not yet been implemented, but will be described when they are available.

All of the ALGOL-60 standard functions are available in ALGOL-20, and their names are reserved identifiers:

ABS	ENTIER	SIGN	(2.4)
ARCTAN	EXP	SIN	[3.1.4]
COS	LN	SQRT	

See Chapter 5 for further information on these functions.

"TO" is reserved only when it follows immediately after the reserved identifier GO. In any other context, "TO" may be used as an ordinary

identifier by the programmer. See page 2.10 of these notes.

In addition to the reserved words listed above, ALGOL-20 includes a set of "privileged" identifiers which have built-in meanings without being declared; they are, in effect, declared by the translator in a block head outside of the outer-most block of the program. Therefore, if the programmer does not wish to use one of these identifiers in its privileged meaning, he may simply ignore the fact that it is privileged and declare and use it as he would any non-special identifier. Further, if a privileged identifier is declared in an inner block, it resumes its privileged meaning as soon as the end of the inner block is passed. These identifiers are listed and their meanings are explained in Chapter 6d. Identifiers may be added to this list by the Computation Center at some future time. Since they are not reserved, additional privileged identifiers cannot accidentally interfere with identifiers written into a current ALGOL program.

(c) Spaces may not appear within an identifier in ALGOL-20. The programmer may, however, freely sprinkle periods (.) within identifiers to separate them into words and improve the readability of the program. These periods are ignored by the ALGOL-20 translator; therefore, the following are all instances of the same identifier::

```
READACARD
READ.A.CARD
R.E.A.D.A.CARD..
```

(d) ALGOL-20 allows both simple and subscripted variables of type half, and logic, as well as real, integer, and Boolean. Also, simple variables may be of type index.

Real variables are stored in the G-20 with a precision of 42 binary digits, requiring two successive memory cells per variable. Half variables are stored with a precision of only 21 binary digits (about 6 significant decimal digits) and occupy only a single location, but otherwise act as real variables. Therefore, the programmer may use half variables to gain memory space at the expense of precision.

Logic variables are unsigned 32 bit G-20 logic words, which may be used for bit and character manipulation processes. They may be used in either arithmetic or Boolean expressions. Simple variables of type index will be assigned to G-20 index registers but act otherwise as variables of type

integer. The uses of logic and index variables are complex to explain but obvious to those ALGOL programmers who are also knowledgeable in G-20 machine language For more information see Chapter 6e.

(e) The value of a real or half variable must either be zero or else lie within the range given below:

$$\text{real:} \quad 1.275_{10}\text{-}57 \leq \text{abs(R)} \leq 3.450_{10}\text{+}69$$

$$\text{half:} \quad 1.275_{10}\text{-}57 \leq \text{abs(H)} \leq 1.645_{10}\text{+}63$$

integer and index variables will always take on integer values in the range

$$-2097152 < I < 2097152 \quad (= 2^{21})$$

logic variables are always positive. If used as strings, they are four or less characters in length, and if used as numeric quantities they are restricted to

$$0 \leq L < 4294\ 967296 \quad (= 2^{32})$$

The values of Boolean variables must be either true or false.

The G-20 replaces by zero any non-zero arithmetic result which is smaller than 1.175_{10}-57 in magnitude; this situation is called an underflow. An intermediate arithmetic result which is greater than 3.450_{10}+69, the largest number representable in the G-20, is called an overflow. An overflow during execution of the object program will cause the run-time error message "RUN ERROR-EXPO" to be printed, and terminate execution of the program (unless error recovery is in use). See Chapter 6b for further details on run-time errors.

An exponent overflow cannot occur during translation of the ALGOL source program; violation of the restrictions on ALGOL-20 numbers given above will cause a normal syntactic error message which will not, however, terminate translation.

The number 3.450_{10}+69 is the upper limit for the result of each individual arithmetic operation in the evaluation of any arithmetic expression, regardless of the types of the variables in the expression. However, if the result of the expression is assigned to a half variable, then a value greater than 1.645_{10}+63 will result in an exponent overflow message as explained above. A value assigned to an integer variable, on the other hand, will be truncated modulo $2^{21} = 2097152$; while a value assigned to a logic variable will be truncated modulo 2^{32} (and given a positive sign); in either case, no overflow message will occur.

ARITHMETIC EXPRESSIONS (2.3) [3.3]

(a) In ALGOL-20, the asterisk ("*") is used in place of the multiplication sign ("×") of ALGOL-60.

(b) ALGOL-20 arithmetic expressions may contain the truncation operator "↓" defined mathematically by

$$\downarrow X = \text{sign}(X) * \text{entier}(\text{abs}(X))$$

That is, ↓X is simply the integer part of X if $X \geq 0$, and is -(integer part (-X)) for $X < 0$. Thus, ↓(1.7) = 1, ↓(-1.7) = -1. Truncation is performed modulo 2^{32} = 4294967296; for example, ↓4294967298 = 2.

The truncation operator is unary, having exactly one operand which is the complete expression immediately to the right of the "↓" symbol. The precedence of "↓" is very high, so that "↓" will be executed before "↑" or any other arithmetic operation (unless parentheses are used to force a difference order). For example, "↓X/Y" means (↓X)/Y and "X↑↓Y" means X↑(↓Y). (Truncation is done by an add-logical in mode zero of zero.)

(c) The truncation operator, "↓", can be used to get the effect of the integer divide operation, "÷", which is not available in ALGOL-20. If I and J are integer variables, then

$$I \div J = \downarrow(I/J)$$

Notice that the "↓" operator can operate on any integer or real expression, and is therefore more general than "÷".

(d) When a variable of type half appears in an arithmetic expression, the rules for determining the type of the result are exactly as if the half variable had been of type real. In fact, full precision (42 bit) floating point arithmetic is always performed on all variables other than Boolean and logic in the G-20.

(e) The "+" and "-" can be used either as binary operators or else as unary operators. When "+" and "-" are used as unary operators with "↑" in the combination "↑+" or "↑-", parentheses around the exponent may be omitted.

The following table shows some examples of this rule:

The ALGOL-20 Expressions	Means: (both in ALGOL-20 and ALGOL-60)
X↑+Y	X↑(+Y)
X↑-Y	X↑(-Y)
X↑↓-Y	X↑(↓(-Y))

(f) The precedence of operators and relations in ALGOL-20 is

↓ (done first)
mod
↑
- + (used as unary operators)
/ *
- + (used as binary operators)
≠ = ¬> ¬< > <
¬
∧
∨ (done last)

That is, unless parentheses force a different order, ↓ will be performed, then mod, then ↑, and so on. The unary operators + and - are special cases. Unary + is ignored. Unary - is performed on the expression on its right whose operators have higher precedence than it. For example,

q mod - a ↑ b mod c * d

is

q mod((-(a ↑ (b mod c))) * d)

BOOLEAN EXPRESSIONS (3.6) [3.4]

(a) The Boolean operator "⊃" ("Implies") is not available in ALGOL-20. However, for any Boolean expressions B_1 and B_2, the ALGOL-60 expression $B_1 \supset B_2$ may be replaced by either of the equivalent forms:

$$\neg B_1 \vee B_2$$

$$\neg(B_1 \wedge \neg B_2)$$

(b) ALGOL-20 substitutes the equality symbol "=" for the Boolean equivalence operator "≡". Note that the ALGOL 60 report gives ≡ very

low precedence. ALGOL-20 cannot distinguish between $\equiv$ and $=$ and thus gives them the same precedence. Thus $A \wedge B = C \vee D$ is taken as $A \wedge (B = C) \vee D$, and parentheses must be used if any other meaning is intended.

STANDARD FUNCTIONS (2.4) [3.2.4]

ALGOL-20 has three built-in operators, MAX, MIN and MOD, which are not in ALGOL. These are defined mathematically as follows, where E_1, E_2,...E_n are arithmetic expressions.

MAX (E_1, E_2, ... E_N) = the largest algebraic value of the N expressions;

MIN (E_1, E_2, ... E_N) = the smallest algebraic value of the N expressions;

E_1 MOD E_2 = $E_1 - E_2 * \downarrow (E_1/E_2)$

MAX and MIN may have any number of expressions as arguments. Note that MOD is written as an operator between its two arguments. The above definition for MOD holds for all values of E_1 and E_2, but in the case where both arguments are positive integer-valued expressions, then E_1 MOD E_2 is the remainder for E_1 divided by E_2 (and $\downarrow(E_1/E_2)$ is the integer quotient). Although E_1 and E_2 each appear twice in the definition, they are actually evaluated only once.

ASSIGNMENT STATEMENTS (2.5) [4.2]

(a) In addition to the ":=" operator of the reference language, ALGOL-20 allows the left arrow ("←") as an assignment operator. The left arrow has the same meaning as ":=", except when a non-integer expression is assigned to an integer variable. The assignment statement

<integer variable> ← <non-integer expression>

result in truncating the value of the expression to an integer without rounding. If ":=" is used instead, the value will be rounded to an integer in conformity with the reference language; however, the "←" operator produces more efficient object code.

(b) In a multiple assignment statement, the "left part" variables need not all be of the same type. For example, the sequence

```
REAL X ; INTEGER I, J ;
I ← X ← J := 3.7*X;
```

is allowed in ALGOL-20. The rule given in (a) above determines for each integer left part variable whether or not rounding will occur.

LABELS AND GO TO STATEMENTS (3.2) [4.3]

(a) Only identifiers may be used as labels in ALGOL-20; integer labels are not permitted.

(b) In ALGOL-20, GOTO and GO are both reserved identifiers, and TO is ignored when it follows after GO. Hence

```
GO TO Label
GOTO Label
GO Label
```

are all equivalent and permissible.

CONDITIONAL STATEMENTS (3.3) [4.5]

(a) Because of character set restrictions, ALGOL-20 must make the following substitutions for relational operators:

ALGOL-60	ALGOL-20
≧	¬<
≦	¬>

In addition, both "≠" and "¬=" are allowed in ALGOL-20.

(b) There are some complex syntactic construction which were allowed by the original ALGOL-60 report but were subsequently found to be ambiguous or controversial. One such ambiguity arises when a for statement comes within the scope of an if clause.

(1) Consider the following construction:

```
if ... then
  for . . do
    begin
      if ... then <unconditional statement>
             else <statement>
    end;
```

If the "begin ... end" pair is omitted, this construction becomes ambiguous since the phrase "else <statement>" could belong to either the inner or the outer if clause. ALGOL-20, in agreement with the 1962 revision of ALGOL-60, allows the "begin ... end" pair to be omitted, and considers "else <statement>" to belong with the second <if clause>; i.e., the construction is treated as if the "begin ... end" pair were actually present.

(2) The following construction:

```
if ... then
  for ... do <unconditional statement>
    else <statement>
```

is not actually ambiguous. However, the revision of ALGOL-60 syntax which took care of case (1) also had the undesirable effect of outlawing construction (2) which is perfectly respectable. Therefore, ALGOL-20 will allow (2) but will print a "Note 7" (see Chapter 6b) to point out that it is inconsistent with revised ALGOL-60 syntax.

CONDITIONAL EXPRESSIONS (3.5)

(a) ALGOL-20 allows certain constructions with conditional expressions which are unambiguous but illegal in revised ALGOL-60. The ALGOL-20 translator will flag any of these constructions with a "Note 4" message (see Chapter 6b) to call the programmer's attention to the violation of ALGOL syntax.

In ALGOL-20 the right-hand operand of a binary operator may be a conditional expression without parentheses; e.g., the second set of parentheses may be omitted in:

```
(if X > 0 then X else Y) + (if Y > 0 then 3 else X)
```

Note, however, that omission of the first set of parentheses, surrounding the conditional expression which is the left-hand operand of the binary operator "+", would change the meaning to the following:

```
if X > 0 then X else (Y + if Y > 0 then 3 else X).
```

Similarly, the following construction is legal in ALGOL-20:

```
X * if A > B then 3 else Y + Z
```

but will cause a "Note 4". It will be interpreted as:

X * (if A > B then 3 else (Y + Z)) .

ALGOL-20 allows the analogous constructions with binary Boolean operators and conditional Boolean expressions, and with relational operators and conditional arithmetic expressions. An example of the last is the Boolean expression

(if BOOL then X else Y) < if BOOL then 3 else Z

The expression with the first set of parentheses omitted would be interpreted as

if BOOL then X else (Y < (if BOOL then 3 else Z))

FOR STATEMENTS (4.1) [4.6]

(a) A left arrow may be used instead of ":=" in an ALGOL-20 for clause; "←" will truncate and ":=" will round each implicit assignment to a for variable of type integer.

(b) The value of the controlled variable is not undefined upon normal exit from an ALGOL-20 for statement. The value of the for variable upon exit depends upon the form of the last element in the for list, and is in general just what would be obtained if the equivalent basic programs (see section 4.1 of McCracken or section 4.6.4 of the report) were substituted for the for statement. Thus, upon exit from an until or while form of for list element, the for variable has the first value for which the final test failed. For example:

FOR I ← 1 STEP 1 UNTIL 10 DO S ;

leaves I = 11 when the for list is exhausted and control passes to the next statement.

(c) A fourth form of for list element is permitted in ALGOL-20:

FOR V ← E_1 STEP E_2 WHILE B DO S;

where E_1 and E_2 are arithmetic expressions, B is a Boolean expression, and

S is any statement. This is equivalent to the simple program:

```
        V ← E1 ;
LOOP:   IF  B  THEN
          BEGIN
            S   ;
            V ← V + E2 ;   GO TO LOOP
          END ;
```

Notice that if the Boolean expression B is: $(V - E_3) * (E_2) \leq 0$ then the new step ... while form of for list element is identical to the step ... until form. However, when (as is usual) the sign of the step expression E_2 is known to the programmer, the step ... while form (omitting the multiplication by E_2) will be more efficient in both space and time.

ARRAYS (5.2, 5.3) [5.2, 3.1.4]

(a) ALGOL-20 arrays may be of type integer, real, Boolean, half, or logic. Index arrays are not permitted.

(b) A non-integer value of a subscript expression in ALGOL-20 is not rounded, only truncated. This may lead to hard-to-detect errors. For example, suppose that the result computed for a subscript expression is 3.9999... instead of 4 because of round-off error; this value will be truncated to 3, referring to the wrong element of the array. Thus, the plausible program:

```
FOR  X ← 0  STEP  0.2  UNTIL  1.0  DO
     A [5 * X] ← X ;
```

may not work correctly because of the round-off error in 0.2 which cannot be exactly represented in a binary computer like the G-20. The following alternative will work:

```
FOR  I ← 0  STEP  1  UNTIL  5  DO
     A [I] ← I/5 ;
```

(c) The speed of execution of an ALGOL-20 program does not depend upon the lower or upper bounds of an array subscript, upon the order of the dimensions, or upon the types of variables appearing in subscript

expressions, however, the number of memory cells required by an array does depend upon the order of the dimensions; the least number of cells is required if the longer dimension is listed last.

OWN VARIABLES (6.6) [5.0]

Own arrays may be used in ALGOL-20, but they must have fixed subscript bounds so that storage may be allocated to them before execution begins; that is, "dynamic own arrays" are not allowed.

Own simple variables and own arrays are initialized to zero (or false, in the case of Boolean quantities or ' ' in the case of logic quantities) before execution begins.

PROCEDURES

(a) Parameters (7.4) [4.7]

When the first occurrence of a label in a block is as an actual parameter in a procedure call, then the ALGOL-20 processor must be forewarned that this identifier is a label. This requires that the label identifier appear in a label declaration in the block head. For example:

```
BEGIN
   INTEGER I, J; LABEL L;
   PROC (X, L) ;
L : I ← I + 1 ;
END ;
```

This is the only circumstance in which a label declaration is required in ALGOL-20.

(b) Specifications (7.5) [5.4.5]

All formal parameters in an ALGOL-20 procedure declaration must appear in the specification part of the procedure heading.

(c) Recursive Procedures (7.7)

Recursive procedures are not now available in ALGOL-20.

(d) Arrays, switches, and labels cannot be called by value.