CS86

$2.25

ALGOL W

COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY
JANUARY 1968

ALGOL W NOTES

FOR INTRODUCTORY

COMPUTER SCIENCE COURSES

by

Henry R. Bauer
Sheldon Becker
Susan L. Graham

COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY
JANUARY 1968

Introduction

The textbook Introduction to ALGOL by Baumann, Feliciano, Bauer, and Samelson describes the internationally recognized language ALGOL 60 for algorithm communication. ALGOL W can be viewed as an extension of ALGOL.

Part I of these notes describes the differences between similar constructs of the two languages.

, For clarity, Part I is numbered according to the sections of the textbook. In general only differences are mentioned; items which are the same in both languages are usually not discussed.

Part II presents some of the details concerning the new features of ALGOL W. A complete syntactic and semantic description of these constructs as well as of all others in the language is available in "ALGOL W Language Description".

1. Basic Symbols of the Language

1.1. The basic symbols

1.1.1. Letters

Only upper case letters are used.

1.1.3. Other symbols

The following are the same in ALGOL 60 and ALGOL W.

+ - / . ,

:= ;

( ) ¬

= < >

The following are different in the two languages. The correspondence between the symbols is shown in the following table:

ALGOL 60	ALGOL W
10	'
X	*
t	**
[	(
]	)
‘ ’	"
÷	DIV
≡	=
⊃	no equivalent
∨	OR

ALGOL 60	ALGOL W
$\wedge$	AND
␣	one blank space
$\neq$	¬ =
<	< =
>	> =
:	: or :: (cf. section 6.1 and 4.2.1)
no equivalent	I #

All characters indicated for ALGOL W are on the IBM 029 key-punch.

The significance of spaces in ALGOL W will be discussed in subsequent sections.

1.2. Numbers

A number is represented in its most general form with a scale factor to the base 10 as in conventional scientific notation.

EXAMPLE 3.164981'-4 means 3.164981×10^{-4}

This is often called the floating point form. Certain abbreviations omitting unessential parts are permissible.

EXAMPLES	77	317.092	126'04
	551	.5384	04.719'2
	'30	0.710	9.123'+1
	'-7	0	2'-6
	'-3	009.123'+01	2.0'-06

To represent a long floating point (cf. Section 2.3.1) number an

L must be added as part of the number specified,

EXAMPLES 77L 317.092L 126'04L

In ALGOL W, complex numbers (short and long forms) may be used, The imaginary part of a complex number is written as an unsigned real number followed by an I.

EXAMPLES 4I 4.8I 4'-5I

Long imaginary numbers are followed by an L.

EXAMPLE 4.8IL

Numbers may be written in a variety of' equivalent forms.

EXAMPLE 12'04 ≡ .12'6 ≡ 1.2'05 ≡ 120000.0

No spaces may appear within an unsigned number, The magnitude of an integer or the integer part before the decimal point in a floating point number must be less than or equal to 2147483647. The magnitude of a non-zero floating point number must be between approximately 5.4×10^{-79} and 7×10^{75} ($1/16 \times 16^{-64}$ and $(1-16^{-6}) \times 16^{63}$).

1.3. Identifiers

A letter followed by a sequence of letters and/or digits constitutes an identifier, Identifiers may be as short as one letter or as long as 256 letters and digits,

Identifiers may be chosen freely and have no inherent meaning. However, ALGOL W recognizes a set of' reserved words which must not 'be used as identifiers,,

RESERVED WORDS

ABS	GOTO	REM
AND	GO TO	RESULT
ARRAY	IF	SHL
BEGIN	INTEGER	SHORT
BITS	IS	SHR
CASE	LOGICAL	STEP
COMMENT	LONG	STRING
COMPLEX	NULL	THEN
DIV	OF	TRUE
DO	OR	UNTIL
ELSE	PROCEDURE	VALUE
END	REAL	WHILE
FALSE	RECORD	
FOR	REFERENCE	

Spaces are used to separate reserved words and identifiers from each other and from numbers.

Certain identifiers are predefined for use by the programmer but are not reserved words. Their meaning will be discussed later. Among these are three input and output identifiers: READ, READON, WRITE, (See Sections 2.2.2. and 2.5.)

1.4 Nonarithmetic symbols

The symbols which are printed in bold type in the text are usually underlined in typewritten copy, They are contained in the list of reserved words (cf. Section 1.3) for ALGOL W. They are not distinguished in any other way but they must not be used for any purpose other than that for which they are specifically intended, The symbol END, for example, must not be used as an identifier,

2. Arithmetic Expressions

2.1. Numerical Expressions

The basic arithmetic operators of ALGOL W are

    + - * / ** DIV REM .

EXAMPLES

```
3.1459                                        7 DIV 3
(3.47'-4 + 9.01'+1) / 4                      17 REM 12
9 * 8 * 7 / (1 * 2 * 3)                        -1.2
(9 + 2.7) / (-3)
(((1.5 * 3 - 4) * 3 + 0.19'1) * 3 - 2.6'3) * 3
10 + 1.4 / (1 + 0.9 / (7 - 0.4 / 3))
```

The symbol * denotes multiplication while ** denotes exponentiation. For instance, 4.5 ** 3 means 4.5^3. The exponent must always be an integer in ALGOL W. An integer to any exponent gives a real result.

EXAMPLES

ALGOL W form	Conventional form
4.1 - 3 ** 2	$4.1 - 3^2$
(4.1 - 3) ** 2	$(4.1 - 3)^2$
3.2 ** 2 + 5.2	$3.2^2 + 5.2$
-4 ** 2	-4^2

ALGOL W form	Conventional form
(-4) ** 2	$(-4)^2$
4 * 5 / 2 ** 3	$\frac{4x5}{2^3}$
5 ** 2 * 3	$5^2 \circ 3$

Also notice

2**3**4 $\equiv$ $(2^3)^4$

In ALGOL W the following two constructs are not allowed because the exponent is a real numbers

3.2**(2 +5 . 2) and 2**(3**4)

2.202. Assignment of numerical values through input

If the value of an identifier is to be provided by input it is assumed that this value appears on a data card which is in the card reader waiting to be read, The statement

READØN (V)

where V stands for variable identifier, reads the next number on the current input card. If there are no more numbers on the current input card, subsequent cards are read until a number is found., This statement assigns the value of the number to the variable whose name is specified,

READØN $(V_1, V_2, \ldots, V_n)$

is equivalent to

READØN (V_1); READØN (V_2);...; READØN (V_n) .

The constants on the data cards are assigned in the same order as the variable names in the READØN statement. One or several numbers may appear on a single card separated by one or more blank spaces with column 80 of one card immediately followed by column 1 of the succeeding card.

The statement

READ (V)

is similar to READØN (V) except that scanning for the number begins on a new input card.

The statement

READ $(V_1, V_2, V_3, \ldots, V_n)$

is equivalent to

READ (V_1); READØN $(V_2, V_3, \ldots, V_n)$.

Numbers are punched into data cards in the forms described in Section 1.2, and may be prefixed by "-". Numbers corresponding to variables of type integer must not contain decimal fractions or scale parts.

EXAMPLES READØN (A2)

In this case the data card must contain at least one number, say 1.279'-7 if A2 is not an integer variable.

READ (B10,B11,B12,B15);

The data cards must contain four numbers, say

3.4'-1 7.149 825'1 9 if B10, B11, B12 are not integer variables, B15 may be an integer variable or a real variable, One could spread these constants over several cards if desired, In general input read into the machine must be assignment compatible with the corresponding variable (cf. Section 2.3.2).

2.3. Assignment of numerical values through expressions

Exponentiation a^b (a**b) is defined by repeated multiplication if b is a positive integer and by $1/a^{|b|}$ when b is negative. b must have type integer. If one desires the result of A^R where R is real, use EXP (R * LN (A)).

2.3.1. Evaluation of expressions

The discussion in this paragraph is correct, However, in ALGOL W the type of a resulting expression is defined for each type and each operator. The type complex and the discussion of the long forms is provided for completeness and may be ignored by beginning programmers,

I: A + B, A - B

A \ B	integer	real	complex
integer	integer	real	complex
real	real	real	complex
complex	complex	complex	complex

The result has the quality "long" if both A and B have the quality "long", or if one has the quality "long" and the other is integer.

II: A * B

A \ B	integer	real	complex
integer	integer	long real	long complex
real	long real	long real	long complex
complex	long complex	long complex	long complex

A or B having the quality "long" does not affect the resultant type of the expression.

III: A / B

A \ B	integer	real	complex
integer	real	real	complex
real	real	real	complex
complex	complex	complex	complex

The specifications for the quality "long" are those given for + and

IV: A ** B

A \ B	integer
integer	real
real	real
complex	complex

The result has the quality "long" if and only if A does.

V: ABS A means the "absolute value of A".

A	ABS A
integer	integer
real	real
complex	real

2.3.2. Type of the variable to which a value is assigned,

The assignment V := E is correct only if the type of E is <u>assignment compatible</u> with V. That is; the type of V must be lower or on the same level in the list below as the type of E.

integer

real, long real

complex, long complex

Several transfer functions are provided as standard functions (cf. Section 2.4). For example, to change the type of expression E from real to integer either ROUND(E), TRUNCATE(E) or ENTIER(E) may be used,

2.3.4. Multiple assignments

The assignment of the value of an expression can be extended to several variables. As in ALGOL 60, the form in ALGOL W is

V_1 := V_2 := ... := Vn := E;

. The multiple assignment statement is possible only if all the variables occurring to the left of V_i:= are assignment compatible with the type of the variable or expression to the immediate right of the :=.

2.4 Standard Functions

All the standard functions listed in this section are provided in ALGOL W except sign and abs. ABS is a unary operator in ALGOL W. In addition the following standard functions are provided.

truncate(E)	if $E \leq 0$, then entier(E)
	if $E < 0$, then -entier(-E)
round(E)	if $E \geq 0$, then truncate $(E + 0.5)$
	if $E < 0$, then truncate $(E - 0.5)$
log(E)	the logarithm of E to the base 10 (not defined for $E \leq 0$)
time(E)	if $E = 1$, elapsed time returned in 60th's of a second
	if $E = 2$, elapsed time returned in 60th's of a second and printed in minutes, seconds, and 60th's of a second

2.5. output

The identifier "print" should be replaced by "write". A print line consists of 132 characters.

-EXAMPLES WRITE(E); WRITE($E_1, E_2, \ldots, E_n$);

The format of the values of each type of variable is listed below:

integer	right justified in field of 14 characters and followed by two blanks. Field width can be changed by assignment to INTFIELDSIZE.
real	same as integer except that field width is invariant.

long real	right justified in field of 22 characters followed by 2 blanks,
complex	two adjacent real fields.
long complex	two adjacent long real fields.
logical	TRUE or FALSE right justified, in a field of 6 characters followed by 2 blanks.
string	field large enough to contain the string and continuing onto the next line if the string is longer than 132 characters,,
bits	same as real.

In order to provide headings or labels for printed results, a sequence of characters may be printed by replacing any expression in the write statement by the sequence of characters surrounded by ". If the " mark is desired in a string it must be followed by a ".

EXAMPLES

```
WRITE   ("N = ", N)
```

This statement will cause the following line to be printed if N is integer and has the value 3.

```
N =              3
```

```
WRITE  ("SHAKESPEARE WRØTE ""HAMLET""")
```

This statement will cause the following line to be printed,

```
SHAKESPEARE WRØTE "HAMLET".
```

In the statement

WRITE $(E_1, E_2, \ldots, E_n)$

the type of each E_i determines the field in which its value will be placed. The field for E_{i+1} follows the field for E_i on the current print line.

If there is not enough space remaining on the current print line, the line is printed and the field for E_{i+1} begins at the beginning of a new print line. The first field of each write statement begins on a new print line.

3. Construction of the program

3.1. Simple Statements

Note that the simple assignment statement takes the form V := E and that the input-output statements are respectively

READ (V) and WRITE(E)

where V is a variable or a variable list and E is an expression or expression list.

3.2. Compound Statements

In later descriptions in these notes "compound statements" will be synonomous with "blocks without declarations".

3.4. Comments

The construction

comment text;

may appear anywhere in an ALGOL W program. However, in ALGOL W the comment following an end is limited to one identifier which is not a reserved word.

3.5. Example

To clarify the change necessary to form an AIGOL W program from the program enclosed in the box, the example is shown as it would be punched, Note that an AIGOL W program must end with a . (period).

```
BEGIN COMMENT EVALUATION OF A POLYNOMIAL;
     REAL AO, Al, A2, A3, Xl, P;
     READ (AO, Al, A2, A3, Xl);
     P := ( (A3 * Xl + A2) * Xl + Al) * Xl + AO;
     WRITE (P)
END.
```

Note that the indentation, although not required, allows the begin and end to be matched easily. In complicated programs indentation will improve readability and therefore reduce the number of careless errors.

4. Loops

4.1. Repetition

The variable V of the for statements described is always of the type integer and cannot be declared in AIGOL W; its declaration is implicit (cf. Section 7), and its value cannot be changed by explicit assignment within the controlled statement. Each expression E of the for clause must be of type-integer.

The statement of the form

for V := $H_1, H_2, \ldots, H_n$ do S;

is correct for $n > 1$ in AIGOL W only if $H_1, H_2, \ldots, H_n$ are all integer expressions.

The form

for V := E_1 step 1 until E_2 do S;

may be abbreviated as

for V := E_1 until E_2 do S;

4.2. Subscripted Variables

In ALGOL W the subscript expression must be of type integer. Any other type will result in an error detected during compilation.

4.2.1. Array declarations

In the text, the : in array declarations must be replaced by :: for ALGOL W. The word array must always be preceded by its type.

ARRAY A[1:10,1:20]; is incorrect and should be written

REAL ARRAY A (1::10, 1::20);

Only one set of subscript bounds may be given in an array declaration. Hence, the examples should be corrected for ALGOL W to read

EXAMPLES

```
real array A, B, C(1::10);
real array D, EG(1::10, 1: :20);
integer array N, M(1::4);
```

4.4.2. Example

In ALGOL W the example in the box would be written as listed below,

```
BEGIN COMMENT DERIVATIVE OF A POLYNOMIAL;
    INTEGER N;     REAL P, C;
    REAL ARRAY A(1::20);
    READ (N, C);
    FOR I := 1 UNTIL N DO READON (A(I));
    P := 0;
    FOR I  := N STEP -1 UNTIL 1 DO
        P  := P*C + I*A(I);
    WRITE (P)
END.
```

5. The Conditional Statement

Conditional statements are very useful and are used in ALGOL W as discussed in this chapter for ALGOL 60. Note that the symbols $\leq$, $\geq$, and $\neq$ must be replaced by < =, > =, and ¬ =, respectively.

6. Jumps

6.1. Labels

All labels in ALGOL W must be identifiers which are not reserved words.

6.2. The Jump Statement

go to may be written as GO TO or GOTO in ALGOL W.

6.2.1. Jumps out of loops or conditional statements

The value of the loop variable is not accessible outside of the loop in ALGOL W.

6.2.2. Inadmissible Jumps

It is not possible to jump from outside into a loop in ALGOL W. Likewise, it is not possible to jump into a conditional statement.

In general, it is not possible to jump into the middle of any statement, viz, for statement, conditional statement, while statement, compound statement, block.

6.4. Another Form of Loop Statement

The statement described in the text does not exist in ALGOL W. However, ALGOL W has another form of loop statement which is useful -- it is called the while statement.

```
FORM       while B do S;
```

B is a condition like that described in Chapter 5. As long as B is true, the statement S will be repeated. It is possible that S is never executed. More precisely, this loop may be interpreted

```
L:  if B then
    begin S; goto L
    end
```

The example in Section 6.3 can be rewritten as follows:

```
BEGIN COMMENT DETERMINATION OF THE CUBE ROOT;
     REAL A, APPROXIMATIONVALUE, X, Y, D;
     READ (A, APPROXIMATIONVALUE);
     X := APPROXIMATIONVALUE;      2 := ABS X;
```

```
          WHILE D> .5'-9 * ABS X DO
          BEGIN
               Y := X;  x := (2*Y + A/(Y*Y))/3;
               D := ABS (X-Y);
          END;
     END.
```

7. Block Structure

For the purposes of block structure in ALGOL W compound statements must be considered as blocks, namely blocks without declarations. A compound statement with a label defined in it is a block. (Reread the notes in this paper concerning Chapter 6.) In <u>for</u> statements the scope of the variable V in the <u>for</u> clause is the statement S following the <u>do</u>.

7.4. Dynamic Array Declarations

The expressions specifying the subscript bounds in dynamic array declarations must be of type integer.

8. Propositions and Conditions

The word "Boolean" in the text should be replaced throughout by "logical".

8.1. Logical Operations

-Some of the symbols for logical operations are different in ALGOL w.

Operation	ALGOL	ALGOL W	READ AS
negation	¬	¬	not
conjunction	∧	AND	and
disjunction	∨	OR	or
equivalence	≡	=	is equivalent to

ALGOL W does not have an equivalent form of the ALGOL implication symbol, ⊃.

, The following hierarchical arrangement defines the rank of the operator with respect to other operators.

Level	Operation Symbol
1	LONG, SHORT, ABS
2	SHL, SHR, **
3	¬
4	AND, *, /, DIV, REM
5	OR, +, -
6	<, < =, >, > =, =, ¬ =, IS

In a particular construct, the operations are executed in a sequence from the highest level (smallest number) to the lowest level (largest number): Operations of the same level are executed in order from left to right when logical operations are involved and in undefined order in arithmetic expressions.

The discussion in this section is correct except concerning the hierarchy of operators, In general, the extra parentheses are required in ALGOL W when using arithmetic expressions with logical operators, The examples below are correct ALGOL W and correspond to examples in tne text. All parentheses are necessary.

EXAMPLES

```
(A > 5) OR (B > = 1)
(A * B > = C + D) = (ABS (Z1 + Z2) > M)
(0 < = X) AND (X < = 1)

(X = 3) OR (1 < = X) AND (X < = 2)
means (X = 3) OR ((1 < = X) AND (X < = 2))
```

9. Designational Expressions

The designational expressions described in the text do not exist in ALGOL W. The chapter may be skipped.

However, ALGOL W provides a designational statement and expression which is equivalent to that described by the text.

9.1. The Case Statement

The form

CASE E OF

BEGIN

$S_1;S_2;. . . ;S_n$

END

is called a case statement. The expression E must be of type integer. The value of the expression, E, selects the S_E statement between the BEGIN END pair. Execution is terminated if the value of E is less than 1 or greater than n. After the designated expression is executed, execution continues with the statement following the END.

EXAMPLE

```
CASE I OF
BEGIN
     BEGIN J := I; GOTO L1;
     END;
     I := I + 1;
     IF J < I THEN GOTO L1
END
```

If the value of the expression, I, is 3, for example, the statement, IF J < I THEN GOTO L1 is executed, If J > = I then execution continues following the END.

9.2. The Case Expression

Analogous to the case statement, the case expression has the form

CASE E OF $(E_1, E_2, \ldots, E_n)$

The value of the case expression is the value of the expression selected by the value of the expression E. If the value of E is e, then the value of E_e is the value of the case expression. The type of the case expression is the type of the E_i expression whose type is lowest on the list

integer
real
long real
complex
long complex

EXAMPLE

CASE 3 OF (4.8, 12, 17, 4.9) has the value 17 in floating point representation since the type of the case expression is real.

10. Procedures

10.1.1. Global and formal parameters

Labels may not be used as formal parameters. Switches do not exist in -ALGOL W.

10.1.2.1. Arguments

Arguments serve to introduce computational rules or values into the procedure. A rule of computation can be brought into the procedure if the computation is defined by means of another procedure declaration, or a statement.

Formal simple variables, formal arrays, and formal procedures can be arguments.

Example 3 is correct in the text.

A formal array can be used as an argument in only one way, "call by name". The discussion concerning "call by value" should be ignored.

10.1.2.3. Exits

Because labels may not be used as actual parameters to a procedure, the text's discussion of exits is not correct for ALGOL W. However, a statement (in particular a GOTO statement) may be used as an actual parameter corresponding to a formal procedure identifier. In this way side exits leading out of the procedure are provided,

10.1.3. Function procedures and proper procedures

From given pieces of programs, procedures can be derived either in the form of function procedures or in the form of proper procedures,,

The body of a function procedure is either an expression or a block with an expression before the final END in the procedure body. The value of the expression is the value of the function procedure,

The way in which a procedure is set up and used is a fixed characteristic of the procedure and is established directly in the declaration by means of the introducing-symbols, The declaration of functions is introduced by the symbols

```
INTEGER PROCEDURE
REAL PROCEDURE
LOGICAL PROCEDURE
      .
      .
      .
```

according to the type of the resulting value. The type of the expression giving the value of the procedure must be assignment compatible with the declared type of the function procedure.

The declaration of the proper procedure begins with the symbol

```
PROCEDURE
```

No resulting expression can be placed at the end of the procedure body.

10.1.4. The procedure head

All necessary assertions about the formal parameters and the use of the procedure are contained in the head of the procedure declaration, In ALGOL W the head consists of three parts:

(1) Introductory symbol

(2) Procedure name

(3) List of formal parameters and their specifications

(1) The introductory symbol determines the later use of the procedure (cf. Section 10.1.3.)

(2) The procedure name can be chosen almost arbitrarily. The only restriction is the general limitation concerning some reserved , names (cf. Section 1.3).

(3) The type, value specification, and identifier name of formal parameters appear in the list of formal parameter specifications, and not separately as in ALGOL 60. The comma serves as the general separation symbol between formal parameter identifiers of the same type and vaiue specification. The semicolon serves as the general separation symbol between specifications of formal parameters of different types or value specifications.

The type of the formal parameter is specified by the symbols

REAL
LONG REAL
INTEGER
COMPLEX
LONG COMPLEX
LOGICAL
REAL ARRAY
LONG REAL ARRAY
COMPLEX ARRAY
LONG COMPLEXARRAY
INTEGER ARRAY
LOGICAL ARRAY

```
REAL PROCEDURE
LONG REAL PROCEDURE
COMPLEX PROCEDURE
LONG COMPLEX PROCEDURE
INTEGER PROCEDURE
LOGICAL PROCEDURE
PROCEDURE
```

The value specification is used only for parameters called by value. It is specified by the symbol value. It may only follow the types INTEGER, REAL, LONG REAL, LOGICAL, COMPLEX, LONG COMPLEX.

EXAMPLES

```
PROCEDURE P (REAL X, Y; INTEGER VALUE I; PROCEDURE Q, R);
REAL PROCEDURE Z (LOGICAL L, M, N; REAL PROCEDURE P);
```

Note that in the case of formal parameters used as array identifiers, information about the number of dimensions must be given. The last identifier following each array specification must be followed by (followed by one asterisk for each dimension separated by commas, followed by).

EXAMPLE

```
PROCEDURE P (REAL ARRAY X, Y (*,*); REAL ARRAY Z (*)) .
```

10.2. The Procedure Call

The procedure call in ALGOL W is unchanged from ALGOL 60. This section should be read carefully.

Since labels are not allowed as parameters, it was earlier suggested that jump statements be used and that the corresponding formal parameter be a proper procedure (cf. 10.1.4. Example 8). In general, any statement may be used as an actual parameter corresponding to a formal proper procedure which is used without parameters.

EXAMPLE

```
BEGIN
          PROCEDURE VECTOROPERATIONS (INTEGER J; INTEGER VALUE N;
                              PROCEDURE P);
               BEGIN J := 1;
                    WHILE J < = N DO
                    BEGIN P; J := J + 1
                    END
               END;
          REAL PROD; INTEGER I;
          REAL ARRAY A, B, C(1::10);
                    (initialize A and B)
     L1:  VECTOROPERATIONS (I, 10, C(1) := A(1) + B(1));
          PROD := 0.0;
     L2:  VECTOROPERATIONS (I, 10, PROD := PROD + A(1) * B(1));
     END
```

The statement L1 is a procedure call which causes a vector addition of A and B to be placed in C. The statement L2 causes the element-by-element vector product of A and B to be calculated and placed in PROD.

10.3. Example

```
REAL PROCEDURE ROMBERGINT (REAL PROCEDURE FCT;
    REAL VALUE A, B;   INTEGER VALUE ORD);
  BEGIN REAL T1, L;
    ORD := ENTIER ((ORD + 1) / 2);
    BEGIN INTEGER F, N; REAL M, S;
        REAL ARRAY U, T (1 :: ORD);
        L :=  B-A;
        T(1) := (FCT(A) + FCT(B)) / 2;
        U(1) := FCT ((A + B) / 2);
        F := N := 1;
        FOR H :=2 UNTIL ORD-1 DO
        BEGIN N := 2 * N; S := 0;
            M := L / (2 * N);
            FOR J := 1 STEP 2 UNTIL 2 * N - 1 DO
                S := S + FCT (A + J * M);
            U(H) := S / N;
            T(H) := (T(H - 1) + U(H - 1)) / 2;
            F :=1;
            FOR J := H - 1 STEP -1 UNTIL 1 DO
            BEGIN F := 4 * F;
                T(J) := T(J + 1) + (T(J + 1) - T(J)) / (F - 1);
                U(J) := U(J + 1) + (U(J + 1) - U(J)) / (F - 1);
            END;
            IF ORD > 1 THEN
```

```
                BEGIN
                    T(2) := (U(1) + T(1)) /2;
                    T(1) := T(2) +(T(2) - T(1)) /(4 * F - 1)
                END;
                T1 := T(1)
            END;
        END;
        T1 * L
    END;
```

The names of standard functions and standard procedures cannot appear as actual parameters in ALGOL W. Therefore the calls to RØMBERGINT in Section 10.3 are incorrect. However, this situation may be overcome by declaring a procedure which returns the value of the standard function or performs the computation of the standard procedure.

EXAMPLE

```
    REAL PRØCEDURE SINE (REAL VALUE X); SIN(X);
```

Then a call to RØMBERGINT might be

```
    A := RØMBERGINT (SINE, x(1), X(2), 10);
```

EXAMPLE 6

```
    REAL PROCEDURE TRACE (REAL ARRAY A(*,*); INTEGER VALUE N);
      BEGIN REAL S;
        S := 0;
        FOR I := 1 UNTIL N DO
            S := S + A(I,I);
        S
      END
```

EXAMPLE 7

```
        PROCEDURE COUNTUP (INTEGER X);
             X := X + 1
```

EXAMPLE8

```
        PROCEDURE ROOTEX (REAL VALUE X; REAL Y; PROCEDURE P);
             IF X > = OTHEN
                Y : = SQRT(X)
             ELSE
                BEGIN Y := SQRT(ABS X);
                    P
                END
```

The actual parameter corresponding to the formal parameter P should be a jump statement.

PART II: Some Extensions of ALGOL 60 in ALGOL W

1. Procedures

1.1. Call by Result

Besides "call by value" and "call by name", ALGOL W allows parameters to be called by result. The formal simple variable is handled as a local quantity although no declaration concerning this quantity is present, The value of the simple variable is not initialized at the procedure call. If the procedure exits normally, the value correspoinding to the formal simple variable is assigned to the corresponding actual parameter. The formal parameter must be assignment compatible with the actual parameter. To specify a result parameter, insert the word RESULT after the type and before the identifier (as with VALUE).

EXAMPLE

```
PROCEDURE P(REAL RESULT X,Y; INTEGER VALUE I; LONG COMPLEX RESULT Z);
```

1.2. Call by Value Result

Formal simple variables may be called both by value and result. This combines the calls of value and result so that the formal identifier is initialized to the value of the corresponding actual parameter at procedure call and the value of the formal identifier is assigned to the corresponding actual parameter at a normal procedure exit, To specify a value result parameter, insert the words VALUE RESULT after the type and before the identifiers.

EXAMPLE

```
PROCEDURE Q(INTEGER VALUE RESULT I,J,K);
```

2. Procedure Calls

2.1. Sub-arrays as Actual Parameters

In ALGOL W, it is possible to pass any rectangular sub-array of an actual or formal array to a procedure, Those dimensions which are to be passed to the procedure are specified by *'s, and those which are to remain fixed are specified by integer expressions. The number of dimensions passed must equal the number of dimensions specified for the corresponding formal array.

EXAMPLE

The actual parameter may be a sub-array of a three dimensional real array A. Examples of possible actual parameter specifications and corresponding formal parameter specifications are listed below.

Actual Parameter	Corresponding Formal Parameter Specification
A or A(*,*,*)	<u>real array</u> B(*,*,*)
A(I,*,*)	<u>real array</u> B(*,*)
A(*,I,*)	<u>real array</u> B(*,*)
A(*,*,I)	real <u>array</u> B(*,*)
A(I,J,*)	<u>real array</u> B(*)
A(I,*,J)	<u>real array</u> B(*)

EXAMPLE

Read in the size of one dimension of a cubic array X, then read in the elements of X.

Calculate and write out the sum of the traces of all possible two dimensional arrays in A using the previously defined real procedure TRACE.

```
BEGIN
     REAL SUM;
     REAL PROCEDURE TRACE (REAL ARRAY A(*,*); INTEGER VALUE N);
          BEGIN COMMENT THE BODY OF THIS PROCEDURE IS GIVEN IN A
               PREVIOUS EXAMPLE: ;
          END;
     INTEGER N;
     READ(N);
     BEGIN
          REAL ARRAY X(1::N, 1::N, 1::N);
          FOR I := 1 UNTIL N DO
               FOR J := 1 UNTIL N DO:
                    FOR K := 1 UNTIL N DO READON(X(I,J,K));
          SUM := 0;
          FOR I := 1 UNTIL N DO
               SUM := SUM + TRACE(X(I,*,*),N) + TRACE (X(*,I,*),N)
                       + TRACE (X(*,*,I),N);
          WRITE (SUM)
     END
END.
```

3. String Variables

Frequently, it is desirable to manipulate sequences of characters. This facility is available in ALGOL W in the form of string variables, Each variable has a fixed length specified in the string declaration. The form of' the declaration is string (<integer number>) <variable list>

The integer number must be greater than 0 and less than or equal to 2560 The specification "(<integer number>)'" may be omitted; a default length of 16 is assigned to the variables. Arrays of strings also may be declared,

EXAMPLE

```
STRING A, B, C
STRING (24) X, Y, Z
STRING (10) ARRAY R, S(0::10,-5::15)
```

In order to be able to inspect elements of the string or to manipulate portions of the string, a substring operation is provided.

```
FORM <string identifier> (E | <integer number>?
```

The expression E must be of type integer, This string expression selects a substring of the length specified by the integer number from the string variable beginning at the character specified by the integer expression, The first character of the string has position 0.

EXAMPLE

```
BEGIN STRING (5) A;
    A : = "QRSTU" ;
    A (3|2) := A (0|2);
    WRITE (A)
END
```

In this example the constant string "QRSTU", is assigned to the variable A which is declared to be of length 5. Then the character positions 0 and 1 of A are assigned to positions 3 and 4 of A.

Consequently, when the string A is written its value is QRSQR. It should be noted that the assignments are made character by character. If the second assignment statement in the example above had been

```
A (2|3) := A(0|3)
```

the resulting value of A would have been QRQRQ.

4. Records and References

Records are structured quantities composed of quantities of any of the simple types such as REAL, INTEGER, STRING, etc. Records themselves do not have values; only the quantities which compose the records may have values.

4.1. ,Record Class Declarations

Record declarations indicate the composition of a record. Unlike simple type declarations or array declarations no storage is reserved for a record when the record declaration is encountered. Essentially, the record declaration only describes the form of records to 'be created, The record declarations appear with all other declarations.

FORM:

```
RECORD V ( <declarations of variables of simple type>);
```

The name V is the name of the record class . The variables declared between the parentheses are called the fields of the record.

EXAMPLES

```
RECORD A(INTEGER I,J; REAL Z; STRING (5) S);
RECORD B(REAL X; LONG REAL LX; REAL Y);
```

The punctuation of the examples should be noted carefully, The names in the list of identifiers following the indication of the simple type are separated by ",". The listis ended with a ";" unless the ";" would immediately precede the closing ")".

4.2. Reference Declarations

In order to specify a record of some record class, REFERENCE is a simple type in ALGOL W. The value of a variable of type reference is an address of a record, This address is sometimes called a pointer to a record..

Reference declarations appear in a program where ail other declarations appear.

FORM

REFERENCE (V) V_1;

V is a name of a record class, V_1 is a name of a reference variable or a list of names of reference variables separated by ",".

EXAMPLE

REFERENCE (A) R1, R2, R3;

The name V of a record class may also be a list of Tames separated 'by ",". This list indicates the record classes to which records referenced by the reference variables must 'belong.

EXAMPLE

REFERENCE (A,B) 34, 35;

R4 and R5 may point, only to records of record class A or B.

The reserved word NULL stands for a reference constant which fails to designate a record,

Arrays of references are declared and used analogously to arrays of other simple types.

FORM

```
REFERENCE (V) ARRAY V1. (<subscript bounds>);
```

EXAMPLE

```
REFERENCE (A,B) ARRAY AR1, AR2 (1::10, 3::7);
```

The implementation requires that all reference arrays declared in a block be declared in the same reference array declaration or immediately following a reference array declaration.

EXAMPLE

```
REFERENCE (A) ARRAY AR1, AR2 (1::10, 3::7);
REFERENCE (B) ARRAY AR3 (2: :17);
```

In the example above, any other declaration except a reference array declaration is not allowed between the two reference array declarations,

4.3. Reference Expressions

Quantities of simple type reference may be used in assignmeni; statements and comparisons,

EXAMPLES

```
R1 := R2
RI := NULL
R1 = R2
R2 ¬ = R3
```

Only the relations = and ¬ = are allowed between references. In order to inquire to which record class a reference expression is bound, the IS operator is provided.

FORM

```
E IS V
```

E is a reference expression and V is a name of a record class. The value of the IS operator is logical, either TRUE or FALSE.

'EXAMPLE

4.4. Record Designators

A particular type of reference expression is the record designator. A record designator is the name of a record class when used as an expression.

EXAMPLE

```
R1 := A
R4 := B
```

When the record class name is encountered, the value is a pointer to a new record of that class. The values of the fields of the new record are undefined.

4.5. Field Designators

In order to manipulate the values of the fields of a record, the expression

FORM

$V_1(E)$

exists in ALGOL W. E is a reference expression. V_1 is a field of the record class of the record pointed to by E. The type of the field designator is the type of the variable V_1.

EXAMPLES

```
Z(R1)
LX(R4)
```

EXAMPLE 1

```
BEGIN RECORD H (INTEGER C,D; STRING (2) S);
     REFERENCE (H) R1;
     R1 := H:
     C(R1) := 5;
     D(R1) := 8;
     S(R1) := "AZ"
END.
```

Example 1 is a short program which declares a record class H and one reference -variable R1 whose values may point to records of class H. One record of class H is created and each field of the record pointed to by R1 is initialized,

ALGOL W provides a short notation for creating a record and initializing its fields, This modified record creator has the form

$$V(E_L) .$$

V is the name of the record class. The expression list E_L between the parentheses is the list of the values of the fields specified in the order they appear in the record. class declaration.

EXAMPLE

```
     B(4.8, 3.14159E, 8'6)
```

Example 1 may be rewritten as follows:

EXAMPLE 2

```
     BEGIN RECORD H (INTEGER C,D; STRING (2) S);
          REFERENCE (H) R1;
          R1 := H(5, 8, "AZ")
     END.
```

Algol W Deck Set-Up

```
        < Job Card >
        //JOBLIB DD DSNAME=SYS2.PROGLIB,DISP=(OLD,PASS
        //   EXEC  ALGOLW
        //ALGOLW.SYSIN DD *
      ┌ %ALGOL
  **  {      < program >
      │ %EOF
      │ ┌
      │*{      < data >
      └ └ %EOF
        /*
```

* Optional

** May be repeated

ALGOL W

LANGUAGE DESCRIPTION

by

Henry R. Bauer
Sheldon Becker
Susan L. Graham

COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY
JANUARY 1968

"A Contribution to the Development of ALGOL" by Niklaus Wirth and C. A. R. Hoare[1)] was the basis for a compiler developed for the IBM 360 at Stanford University. This report Is a description of the implemented language, ALGOL W. Historical background and the goals of the language may be found in the Wirth and Hoare paper.

1) Wirth, Niklaus and Hoare, C. A. R., "A Contribution to the Development of ALGOL"', Comm. ACM 9, 6(June 1966), pp. 413-431.

CONTENTS

CONTENTS (cont.)

CONTENTS (cont.)

1. TERMINOLOGY, NOTATION AND BASIC DEFINITIONS

The Reference Language is a phrase structure langauage, defined by a formal system. This formal system makes use of the notation and definitions explained below. The structure of the language ALGOL W is determined by three quantitites:

(1) $\mathcal{V}$, the set of basic constituents of the language,
(2) $\mathcal{U}$, the set of syntactic entities, and
(3) $\mathcal{P}$, the set of syntactic rules, or productions.

1.1. Notation

A syntactic entity is denoted by its name (a sequence of letters) enclosed in the brackets < and >. A syntactic rule has the form

```
<A> ::= x
```

where <A> is a member of $\mathcal{U}$, x is any possible sequence of basic constituents and syntactic entities, simply to be called a "sequence". The form

```
<A> ::= x | y | ... | z
```

is used as an abbreviation for the set of syntactic rules

```
<A> ::= x
<A> ::= y
.........
<A> ::= z
```

1.2. Definitions

1. A sequence x is said to <u>directly produce</u> a sequence y if and

only if there exist (possibly empty) sequences u and w, so that either (i) for some <A> in $\mathcal{U}$, x = u<A>w, y = uvw, and<& ::= v'is a rule in $\mathcal{P}$; or (ii) x = uw, y = uvw and v is a "comment" (see below).

2. A sequence x is said to <u>produce</u> a sequence y if and only if there exists an ordered set of sequences s[0], s[1], . . . , s[n], so that x = s[0], s[n] = y, and s[i-1] directly produces s[i] for all i = 1, . . . , n.

3. A sequence x is said to be an ALGOL W program if and only if its constituents are members of the set If, and x can be produced from the syntactic entity <program>.

The sets $\mathcal{V}$ and $\mathcal{U}$ are defined through enumeration of their members in Section 2 of this Report (cf. also 4.4.). The syntactic rules are given throughout the sequel of the Report. To provide explanations for the meaning of ALGOL W programs, the letter sequences denoting syntactic entities have been chosen to be English words describing approximately the nature of that syntactic entity or construct. Where words which have appeared in this manner are used elsewhere in the text, they refer to the corresponding syntactic definition. Along with these letter sequences the symbol $\mathcal{T}$ may occur. It is understood that this symbol must be replaced by any one of a finite set of English words (or word pairs). Unless otherwise specified in the particular section, all occurrences of the symbol $\mathcal{T}$ within one syntactic rule must be replaced consistently, and the replacing words are

integer
real
long real
complex
long complex

logical
bits
string
reference

For example, the production

<$\mathcal{T}$ term> ::= <$\mathcal{T}$ factor> (cf. 6.3.1.)

corresponds to

<integer term> ::= <integer factor>
<real term> ::= <real factor>
<long real term> ::= <long real factor>
<complex term> ::= <complex factor>
<long complex term> ::= <long complex factor>

The production

<$\mathcal{T}_0$ primary> ::= long <$\mathcal{T}_1$ primary> (cf. 6.3.1. and table for long 6.3.2.7.)

corresponds to

<long real primary> ::= long <real primary>
<long real primary> ::= long <integer primary",
<long complex primary> ::= long <complex primary>

It is recognized that typographical entities exist of lower order than basic symbols, called characters. The accepted characters are those of the IBM System 360 EBCDIC code.

The symbol comment followed by any sequence of characters not containing semicolons, followed by a semicolon, is called a comment. A comment has no effect on the meaning of a program, and is ignored during execution of the program. An identifier (cf. 3.1.) immediately following the basic symbol end is also regarded as a comment.

The execution of a program can be considered as a sequence of units of action. The sequence of these units of action is defined as the evaluation of expressions and the execution of statements as denoted by the program. In the definition of the implemented language the evaluation or execution of certain constructs is either (1) defined by System 360 operations, e.g., real arithmetic, or (2) left undefined, e.g., the order of evaluation of arithmetic primaries in expressions, or (3) said to be not valid or not defined.

2. SETS OF BASIC SYMBOLS AND SYNTACTIC ENTITIES

2.1. Basic Symbols

```
A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
Q | R | S | T | U | V | W | X | Y | Z |
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
true | false | " | null | # | ' |
integer | real | complex | logical | bits | string |
reference | long real | long complex | array |
procedure | record |
, | ; | : | . | ( | ) | begin | end | if | then | else |
case | of | + | - | * | / | ** | div | rem | shr | shl | is |
abs | long | short | and | or | ¬ | | | = | ¬ = | < |
<= | > | >= | :: |
:= | goto | go to | for | step | until | do | while |
comment | value | result
```

All underlined words, which we call "reserved words", are represented by the same words in capital letters in an actual program.

Adjacent reserved words, identifiers (cf. 3.1.) and numbers must be separated by at least one blank space. Otherwise blanks have no meaning and can be used freely to improve the readability of the program,

2.2, Syntactic Entities

(with corresponding section numbers)

<actual parameter list>	7.3	<formal type>	5.3
<actual parameter>	7.3	<go to statement>	7.4
<bit factor>	6.5	<hex dig:-">	4.3
<bit primary',	6.5	<identifier list>	3.1
<bit secondary>	6.5	<identifier>	3.1
<bit sequence>	4.3	<if clause>	6
<bit term>	6.5	<imaginary number>	4.1
<block body>	7.1	<increment>	7.7
<block head>	7.1	<initial value>	7.7
<block>	7.1	<iterative statement>	7.7
<bound pair list>	5.2	<label definition>	7.1
<bound pair>	5.2	<label identifier>	3.1
<ease clause>	6	<letter>	3.1
<case statement>	7.6	<limit>	7.7
<control identifier>	3.1	<logical element>	6.4
<declaration>	5	<logical factor>	6.4
<digit3	3.1	<logical primary>	6.4
<dimension specification>	5.3	<logical term>	6.4
<equality operator>	6.4	<logical value>	4.2
<expression list>	6.7	<lower bound>	5.2
<field list>	5.4	<null reference	4.5
<for clause>	7.7	<procedure declaration>	5.3
<for list>	7.7	<procedure heading>	5.3
<formal array parameter>	5.3	<procedure identifier>	3.1
<formal parameter list>	5.3	<procedure statement>	7.3
<formal parameter segment3	5.3	<program>	7

3. IDENTIFIERS

3.1. Syntax

```
<identifier> ::= <letter> | <identifier> <letter> | <identifier> <digit>
<𝒯 variable identifier> ::= <identifier>
```

```
<𝒯 array identifier> ::= <identifier>
<procedure identifier>  ::=  <identifier>
<𝒯 function identifier>  ::=  <identifier>
<record class identifier> ::= <identifier>
<𝒯 field identifier>  ::=  <identifier>
<label identifier>  ::= <identifier>
<control identifier>  ::=  <identifier>
<letter>  ::=  A | B | C | D | E | F | G | H | I | J | K | L | M |
               N | O | P | Q | R | S | T | U | V | W | X | Y | Z
<digit>  ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<identifier list>  ::=  <identifier> | <identifier list> , <identifier>
```

3.2. Semantics

Variables, arrays, procedures, record classes and record fields are said to be quantities. Identifiers serve to identify quantities, or they stand as labels, formal parameters or control identifiers. Identifiers have no inherent meaning, and can be chosen freely in the reference language. In an actual program a reserved word cannot be used as an identifier.

Every identifier used in a program must be defined, This is achieved through

(a) a declaration (cf. Section 5), if the identifier identifies a quantity. It is then said to denote that quantity and to be a 𝒯 variable identifier, 𝒯 array identifier, 𝒯 procedure identifier, 𝒯 function identifier, record class identifier or 𝒯 field identifier, where the symbol 𝒯 stands for the appropriate word reflecting the type of the declared quantity;

(b) a label definition (cf. 7.1.), if the identifier stands as a label. It is then said to be a label identifier;

(c) its occurrence in a formal parameter list (cf. 5.3.). It is then said to be a formal parameter;

(d) its occurrence following the symbol <u>for</u> in a for clause (cf. 7.7.). It is then said to be a control identifier;

(e) its implicit declaration in the language. Standard procedures, standard functions, and predefined variables may be considered to be declared in a block containing the program.

The recognition of the definition of a given identifier is determined by the following rules:

Step 1. If the identifier is defined by a declaration of a quantity or by its standing as a label within the smallest block (cf. 7.1.) embracing a given occurrence of that identifier, then it denotes that quantity or label. A statement following a procedure heading (cf.5.3.) or a for clause (cf. 7.7.) is considered to be a block.

Step 2. Otherwise, if that block is a procedure body and if the given identifier is identical with a formal parameter in the associated procedure heading, then it stands as that formal parameter.

Step 3. Otherwise, if that block is preceded by a for clause and the identifier is identical to the control identifier of that for clause, then it stands as that control identifier.

Otherwise, these rules are applied considering the smallest block embracing the block which has previously been considered, If either step 1 or step 2 could lead to more than one definition, then the identification is undefined.

The scope of a quantity, a label, a formal parameter, or a control identifier is the set of statements in which occurrences of an identifier may refer by the above rules to the definition of that quantity, label, formal parameter or control identifier.

3.3. <u>Examples</u>

```
I
PERSON
ELDERSIBLING
x15, X20, x25
```

4. VALUES AND TYPES

Constants and variables (cf. 6.1.) are said to possess a value. The value of a constant is determined by the denotation of the constant. In the language, all constants (except references) have a reference denotation (cf. 4.1.-4.4.). The value of a variable is the one most recently assigned to that variable. A value is (recursively) defined as either a simple value or a structured value (an ordered set of one or more values). Every value is said to be of a certain type. The following types of simple values are distinguished:

<u>integer</u>: the value is a 32 bit integer,
<u>real</u>: the value is a 32 bit floating point number,
<u>long real</u>: the value is a 64 bit floating point number,
<u>complex</u>: the value is a complex number composed of two numbers of type <u>real</u>,

iong complex: the value is a complex number composed of two long real numbers,

logical: the value is a logical value,

bits: the value is a linear sequence of 32 bits,

string: the value is a linear sequence of at most 256 characters,

reference: the value is a reference to a record.

The following types of structured values are distinguished:

array: the value is an ordered set of values, all of identical simple type,

record: the value is an ordered set of simple values.

A procedure may yield a value, in which case it is said to be a function procedure, or it may not yield a value, in which case it is called a proper procedure. The value of a function procedure is defined as the value which results from the execution of the procedure body (cf. 62.2.).

Subsequently, the reference denotation of constants is defined. The reference denotation of any constant consists of a sequence of characters. This, hcwever, does not imply that the value of the denoted constant is a sequence of characters, nor that it has the properties of a sequence of characters, except, of course, in the case of strings.

4.1. Numbers

4.1.1. Syntax

```
<long complex number> ::= <complex number>L
<complex number> ::= <imaginary number>
<imaginary number> ::= <real number>I | <integer number>I
```

```
<long real number>  ::=  <real number>L | <integer number>L
<real number> ::=  <unscaled real> | <unscaled real> <scale factor> |
                   <integer number> <scale factor> | <scale factor>
<unscaled real>  ::= <integer number> . <integer number> |
                     -<integer number>
<scale factor> ::=  '<integer number> | '<sign> <integer number>
<integer number> ::= <digit> | <integer number> <digit>
<sign> ::=  +  |  -
```

4.1.2. Semantics

Numbers are interpreted according to the conventional decimal notation. A scale factor denotes an integral power of 10 which is multiplied by the unscaled real or integer number preceeding it. Each number has a uniquely defined type.

4.1.3. Examples

```
  1              '5              1I
  0100           1'3             0.671
 3.1416          6.02486'+23    1IL
 2.718281828459045235360287L     2.3'-6
```

4.2. Logical Values

4.2.1. Syntax

```
<logical value> ::= true | false
```

4.3. Bit Sequences

4.3.1. Syntax

```
<bit sequence>  ::= # <hex digit> | <bit sequence <hex digit>
<hex digit> ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B |
                 C | D | E | F
```

Note that 2 | ... | F corresponds to 2_{10} | . . . | 15_{10}.

4.3.2. Semantics

The number of bits in a bit sequence is 32 or 8 hex digits. The bit sequence is always represented by a 32 bit word with the specified bit sequence right justified in the word and zeros filled in on the left.

4.3.3. Examples

```
#4F  =  0000 0000 0000 0000 0000 0000 0100 1111
#9   =  0000 0000 0000 0000 0000 0000 0000 1001
```

4.4. Strings

4.4.1. Syntax

<string> ::= "<sequence of character@"

4.4.2. Semantics

Strings consist of any sequence of (at most 256) characters accepted by the System 360 enclosed by ", the string quote. If the string quote appears in the sequence of characters it must be immediately followed by a second string quote which is then ignored. The number of characters in a string is said to be the length of the string.

4.4.3. Examples

" JOHN"

"""" is the string of length 1 consisting of the string quote.

4.5. References

4.5.1. Syntax

<null reference> ::= null

4.5.2. Semantics

The reference value null fails to designate a record; if a reference expression occurring in a field designator (cf. 6.1.) has this value, then the field designator is undefined.

5. DECLARATIONS

Declarations serve to associate identifiers with the quantities used in the program, to attribute certain permanent properties to these quantities (e.g. type, structure), and to determine their scope, The quantities declared by declarations are simple variables, arrays, procedures and record classes.

Upon exit from a block, all quantities declared or defined within that block lose their value and significance (cf. 7.1.2. and 7.4.2.).

Syntax:

```
<declaration>  ::= <simple variable declaration> | <J array
                   declaration> | <procedure declaration> |
                   <record class declaration>
```

5.1. Simple Variable Declarations

5.1.1. Syntax

```
<simple variable declaration>  ::= <simple type> <identifier list>
<simple type>  ::=  integer | real | long real | complex | long
                    complex | logical | bits | bits (32) |
                              string | string (<integer>) | reference
                              (<record class identifier list>)
<record class identifier list> ::= <record class identifier> |
                                   <record class identifier list> ,
                                   <record class identifier>
```

5.1.2. Semantics

Each identifier of the identifier list is associated with a variable which is declared to be of the indicated type. A variable is called a simple variable, if its value is simple (cf. Section 4). If a variable is declared to be of a certain type, then this implies that only values which are assignment compatible with this type (ef. 7.2.2.) can be assigned to it. It is understood that the value of a variable is equal to the value of the expression most recently assigned to it.

A variable of type bits is always of length 32 whether or not the declaration specification is included.

A variable of type string has a length equal to the unsigned integer in the declaration specification. If the simple type is given only as string, the length of the variable is 16.

A variable of type reference may refer only to records of the record classes whose identifiers appear in the record class identifier list of the reference declaration specification.

5.1.3. Examples

```
integer I, J, K, M, N
real X, Y, Z
long complex C
logical L
bits G, H
```

```
string  (10) S, T
reference  (PERSON) JACK, JILL
```

5.2" Array Declarations

5.2.1. Syntax

```
<T array declaration> ::= <simple type> array <identifier list>
                              (<bound pair list>)
<bound pair list>  ::= <bound pair> |<bound pair list>,<bound
                          pair>
<bound pair>  ::=  <lower bound> :: <upper bound>
<lower bound>  ::=  <integer  expression">
<upper bound>  ::=  <integer expression>
```

5.2.2. Semantics

Each identifier of the identifier list of an array declaration is associated with a variable which is declared to be of type array. A variable of type array is an ordered set of variables whose type is the simple type preceding the symbol array. The dimension of the array is the number of entries in the bound pair list.

Every element of an array is identified by a list of indices. The indices are the integers between and including the values of the lower-bound and the upper bound. Every expression in the bound pair list is evaluated exactly once upon entry to the block in which the declaration occurs. In order to be valid, for every bound pair, the value of the upper bound must not be less than the value of the lower bound.

5.2.3. Examples

```
integer array  H(1::100)
```

real array A, B(1::M, 1::N)
string (12) array STREET, TOWN, CITY (J::K + 1)

5.3. Procedure Declarations

5.3.1. Syntax

```
<procedure declaration> ::= <proper procedure declaration> |
                            <T function procedure declaration>
<proper procedure declaration> ::= procedure <procedure heading>;
                                   <proper procedure body>
<T function procedure declaration> ::= <simple type> procedure
                                       <procedure heading>;
                                       <T function procedure body>
<proper procedure body=> ::= <statement>
<T function procedure body> ::= <T expression> | <block body>
                                <T expression> end
<procedure heading> ::= <identifier> | <identifier> (<formal
                        parameter list>)
<formal parameter list> ::= <formal parameter segment> |
                            <formal parameter list> ; <formal
                            parameter segment>
<formal parameter segment> ::= <formal type> <identifier list3 |
                               <formal array parameter>
<formal type> ::= <simple type> | <simple type> value | <simple
                  type> result | <simple type> value result |
                  <simple type> procedure | procedure
<formal array parameter> ::= <simple type> array <identifier
                             list> (<dimension specification>)
-<dimension specification> ::= * | <dimension specification> , *
```

5.3.2. Semantics

A procedure declaration associates the procedure body with the identifier immediately following the symbol procedure. The principal part of the procedure declaration is the procedure body. Other parts of the block in whose heading the procedure is declared can then cause this procedure body to be executed or evaluated. A proper procedure is activated by a procedure statement (cf. 7.3.), a function procedure by a function designator (cf. 6.2,). Associated with the procedure body is a heading containing the procedure identifier and possibly a list of formal parameters.

5.3.2.1. Type specification of formal-parameters. All formal parameters of a formal parameter segment are of the same indicated type. The type must be such that the replacement of the formal parameter by the actual parameter of this specified type leads to correct ALGOL W expressions and statements (cf. 7.3.2.).

5.3.2.2. The effect of the symbols <u>value</u> and <u>result</u> appearing in a formal type is explained by the following rule, which is applied to the procedure body before the procedure is invoked:

(1) The procedure body is enclosed by the symbols <u>begin</u> and <u>end</u> if it is not already enclosed by these symbols;

(2) For every formal parameter whose formal type contains the symbol <u>value</u> or <u>result</u> (or both),

(a) a declaration followed by a semicolon is inserted after the first <u>begin</u> of the procedure body, with a simple type as indicated in the formal type, and with an identifier different from any identifier valid at the place of the declaration.

(b) throughout the procedure body, every occurrence of the formal parameter identifier is replaced by the identifier defined in step 2a;

(3) If the formal type contains the symbol value, an assignment statement (cf. 7.2.) followed by a semicolon is inserted after the declarations of the procedure body. Its left part contains the identifier defined in step 2a, and its expression consists of the formal parameter identifier. The symbol value is then deleted;

(4) If the formal type contains the symbol result, an assignment statement preceded by a semicolon is inserted before the symbol end which terminates a proper procedure body. n the case of a function procedure, an assignment statement is inserted after the final expression of the function procedure body. Its left part contains the formal parameter identifier, and its expression consists of the identifier defined in step 2a. The symbol result is then deleted.

5.3.2.3. Specification of array dimensions. The number of "*"'s appearing in the formal array specification is the- dimension of the array parameter.

5.3.3. Examples

```
procedure INCREMENT; X := X+1
real procedure    X   (real value X, Y);
    if X < Y then Y else X
```

```
procedure  COPY (real array U, V (*,*); integer value A, B);
     for I := 1 until A do
     for J := 1 until B do U(I,J) := V(I,J)

real procedure  HORNER (real array A (*); integer value N;
     real value X);
     begin real S; S := 0;
          for I := 0 until N do S := S * X + A(1);
          S
     end

long real procedure  SUM (integer K, N; long real X);
     begin long real Y; Y := 0; K := N;
          while  K > = 1 do
          begin Y := Y + X; K := K - 1
          end;
          Y
     end

reference (PERSON) procedure YOUNGESTUNCLE (reference (PERSON) R);
     begin reference  (PERSON) P, M;
          P := YOUNGESTOFFSPRING (FATHER (FATHER (R)));
          while (P ¬ = null) and (¬ MALE (P)) or
               (P = FATHER (R))  do
               P := ELDERSIBLING (P);
          M := YOUNGESTOFFSPRING (MOTHER (MOTHER (R)));
          while (M ¬ = null) and  (¬ MALE (M)) do
               M := ELDERSIBLING (M);
          if P = null then M else
          if M = null then P else
          if AGE(P) < AGE(M) then P else M
     end
```

5.4. <u>Record Glass Declarations</u>

5.4.1. Syntax

```
<record class declaration> ::=  record <identifier> (<field list>)
<field list>  ::=  <simple variable declaration> | <field list> ;
                   <simple variable declaration>
```

5.4.2. Semantics

A record class declaration serves to define the structural properties of records belonging to the class. The principal constituent of a record class declaration is a sequence of simple variable declarations which define the fields and their simple types for the records of this class and associate identifiers with the individual fields. A record class identifier can be used in a record designator (cf. 6.7.) to construct a new record of the given class.

5.4.3. Examples

<u>record</u> NODE (<u>reference</u> (NODE) LEFT, RIGHT)
<u>record</u> PERSON (<u>string</u> NAME; <u>integer</u> AGE; logical MALE;
<u>reference</u> (PERSON) FATHER, MOTHER, YOUNGESTOFFSPRJP,
    ELDERSIBLING)

6. EXPRESSIONS

Expressions are rules which specify how new values are computed from existing ones. These new values are obtained by performing the operations indicated by the operators on the values of the operands. Several simple types of expressions are distinguished. Their structure is defined by the following rules, in which the symbol $\mathcal{T}$ has to be replaced consistently as described in Section I, and where the triplets $\mathcal{T}_0$, $\mathcal{T}_1$, $\mathcal{T}_2$ have to be either consistently replaced by the words

logical
bits
string
reference

or by any combination of words as indicated by the following table, which yields $\mathcal{T}_0$ given $\mathcal{T}_1$ and $\mathcal{T}_2$:

$\mathcal{T}_1$ \ $\mathcal{T}_2$	integer	real	complex
integer	integer	real	complex
real	real	real	complex
complex	complex	complex	complex

$\mathcal{T}_0$ has the quality "long" if either both $\mathcal{T}_1$ and $\mathcal{T}_2$ have that quality, or if one has the quality and the other is "integer".

Syntax:

<$\mathcal{T}$ expression> ::= <simple $\mathcal{T}$ expression> | <case clause?
(<$\mathcal{T}$ expression list>)
<$\mathcal{T}_0$ expression> ::= <if clause> <simple $\mathcal{T}_1$ expression> else
<$\mathcal{T}_2$ expression>
<$\mathcal{T}$ expression list> ::= <$\mathcal{T}$ expression>
<$\mathcal{T}_0$ expression list> ::= <$\mathcal{T}_1$ expression list> , <$\mathcal{T}_2$ expression>
<if clause> ::= if <logical expression> then
<case clause> ::= case <integer expression> of

The operands are either constants, variables or function designators or other expressions between parentheses, The evaluation of operands other than constants may involve smaller units of action such as the evaluation of other expressions or the execution of statements.

The value of an expression between parentheses is obtained by evaluating that expression. If an operator has two operands, then these operands may be evaluated in any order with the exception of the logical operators discussed in 6.4.2.2. The construction

<if clause> <simple $\mathcal{T}_1$ expression> <u>else</u> <$\mathcal{T}_2$ expression>

causes the selection and evaluation of an expression on the basis of the current value of the logical expression contained in the if clause. If this value is <u>true</u>, the simple expression following the if clause is selected, if the value is <u>false</u>, the expression following <u>else</u> is selected. The construction

<case clause> (<$\mathcal{T}$ expression list>)

causes the selection of the expression whose ordinal number in the expression list is equal to the current value of the integer expression contained in the case clause. In order that the case expression be defined, the current value of this expression must be the ordinal number of some expression in the expression list.

6.1. <u>Variables</u>

6.1.1. Syntax

```
<simple T variable> ::= <T variable identifier> | <T field designator> |
                        <T array designator>
<T variable> ::= <simple T variable | <simple string variable3
                 (<integer expression> | <integer number>)
<T field designator> ::=<T field identifier> (<reference expression>)
<T array designator3 ::= <T array identifier", (<subscript list>)
<subscript list> ::= <subscript> | <subscript list>, <subscript>
<subscript> ::= <integer expression>
```

6.1.2. Semantics

An array designator denotes the variable whose indices are the current values of the expressions in the subscript list. The value of each subscript must lie within the declared bounds for that subscript position.

A field designator designates a field in the record referred to by its reference expression. The simple type of the field designator is defined by the declaration of that field identifier in the record class designated by the reference expression of the field designator (cf. 5.4.).

6.1.3. Examples

```
X            A(I)        M(I+J, I-J)
FATHER (JACK)            MOTHER(FATHER(JILL))
```

6.2. Function Designators

6.2.1. Syntax

<J function designator> ::= <J function identifier> | <J function identifier> (<actual parameter list>)

6.2.2. Semantics

A function designator defines a value which can be obtained by a process performed in the following steps:

Step 1. A copy is made of the body of the function procedure whose procedure identifier is given by the function designator and of the actual parameters of the latter.

Steps 2, 3, 4, As specified in 7.3.2.

Step 5. The copy of the function procedure body, modifed as indicated in steps 2-4, is executed. The value of the function designator is the value of the expression which constitutes or is part of the modified function procedure body. The simple type of the function designator is the simple type in the corresponding function procedure declaration.

6.2.3. Examples

```
MAX (x **2, Y **2)
SUM (I, 100, H(1))
SUM (I, M, SUM (J, N, A(I,J)))
YOUNGESTUNCLE (JILL)
SUM (I, 10, x(1) * Y(I))
HORNER (X, 10, 2.7)
```

6.3. Arithmetic Expressions

6.3.1. Syntax

In any of the following rules, every occurrence of the symbol $\mathcal{T}$ must be systematically replaced by one of the following words (or word pairs):

integer
real
long real
complex
long complex

The rules governing the replacement of the symbols $\mathcal{T}_0$, $\mathcal{T}_1$ and $\mathcal{T}_2$ are given in 6.3.2.

<simple $\mathcal{T}$ expression> ::= <$\mathcal{T}$ term> | + <$\mathcal{T}$ term> | - <$\mathcal{T}$ term>

<simple $\mathcal{T}_0$ expression> ::= <simple $\mathcal{T}_1$ expression> + <$\mathcal{T}_2$ term> |
<simple $\mathcal{T}_1$ expression> - <$\mathcal{T}_2$ term>

<$\mathcal{T}$ term> ::= <$\mathcal{T}$ factor>

<$\mathcal{T}_0$ term> ::= <$\mathcal{T}_1$ term> * <$\mathcal{T}_2$ factor>

<$\mathcal{T}_0$ term> ::= <$\mathcal{T}_1$ term> / <$\mathcal{T}_2$ factor>

<integer term> ::= <integer term> $\underline{\text{div}}$ <integer factor> |
<integer term> $\underline{\text{rem}}$ <integer factor>

<$\mathcal{T}_0$ factor> ::= <$\mathcal{T}_0$ primary> | <$\mathcal{T}_1$ factor> ** <integer primary>

<$\mathcal{T}_0$ primary> ::= $\underline{\text{abs}}$ <$\mathcal{T}_1$ primary> | $\underline{\text{abs}}$ <$\mathcal{T}_1$ number>

<$\mathcal{T}_0$ primary> ::= $\underline{\text{long}}$ <$\mathcal{T}_1$ primary>

<$\mathcal{T}_0$ primary> ::= $\underline{\text{short}}$ <$\mathcal{T}_1$ primary>

<$\mathcal{T}$ primary=> ::= <$\mathcal{T}$ variable> | <$\mathcal{T}$ function designator> |
(<$\mathcal{T}$ expression>) | <$\mathcal{T}$ number>

<integer primary> ::= <control identifier>

6.3.2. Semantics

An arithmetic expression is a rule for computing a number.

According to its simple type it is called an integer expression, real expression, long real expression, complex expression, or long complex expression.

6.3.2.1. The operators +, -, *, and / have the conventional meanings of addition, subtraction, multiplication and division. In the relevant syntactic rules of 6.3.1. the symbols $\mathcal{T}_0$, $\mathcal{T}_1$ and $\mathcal{T}_2$ have to be replaced by any combination of words according to the following table which indicates $\mathcal{T}_0$ for any combination of $\mathcal{T}_1$ and $\mathcal{T}_2$.

Operators + | -

$\mathcal{T}_1$ \ $\mathcal{T}_2$	integer	real	complex
integer	integer	real	complex
real	real	real	complex
complex	complex	complex	complex

$\mathcal{T}_0$ has the quality "long" if both $\mathcal{T}_1$ and $\mathcal{T}_2$ have the quality "long", or if one has the quality "long" and the other is "integer".

Operator *

$\mathcal{T}_1$ \ $\mathcal{T}_2$	integer	real	complex
integer	integer	long real	long complex
real	long real	long real	long complex
complex	long complex	long complex	long complex

$\mathcal{T}_1$ or $\mathcal{T}_2$ having the quality "long" does not affect the type of the result.

Operator /

$\mathcal{T}_1$ \ $\mathcal{T}_2$	integer	real	complex
integer	real	real	complex
real	real	real	complex
complex	complex	complex	complex

$\mathcal{T}_0$ has the quality "long" if both $\mathcal{T}_1$ and $\mathcal{T}_2$ have the quality "long", or if one has the quality "long" and the other is "integer" .

6.3.2.2. The operator "-" standing as the first symbol of a simple expression denotes the monadic operation of sign inversion. The type of the result is the type of the operand. The operator "+" standing as the first symbol of a simple expression denotes the monadic operation of identity.

6.3.2.3. The operator div is mathematically defined (for B $\neq$ 0) as

A div B = SGN (A X B) X D (abs A, abs B) (cf. 6.3.2.6.)

where the function procedures SGN and D are declared as

```
integer procedure SGN (integer value A);
     if A < 0 then -1 else 1;
integer procedure D (integer value A, B);
     if A < B then 0 else D(A-B, B) + 1
```

6.3.2.4. The operator rem (remainder) is mathemetically defined as

$$A \text{ rem } B = A - (A \text{ div } B) \times B$$

6.3.2.5. The operator ** denotes exponentiation of the first operand to the power of the second operand. In the relevant syntactic rule of 6.3.1. the symbols $\mathcal{T}_0$ and $\mathcal{T}_1$ are to be replaced by any of the following combinations of words:

$\mathcal{T}_0$	$\mathcal{T}_1$
real	integer
real	real
complex	complex

$\mathcal{T}_0$ has the quality "long" if and only if $\mathcal{T}_1$ does.

6.3.2.6. The monadic operation abs yields the absolute value of the operand. In the relevant syntactic rule of 6.3.1. the symbols $\mathcal{T}_0$ and $\mathcal{T}_1$ have to be replaced by any of the following combinations of words:

$\mathcal{T}_0$	$\mathcal{T}_1$
integer	integer
real	real
real	complex

If $\mathcal{T}_1$ has the quality "long", then so does $\mathcal{T}_0$.

6.3.2.7. Precision of arithmetic. If the result of an arithmetic operation is of simple type <u>real</u>, <u>complex</u>, <u>long</u> <u>real</u>, or long complex then it is the mathematically understood result of the operation performed on operands which may deviate from actual operands.

In the relevant syntactic rules of 6.3.1. the symbols $\mathcal{T}_0$ and $\mathcal{T}_1$ must be replaced by any of the following combinations of words (or word pairs):

Operator <u>long</u>

$\mathcal{T}_0$	$\mathcal{T}_1$
long real	real
long real	integer
long complex	complex

Operator <u>short</u>

$\mathcal{T}_0$	$\mathcal{T}_1$
real	long real
complex	long complex

6.3.3. Examples

```
C + A(I) * B(I)
EXP (-X/(2 * SIGMA)) / SQRT (2 * SIGMA)
```

6.4. <u>Logical Expressions</u>

6.4.1. Syntax

In the following rules for <relation> the symbols $\mathcal{T}_0$ and $\mathcal{T}_1$ must either be identically replaced by any one of the following words:

bit

string

reference

or by any of the words from:

complex

long complex

real

long real

integer

and the symbols $\mathcal{T}_2$ or $\mathcal{T}_3$ must be replaced by any of real, long real, integer.

```
<simple logical expression> ::= <logical element> | <relation>
<logical element> ::=  <logical term> | <logical element2 or
                       <logical term>
<logical term> ::=  <logical factor> | <logical term> and
                    <logical factor>
<logical factor>  ::=  <logical primary> | ¬ <logical primary>
<logical primary>  ::=  <logical value> | <logical variable> |
                        <logical function designator> |
                        (<logical expression>)
<relation>  ::=  <simple T0 expression> <equality operator>
                 <simple T1 expression> | <logical element,
                 <equality operator> <logical element> |
                 <reference expression> is <record class identifier> |
                 <simple T2 expression> <relational operator>
                 <simple T3 expression>
<relational operator>  ::=  < | < = | > = | >
<equality operator> ::= = | ¬ =
```

6.4.2. Semantics

A logical expression is a rule for computing a logical value.

6.4.2.1. The relational operators have their conventional meanings, and yield the logical value true if the relation is satisfied for the values of the two operands; false otherwise. Two references are equal if and only if they are both null or both refer to the same record. Two strings are equal if and only if they have the same length and the same ordered sequence of characters.

6.4.2.2. The operators ¬ (not), and, and or, operating on logical values, are defined by the following equivalences:

```
¬ X        if X then false else true
X and Y if X then Y else false
X or Y     if X then true else Y
```

6.4.3. Examples

```
P or Q
(X < Y) and (Y < Z)
YOUNGESTOFFSPRING (JACK) ¬ = null
FATHER (JILL) is PERSON
```

6.5. Bit Expressions

6.5.1. Syntax

```
<simple bit expression>- ::=  <bit term> | <simple bit expression>
                               or <bit term>
<bit term>  ::=  <bit factor> | <bit term> and <bit factor3
<bit factor>  ::=  <bit secondary> | ¬<bit secondary3
<bit secondary3  ::= <bit primary", | <bit secondary=> shl
                         <integer primary> | <bit secondary> shr
                         <integer primary2
<bit primary>  ::=  <bit sequence | <bit variable> | <bit
                       function designator> | (<bit expression>)
```

6.5.2. Semantics

A bit expression is a rule for computing a bit sequence.

The operators and, or, and ¬ produce a result of type bits, every bit being dependent on the corresponding bits in the operand(s) as follows:

X	Y	¬ X	X and Y	X or Y
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

The operators shl and shr denote the shifting operation to the left and to the right respectively by the number of bit positions indicated by the absolute value of the integer primary. Vacated bit positions to the right or left respectively are assigned the bit value 0.

6.5.3. Examples

G and H or #38

G and ¬ (H or G) shr 8

6.6. String Expressions

6.6.1. Syntax

```
<simple string expression> ::= <string primary>
<string primary> ::= <string> | <string variable | <string
                     function designator> | <string variable>
                     (<integer expression> ▌ <integer number>) |
                     (<string expression>)
```

6.6.2. Semantics

A string expression is a rule for computing a string (sequence of characters).

6.6.2.1. The integer expression preceding the ▮ selects the starting character of the sequence from the string variable specified. The value of the expression indicates the position in the string variable. The value must be greater than or equal to 0 and less than the declared length of the string variable. The first character of the string has position 0. The integer number following the ▮ indicates the length of the selected sequence and is the length of the string expression. The sum of the integer expression and the integer number must be less than or equal. to the declared length of the string variable.

6.6.3. Example

```
string (10) S;
s  (4▮3)
S (I+J ▮1)
string (10) array T (1::m,2::n);
     T (4,6)(3▮5)
```

6.7. Reference Expressions

6.7.1. Syntax

```
<simple reference expressions> ::= <null reference> | <reference
                                   variable> | <reference function
                                   designator> | <record designator> |
                                   (<reference expression>)
```

<record designator> ::= <record class identifier> | <record class identifier> (<expression list>)

<expression list> ::= <T expression> | <expression list>, <T expression>

6.7.2. Semantics

A reference expression is a rule for computing a reference to a record. All simple reference expressions in a reference expression must be of the same record class.

The value of a record designator is the reference to a newly created record belonging to the designated record class. If the record designator contains an expression list, then the values of the expressions are assigned to the fields of the new record. The entries in the expression list are taken in the same order as the fields in the record class declaration, and the simple types of the fields must be assignment, compatible with the simple types of the expressions (cf. 7.2.2.).

6.7.3. Example

PERSON ("CAROL", 0, false, JACK, JILL, null, YOUNGESTOFFSPRING (JACK))

6.8. Precedence of Operators

The syntax of 6.3.1., 6.4.1., and 6.5.1. implies the following hierarchy of operator precedences:

long, short, abs
shl, shr, **
¬
*, /, div, rem, and

+, -, or

<, < =, =, ¬ =, > =, >, is

Example

A = B and C   is equivalent to   A = (B and C)

7. STATEMENTS

A statement denotes a unit of action. By the execution of a statement is meant the performance of this unit of action which may consist of smaller units of action such as the evaluation of expressions or the execution of other statements.

Syntax:

```
<program'>  ::=  <block> .
<statement> ::=  <simple statement> | <iterative statement> |
                 <if statement> | <case statement>
<simple statement>  ::=  <block> | <𝒯 assignment statement> |
                         <empty> | <procedure statement> |
                         <goto statement>
```

7.1. Blocks

7.1.1. Syntax

```
<block>  ::=  <block body> <statement> end
<block body>  ::= <block head> | <block body> <statement>; |
                  <block body> <label definition>
<block head>  ::=  begin | <block head> <declaration> ;
<label  definition> ::= <identifier> :
```

7.1.2. Semantics

Every block introduces a new level of nomenclature. This is realized by execution of the block in the following steps:

Step 1. If an identifier, say A, defined in the block head or in a label definition of the block body is already defined at the place from which the block is entered, then every occurrence of that identifier, A, within the block is systematically replaced by another identifier, say APRIME, which is defined neither within the block nor at the place from which the block is entered.

Step 2. If the declarations of the block contain array bound expressions, then these expressions are evaluated.

Step 3. Execution of the statements contained in the block body begins with the execution of the first statement following the block head.

After execution of the last statement of the block body (unless it is a goto statement) a block exit occurs, and the statement following the entire block is executed.

7.1.3. Example

```
begin real U;
    u := x;   x := Y; Y := z; z := u
end
```

7.2. Assignment Statements

7.2.1. Syntax

In the following rules the symbols $\mathcal{T}_0$ and $\mathcal{T}_1$ must be replaced by words as indicated in Section 1, subject to the restriction that the type $\mathcal{T}_0$ is assignment compatible with the type $\mathcal{T}_1$ as defined in 7.2.2.

$<\mathcal{T}_0$ assignment statement> ::= $<\mathcal{T}_0$ left part> $<\mathcal{T}_1$ expression> |
$<\mathcal{T}_0$ left part> $<\mathcal{T}_1$ assignment statement>

$<\mathcal{T}$ left parts ::= $<\mathcal{T}$ variable> :=

7.2.2. Semantics

The execution of a simple assignment statement

$<\mathcal{T}_0$ assignment statement> ::= $<\mathcal{T}_0$ left part> $<\mathcal{T}_1$ expression>

causes the assignment of the value of the expression to the variable. In a multiple assignment statement

($<\mathcal{T}_0$ assignment statement> ::= $<\mathcal{T}_0$ left part> $<\mathcal{T}_1$ assignment statement>)

the assignments are performed from right to left. The simple type of each left part variable must be assignment compatible with the simple type of the expression or assignment variable immediately to the right.

A simple type $\mathcal{T}_0$ is said to be assignment compatible with a simple type $\mathcal{T}_1$ if either

(1) the two types are identical (except possibly for length specifications), or

(2) $\mathcal{T}_0$ is <u>real</u> or <u>long</u> <u>real</u>, and $\mathcal{T}_1$ is <u>integer</u>, <u>real</u> or long <u>real</u> or

(3) $\mathcal{T}_0$ is <u>complex</u> or <u>long complex</u>, and $\mathcal{T}_1$ is integer, <u>real</u>, long <u>real</u>, <u>complex</u> or <u>long complex</u>.

In the case of a reference, the reference to be assigned must refer to a record of the class specified by the record class identifier associated with the reference variable in its declaration.

7.2.3. Examples

Z := AGE(JACK) := 28

X := Y ÷ <u>abs</u> Z

C := I + X + C

P := X ¬ = Y

7.3. <u>Procedure Statements</u>

7.3.1. Syntax

. <procedure statement9 ::= <procedure identifier> | <procedure identifier9 (<actual parameter list>)

<actual parameter list9 ::= <actual parameter> | <actual parameter list9 , <actual parameter>

<actual parameter> ::= <𝒯 expression9 | <statement9 | <𝒯 subarray designator9 | <procedure identifier> | <𝒯 function identifier>

<𝒯 subarray designator9 ::= <𝒯 array identifier9 | <𝒯 array identifier9 (<subarray designator list>)

Csubarray designator list> ::= <subscript> | * | <subarray designator list>,<subscript> | <subarray designator list>,*

7.3.2. Semantics

The execution of a procedure statement is equivalent to a process performed in the following steps:

Step 1. A copy is made of the body of the proper procedure whose procedure identifier is given by the procedure statement, and of the actual parameters of the latter.

Step 2. If the procedure body is a block, then a systematic change of identifiers in its copy is performed as specified by step 1 of 7.1.2.

Step 3. The copies of the actual parameters are treated in an undefined order as follows: If the copy is an expression different from a variable,, then it is enclosed by a pair of parentheses, or if it is a statement it is enclosed by the symbols begin and end.

step 4. In the copy of the procedure body every occurrence of an identifier identifying a formal parameter is replaced by the copy of the corresponding actual parameter (cf. 7.3.2.1.). In order for the process to be defined, these replacements must lead to correct ALGOL W expressions and statements"

Step 5 The copy of the procedure body, modified as indicated in steps 2 4, is executed.

7.3.2.1. Actual-formal correspondence, The correspondence between the actual parameters and the formal parameters is established as follows: The actual parameter list of the procedure statement (or of the function designator) must have the same number of entries as the formal parameter list of the procedure declaration heading. The correspondence is obtained by taking the entries of these two lists in the same order.

7 3.2.2. Formal specifications. If a formal parameter is specified by value, then the formal type must be assignment compatible with the type of the actual parameter. If it is specified as result, then the type of the actual parameter must be assignment compatible with the

formal type. In all other cases, the types must be identical. If an actual parameter is a statement, then the specification of its corresponding formal parameter must be procedure.

7.3.2.3. Subarray designators. A complete array may be passed to a procedure by specifying the name of the array if the number of subscripts of the actual parameter equals the number of subscripts of the corresponding formal. parameter. If the actual. array parameter has more subscripts than the corresponding formal parameter, enough subscripts must be specified by integer expressions so that the number of *'s appearing in the subarray designator equals the number of subscripts of the corresponding formal parameter. The subscript positions of the formal array designator are matched with the positions with *'s in the subarray designator in the order they appear.

7.3.3. Examples

```
INCREMENT
COPY (A, B, M, N)
INNERPRODUCT  (I, N, A(I,*), B(*,J))
```

7.4. Goto Statements

7.4.1. Syntax

<goto statement> ::= poto <label identifier> | go to <label identifier>

7.4.2. Semantics

An identifier is called a label identifier if it stands as a label.

A goto statement determines that execution of the text be continued after the label definition of the label identifier. The identification of that label definition is accomplished in the following steps:

Step 1. If some label definition within the most recently activated but not yet terminated block contains the label identifier, then this is the designated label definition. Otherwise,

Step 2. The execution of that block is considered as terminated and Step 1 is taken as specified -above.

7.5. If Statements

7.5.1. Syntax

```
<if statement>  ::=  <if clause <statement> | <if clause>
                     <simple statement> else <statement>
<if clause>  ::=  if <logical expression> then
```

7.5.2. Semantics

The execution of if statements causes certain statements to be executed or skipped depending on the values of specified logical expressions. An if statement of the form

<if clause=> <statement>

is executed in the following steps:

Step 1. The logical expression in the if clause is evaluated.

step 2. If the result of Step 1 is true, then the statement following the if clause is executed. Otherwise step 2 causes no action to be taken at all.

An if statement of the form

<if clause> <simple statement> else <statement>

is executed in the following steps:

Step 1. The logical expression in the if clause is evaluated.

Step 2. If the result of step 1 is true, then the simple statement following the if clause is executed. Otherwise the statement following else is executed.

7.5.3. Examples

```
if X = Y then goto L
if X < Y then U := X else if Y < Z then U := Y else V := Z
```

7.6. Case statements

7.6.1. Syntax

```
<case statement> ::= <case clause> begin <statement list> end
<statement list> ::= <statement> | <statement list> ; <statement>;
<case clause> ::= case <integer expression> of
```

7.6.2. Semantics

The execution of a case statement proceeds in the following steps:

Step 1. The expression of the case clause is evaluated.

Step 2. The statement whose ordinal number in the statement list is equal to the value obtained in Step 1 is executed. In order that the case statement be defined, the current value of the expression in the case clause must be the ordinal number of some statement of the statement list.

7.6.3. Examples

```
case I of
begin  X := X + Y;
       Y := y + Z;
       Z := Z + X
end

case j of
begin_E-:(I) := -H(I);
     begin H(I-1) := H(I-1) + H(I); I := I-1 end;
     begin H(I-1) := H(I-1)xH(I); I := I-1 end;
     begin, H(H(I-1)) := H(1); I := I-2 end
end
```

7.7. Iterative Statements

7.7.1. Syntax

```
<iterative statement> ::= <for clause> <statement> | <while
                          clause> <statement>
<for clause> ::= for <control identifier> := <initial value
                 step <increment> until <limit> do | for
                 <identifier> := <initial value> until <limit>
                 do | for <identifier> := <for list> do
<for list> ::= <integer expression> | <for list> , <integer
               expression>
<initial value> ::= <integer expression>
<increment;> : := <integer expression>
<limit> : := <integer expression>
<while clause> : := while <logical expression> do
```

7.7.2. Semantics

The iterative statement serves to express that a statement be executed repeatedly depending. on certain conditions specified by a for clause or a while clause, The statement following the for clause or the while clause always acts as a block, whether it has the form of a block or not. The value of the control identifier cannot be changed by assignment within the controlled statement.

(a) An iterative statement of the form

for <control identifier> := E1 step E2 until E3 do <statement>

is exactly equivalent to the block

begin <statement-D; <statement-D ... ; <statement-D;
... ; <statement-D end

in the I th statement every occurrence of the control identifier is replaced by the reference denotation of the value of the expression E1 + I x E2, enclosed in parentheses.

The index N of the last statement is determined by $N \leq (E3-E1) / E2 < N+1$. If $N < 0$, then it is understood that the sequence is empty. The expressions E1, E2, and E3 are evaluated exactly once, namely before execution of <statement-0>.

(b) An iterative statement of the form

for <control identifier> := E1 until E3 do <statement>

is exactly equivalent to the iterative statement

for <control identifier> := E1 step 1 until E3 do <statement>

(c) An iterative statement of the form

for <control identifier> := E1, E2, ... , EN do <statement>

is exactly equivalent to the block

```
    begin  <statement-D; <statement-B . . . <statement-* ; . . .
    <statement-N> end
```

when in the Ith statement every occurrence of the control identifier is replaced by the reference denotation of the value of the expression EI.

(d) An iterative statement of the form

```
    while E do <statement>
```

is exactly equivalent to

```
    if E then
    begin <statement> ;
         while E do <statement>
    end
```

7.7.3. Examples

```
for V := 1 step 1 until N-1 do S := S + A(U,V)
while (J > 0) and (CITY(J) ¬= S)do J := J-1
for I := X, X + 1, X + 3, X + 7 do  P(I)
```

7.8. Standard Procedures

The standard procedures differ from explicitly declared procedures in that they may have one or more parameters of mixed simple type. In the following descriptions T is to be replaced by any one of

integer	bits
real.	string
long real.	
complex	
long complex	

7.8.1. Read Statements

Implicit declaration heading:

> procedure read ($\mathcal{T}$ result X_1, $\mathcal{T}$ result X_2 . . . , $\mathcal{T}$ result X_n);
>
> procedure readon ($\mathcal{T}$ result X_1, $\mathcal{T}$ result X_2 . . . , $\mathcal{T}$ result X_n);
>
> (where $n \geq 1$)

Both read and readon designate free field read statements. The quantities on the data cards must be speared by one or more blank columns. All 80 card columns can be used and quantities extending to column 80 on one card can be continued beginning in column 1 of the next card. In addition to the numbers of 4.1., numbers of the following sytactic forms are acceptable quantities on the data cards:

1) <sign> <$\mathcal{T}$ number>

   where $\mathcal{T}$ is one of integer, real, long real, complex, long complex.

2) <sign> <$\mathcal{T}_0$ number> <sign> <$\mathcal{T}_1$ number>

   where $\mathcal{T}_0$ is one of integer, real, long real, and $\mathcal{T}_1$ is one of complex,, long complex.

The quantities on the data cards are matched with the variables of the variable list in order of appearance. The simple type of each quantity read must be assignment compatible with the simple type of the variable designated. The read statement begins scanning for the data on the next card. The readon statement begins scanning for the data where the last read or readon statement finished.

7.8.1.2. Examples

```
read (X,A(I))
for I := 1 until N do readon (A(I)
```

7.8.2. Write Statements

Implicit declaration heading:

<u>procedure</u> write ($\mathcal{T}$ <u>value</u> X1, $\mathcal{T}$ <u>value</u> X2, ... , $\mathcal{T}$ <u>value</u> X_n);

(where n $\geq$ 1);

The values of the variables are output in the order they appear in the variable list in a free field form described below. The first field of each WRITE statement begins on a new line. If there is insufficient space remaining on the 132 character print line for a. new field, that line is printed and the new field starts at the beginning of a new print line.

<u>integer</u>: right. justified in field of 14 characters followed by 2 blanks. Field size can be changed by assignment to intfieldsize.

<u>real</u>: same as <u>integer</u> except the field size cannot be changed,

<u>long real</u>: right justified in field of 22 characters followed by 2 blanks.

<u>complex</u>: two adjacent <u>real</u> fields always on the same line.

<u>long complex</u>: two <u>long real</u> fields adjacent always on the same line.

<u>logical</u>: TRUE or FALSE right justified in a field of 6 characters foil-owed by 2 blanks.

<u>string</u>: placed in a field large enough to contain the string and may extend to a new line if the string is larger than 132 characters.

<u>bits</u>: same as <u>real</u>.

<u>reference</u>: same as <u>real</u>.

8. STANDARD FUNCTIONS AND PREDECLARED IDENTIFIERS

8.1. <u>Standard Transfer Functions</u>

Implicit declaration headings:

```
integer procedure round (real value X);
integer procedure truncate (real value X);
integer procedure entier (real value X);
real procedure realpart (complex value X);
long real procedure longrealpart (long complex value X);
real procedure imagpart <complex value X);
long real procedure longimagpart (long complex value X);
complex procedure imag (real value X);
     comment complex number XI;
long complex procedure longimag (long real value X);
logical procedure odd (integer value X);
bits procedure bitstring (integer value X);
     comment binary representation of number X;
integer procedure number (bits value X);
     comment integer with binary representation X;
integer procedure decode (string (1.) value S);
     comment numeric code of the character S;
string (1) procedure code (integer value X);
     comment character whose numeric code is X REM 256;
```

8.2. Standard Functions of Analysis

```
real procedure sin (real value X);
long real procedure longsin (long real value X);
real procedure cos (real value X);
-long real procedure longcos (long real value X);
real procedure arctan (real value X);
     comment -π/2 < arctan (X) < π/2;
long real procedure longarctan (long real value X);
     comment -π/2 < longarctan (X) < π/2;
real procedure ln (real value X);
     comment logarithm base e;
long real procedure longln (long real value X);
     comment logarithm base e;
```

```
real procedure Log (real value X);
     comment Logarithm base 10;
long real procedure longlog (long real value X);
     comment Logarithm base 10;
real procedure exp (real value X);
long real procedure longexp (long real value X);
real procedure sqrt (real value X);
long real procedure longsqrt (long real value X);
complex procedure complexsqrt (complex value X);
     comment principal square root;
long complex procedure longcomplexsqrt (long complex value X);
     comment principal square root;
```

8.3. Overflow and Underflow

8.3.1. Predeclared Variables

```
logical underflow;
     comment initialized to false.  Set to true at occurrence
     of a floating-point underflow interrupt;
overflow;
     comment initialized to false.  Set to true at occurrence
     of a floating-point or fixed-point overflow or divide-by-
     zero interrupt;
```

8.3.2. Standard Message Function

```
integer procedure msglevel (integer value X);
     comment The value of a system integer variable MSG controls
     the number of underflow/overflow messages printed during
     program execution.  MSG is initialized to zero.
          MSG = 0
          No messages are printed
```

MSG > 0

Underflow and overflow messages are printed.
After each message is printed, MSG is decreased by 1.

MSG < 0

Overflow messages are printed. After each message
is printed, MSG is increased by 1.

Each message gives the type of interrupt and a source card number near which the interrupt occured.

Examples

```
OVERFLOW NEAR CARD 0023
UNDERFLOW NEAR CARD 0071
DIV BY ZERO NEAR CARD 0372
```

The predeclared integer procedure msglevel is used to interrogate and to set the value of MSG. The old value of MSG is the value of the procedure msglevel, and the new value given to MSG is the value of the argument of msglevel.

8.4. Output Field Sizes

integer intfieldsize;

comment indicates number of digits including minus sign if any, Initialized to 14; can be changed by assignment state-men-t;

8.5. Time Function

integer procedure time (integer value X);

comment if X = 1, time is returned in 60ths of a second.
If X = 2, time is printed in minutes, seconds and 60ths of a second and returned in 60ths of a second.

UNIT RECORR EQUIPMENT *

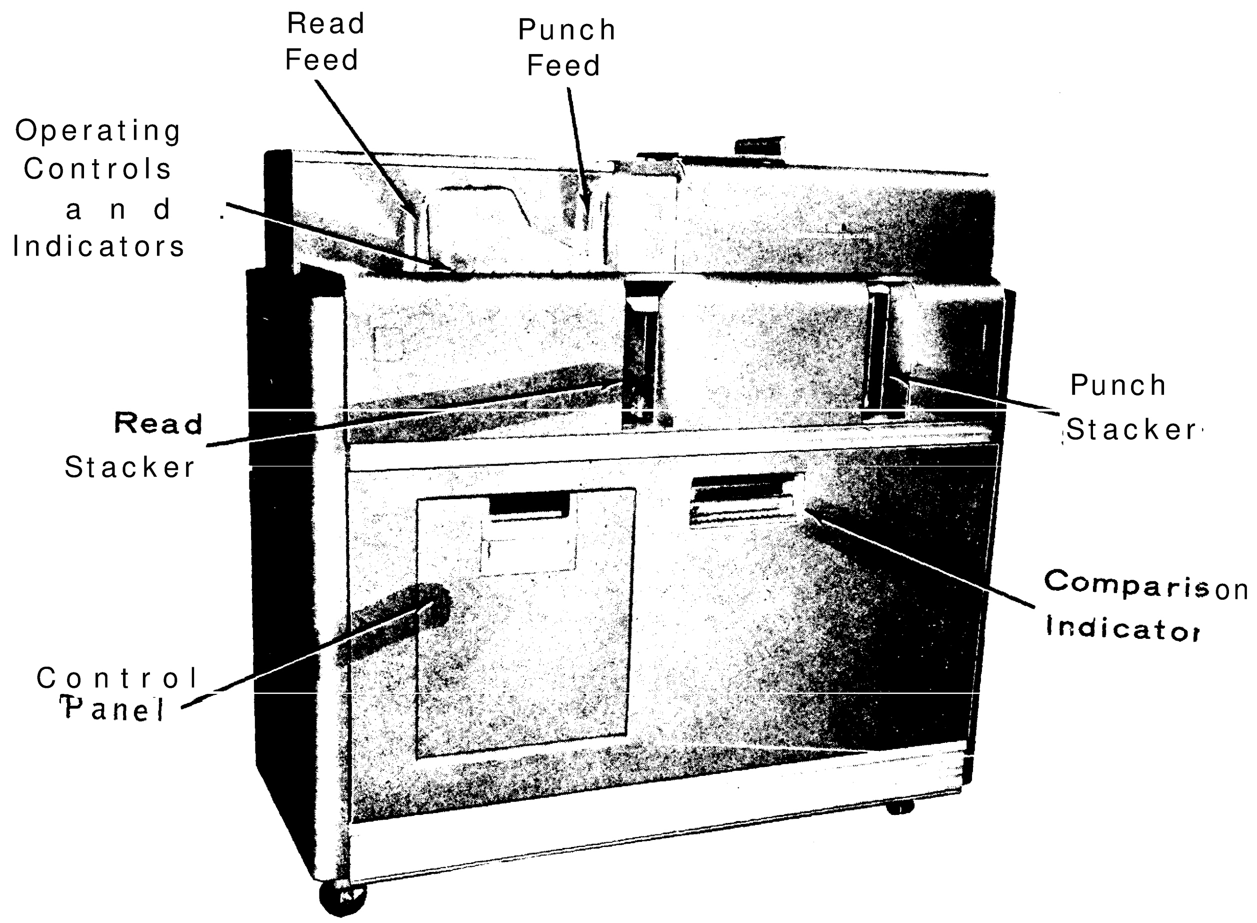
COMPUTATION CENTER
CAMPUS FACILITY
STANFORD UNIVERSITY

* Reprinted from the Campus Facility USERS MANUAL. December 196%

2.2.2 Unit Record Equipment

Necessary unit record equipment is available in Pine Hall, and may be operated by Users to prepare and correct punched cards and list, interpret and duplicate punched card decks. Brief operating instructibns appear below. The personnel in Dispatch will be happy to assist the User in learning how to use and operate the machines. A word of caution -- in the event of a card jam or machine failure, contact a Dispatch clerk immediately and do not attempt to clear the failure or jam.

'1. 519 Reproducing Punch

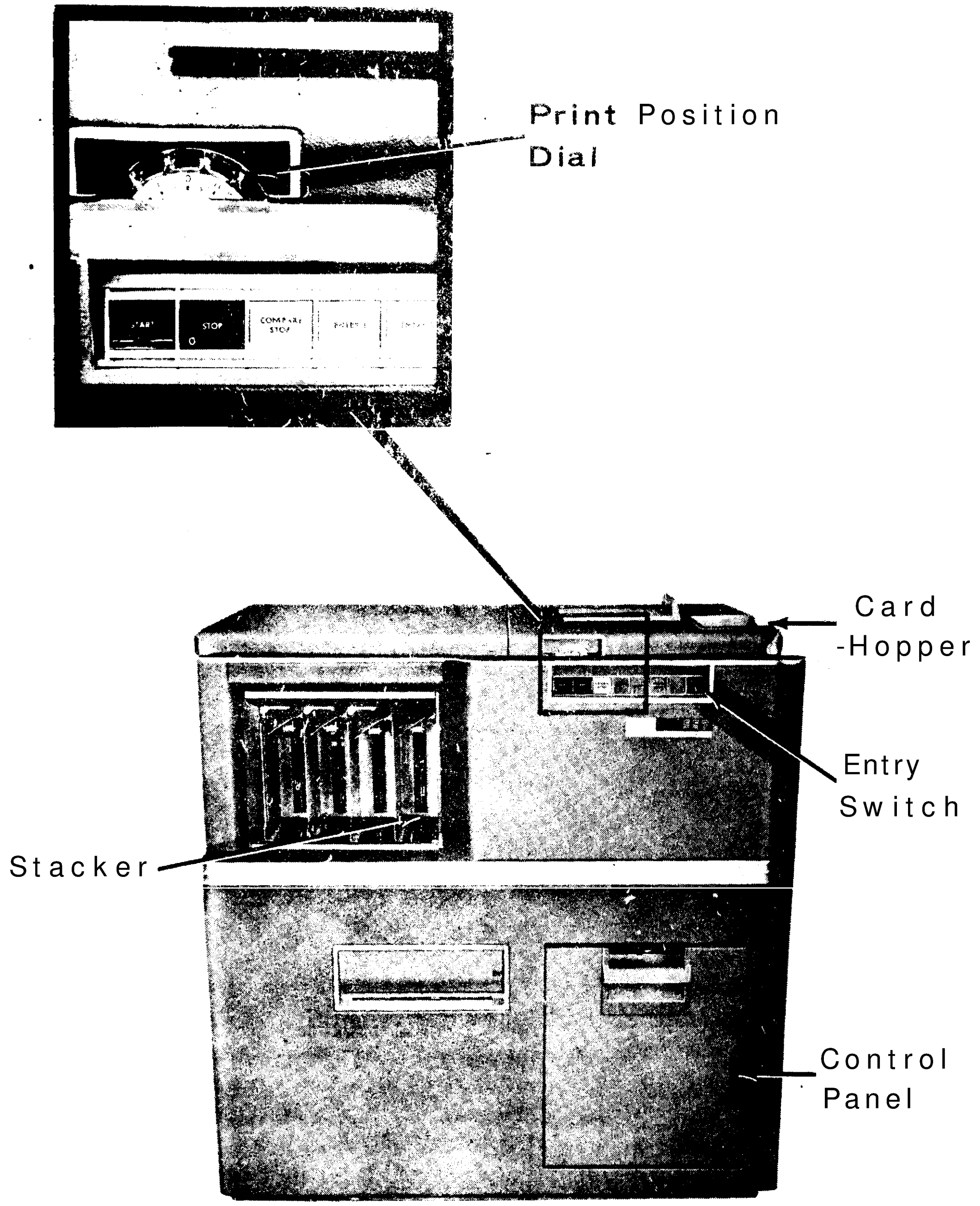


To duplicate a deck, place the source cards into the READ FEED with the top of the cards, face down and toward your right. In the same way, place a supply of blank cards in the PUNCH FEED. Open the CONTROL PANEL cover, insert the 80 X 80 DUPLICATE Control Panel and then reclose the cover. Control panels should be handled with care. Hold. down the START key for a couple of seconds. The cards will begin feeding and will fall into their respective STACKERS. Always stop the machine to replenish the blank card or source card supply. When the last source card has been read, remove the remaining cards from the PUNCH FEED and hold down the START key a few seconds until all cards are in .the STACKERS.

Cards can be duplicated in columns 1-76 and punched with new sequence numbers in columns 77-80. On the Col. 1-76 DUPE and 77-80 NEW SEQ Control Panel select the switch setting desired: count by units or count by 10's. On a blank card, keypunch the starting number you want in your deck, into columns 77-80. Put this card in front of your blank card supply and then load and operate the machine as explained for 80 X 80 duplicating. WARNING: The 519 Reproducing Punch cannot be used to reproduce binary cards.

For comparing, the "Compare" Control Panel is used. The master deck is put in the left-hand feed, and reproduced deck in the right-hand feed. The machine will stop and the red "ERROR" light will glow, if a discrepancy is encountered.

2. <u>557 Interpreter</u>



The interpreter reads information punched into a card and prints it on -the card at the rate of 100 cards per minute. Up to 60 characters can be printed in a single pass through the machine. The remaining 20 characters on the card can be printed on a second pass. Printing can be positioned on the card on any one or' 25 lines. This machine is not yet equipped to interpret all 360/67 code .

Operating Instructions

o Be sure the main power switch on the right-hand end of the machine near the hopper is in the "ON" position, and verify that the proper control board is in the machine.

o Joggle the cards into perfect alignment, and place them face down in the hopper with the 12-edge inward (to the left).

o Set the printing position control (the clear plastic knob with numbers on the edge) to the desired print line. Line No. 1 is above the 12-line on the top edge of the card; line No. 2 is the 12-punch line; line No. 3 is between the 11 and 12 punch lines, etc. The odd-numbered lines (3 through 23) are between the punch lines.

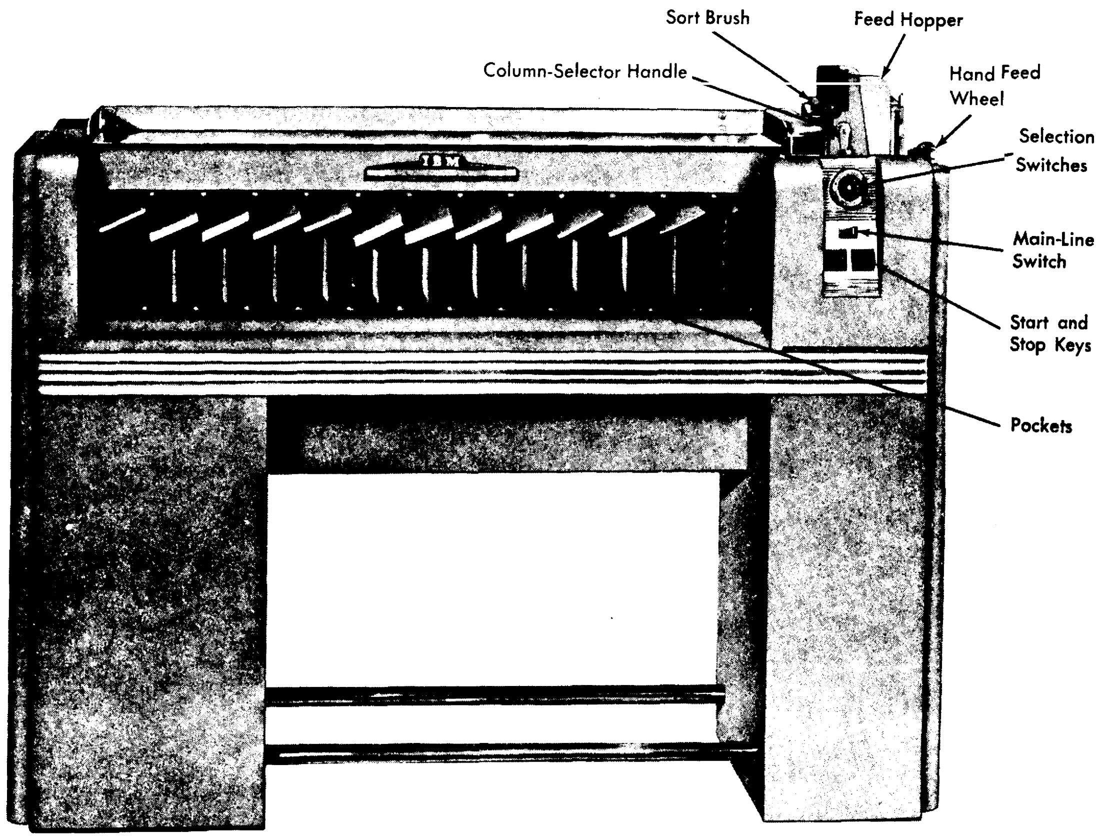
o Set the "ENTRY" toggle switch at the right-hand end of the controls to the "UP" position for entry 1 (the first 60 characters), or the "DOWN" position for entry 2 (the remaining 20 characters), and push the black "START" button.

o The machine will interpret the punches in the cards, which will emerge in their original order in the stacker.

o The machine will stop automatically when the final card has been interpreted, when the stacker is full, if the feed mechanism fails or if the "STOP" button is pushed.

o A special control board is provided for interpreting binary cards.

3. 82 Sorter



The sorter arranges punched cards in either alphabetical or numerical sequence, sorting a single column at a time.

Operating Instructions

- o Be sure the main power supply switch on the right-hand side of the machine is "ON".
- o After a 2-minute warm-up period, press the "START" key to clear the machine of any cards left by the previous User.
- o Joggle the cards into perfect alignment and place them in the hopper at the right-hand end of the machine, then put the card weight on top of the stack. The cards must be face down with the o-edge toward the throat (left).

- Set the control switches as follows:

  a) The "SORTING SUPPRESSION" toggle switch should be set at "OFF".

  b) The "CARD COUNT" toggle switch (black) should be "ON" if a card count is desired, and the counter manually set to zero.

  c) The "COLUMN INDICATOR" (the crank above the selection switches) is set to the card column to be sorted. Sorting is done one column at a time.

  d) The "SELECTION SWITCHES" are set as follows:

NUMERIC SORTING

All tabs set away from center ring	

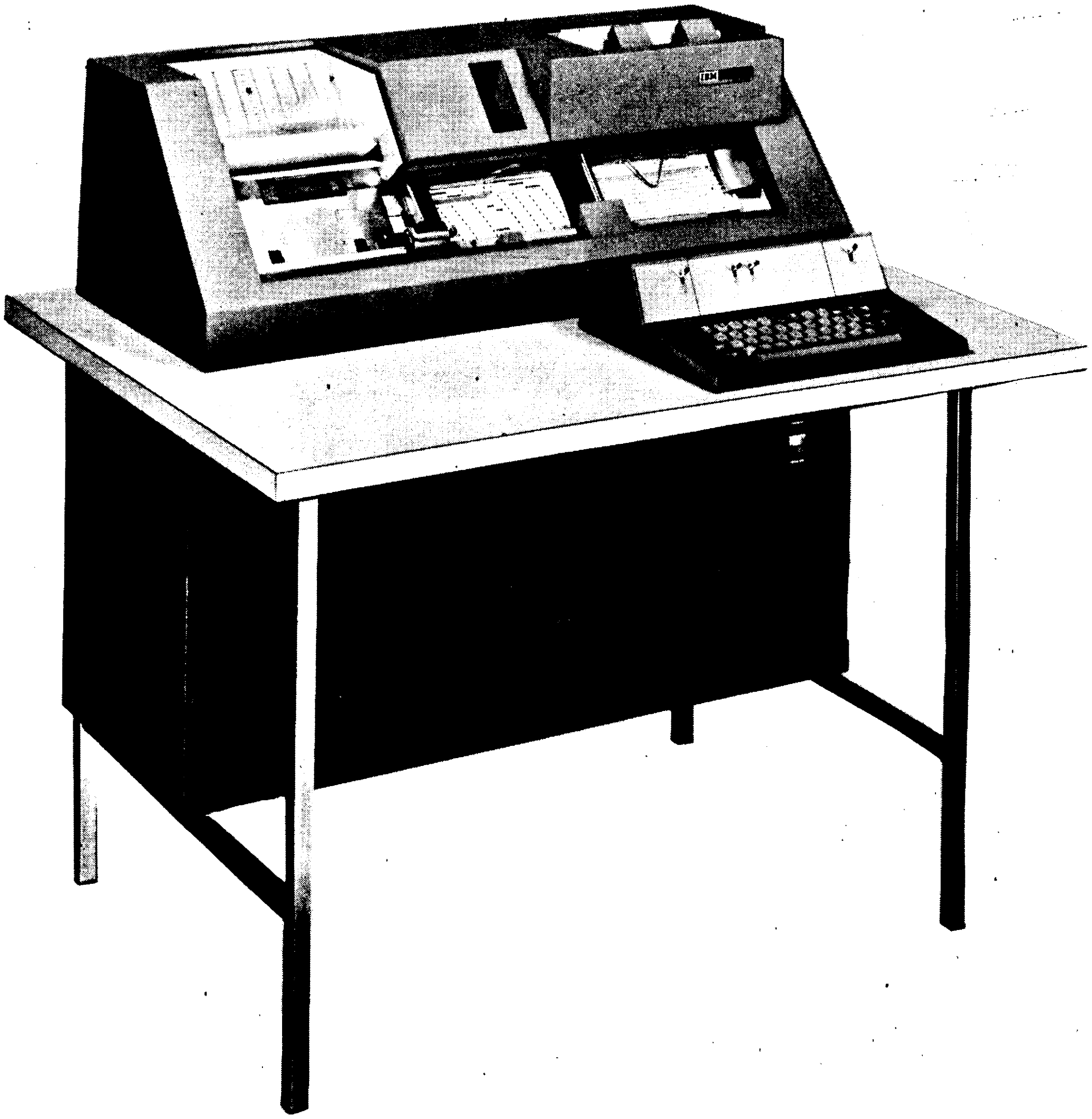
ALPHABETIC SORTING -

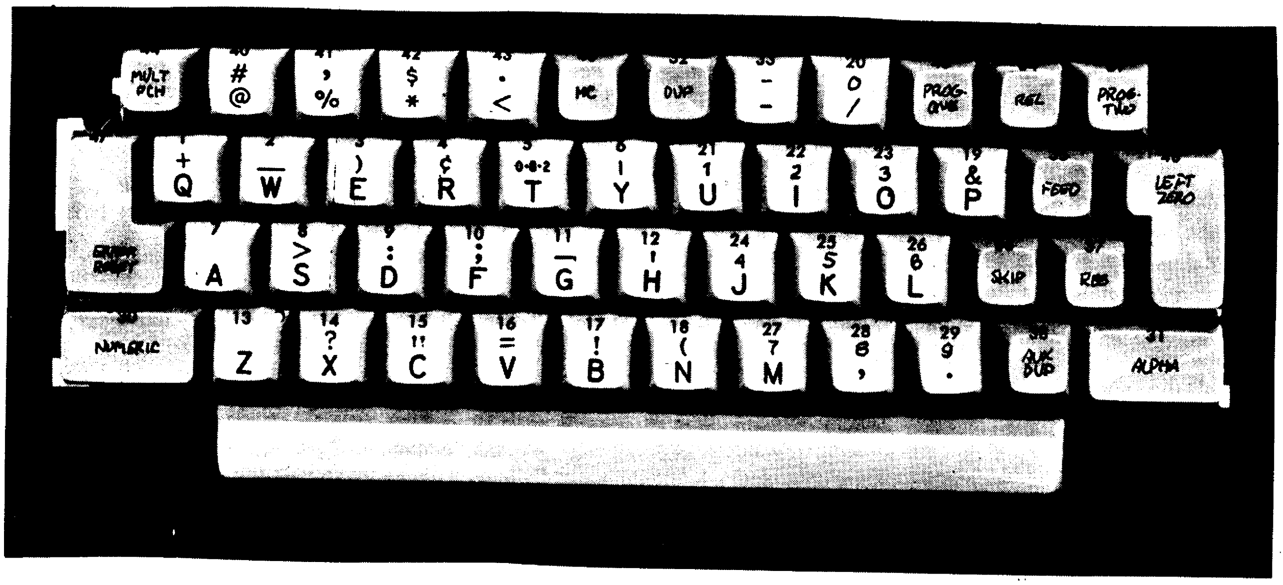
Move red tab to center ring	for sorting out zone (0,11 and 12) punches only. Cards without a zone punch are rejected. The, cards with ietters A-I are put into the 12 pocket. The cards with J through R are put into the 11-pocket, and those with S through Z go into the zero pocket. Cards with numerals and blank go to pocket R.
Move red tab away from center ring	Sort the cards with the letters A-I, J-R, and S-Z separately.

- Press the "START" button until the machine starts feeding cards from the bottom of the stack. Each card passes under the brush head, which determines which of the 13 stacker pockets will accept it, There is a pocket for each punch position in the card, and a reject pocket for cards without a valid punch in the column being sorted.

- The machine will stop when a pocket is full, when the hopper is empty, when the cover over the brush is raised, or when the "STOP" button is pressed.

4. 029 Keypunch





Key Number	ALPHABETIC		NUMERIC	
	Card Code	Graphic	Card Code	Graphic
1	11-8	Q	12-8-6	+
2	0-6	W	0-8-5	_
3	12-5	E	11-8-5	)
4	11-9	R	12-8-2	¢
5	0-3	T	0-8-2	0-8-2
6	0-8	Y	12-8-7	|
7	12-1	A	none	none
8	0-2	S	0-8-6	>
9	12-4	D	8-2	:
10	12-6	F	11-8-6	;
11	12-7	G	11-8-7	¬
12	12-8	H	8-5	'
13	0-9	Z	none	none
14	0-7	X	0-8-7	?
15	12-3	C	8-7	"
16	0-5	V	8-6	=
17	12-2	B	11-8-2	!
19	11-5	N	12-8-5	(
20	11-7 0-1	P	12 0	&
		/		0
21	0-4	U	1	1
22	12-9	I	2	2
23	11-6	O	3	3
24	11-1	J	4	4
25	11-2	K	5	5
27	11-3	L	6	6
28	11-4	M	7	7
	0-8-3	,	8	8
29	12-8-3	.	9	9
33	11	-	11	-
40	8-4	@	8-3	#
41	0-8-4	%	0-8-3	,
42	11-8-4	*	11-8-3	$
43	12-8-4	<	12-8-3	.

Key Graphics and Punched-Hole Codes

o Cards can be punched under "manual" control or " program" control. "Program" controlled punching is advisable when preparing a large number of cards all with a similar format. "Manual" punching is simple and is recommended when a few or randomly formatted cards are to be prepared.

o Manual Control Punching:

a) Put the supply of cards to be punched into the hopper on the upper right-hand side of the machine.

-b) Turn the three switches "AUTO FEED","AUTO SKIP","AUTO DUP", and "PRINT" to the "ON" position..

c) Press the "FEED" key at the right of the keyboard'twice. This will bring down two cards. The first card is ready for punching.

d) If you punch through col. 80 the machine will automatically eject the card punched, position the next card for punching and feed another card.

e) If you want to eject a card before reaching col. 80, manual:,y press the "REL" key.

f) When punching a very few cards, you'can insert cards into the punch station at the right of the machine. Press "REG" and begin, punching,

g) To duplicate a card, put a blank card in the punch station and the source card in the read station (to the left) and then press "REG" . Next, hold down the "DUP" key for contir'ous duplicating, or use the "DUP" key to duplicate column by column. This procedure is commonly used to correct punching errors.

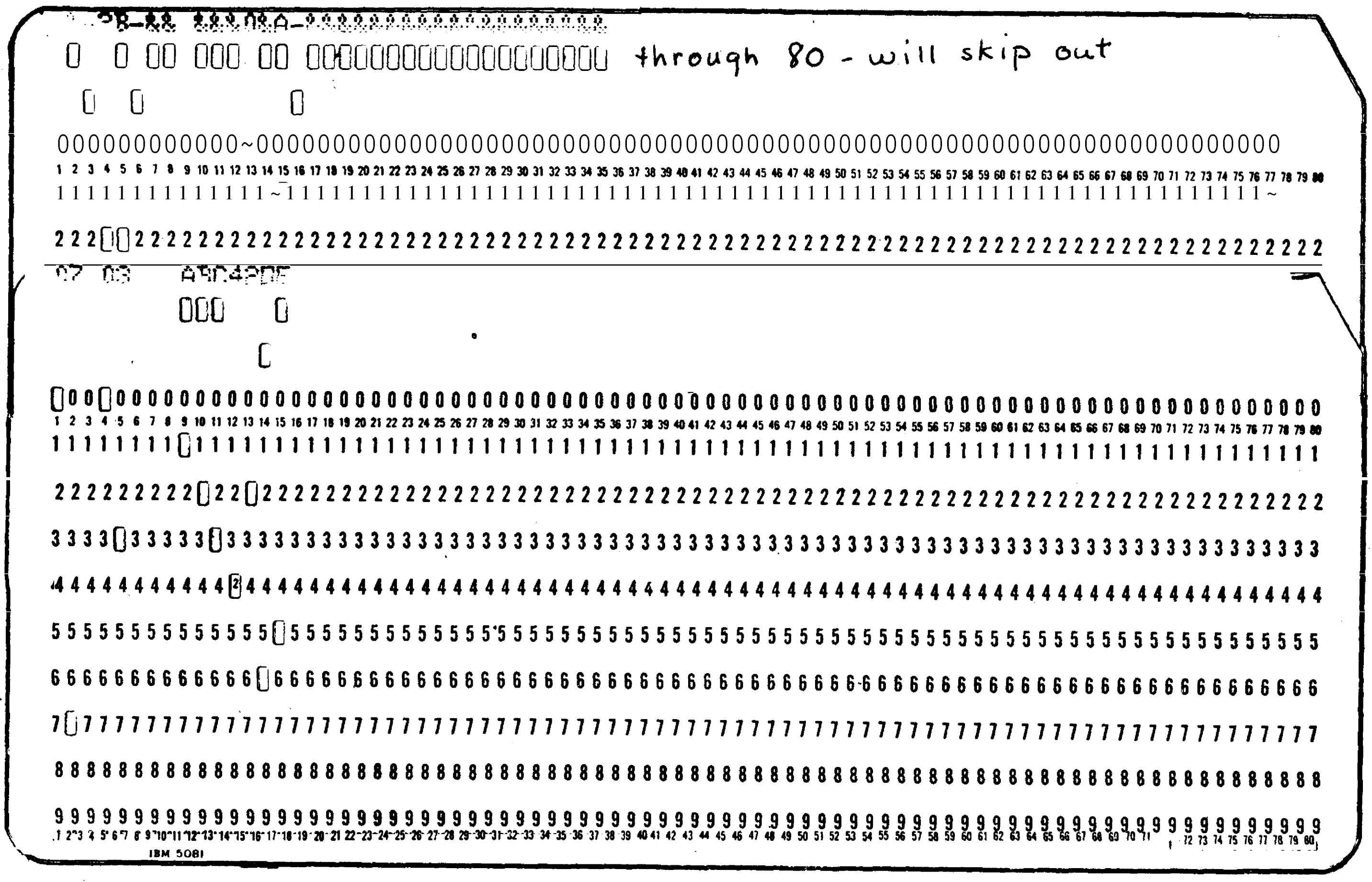
o Program Controlled Punching

a) Preparing a "program" card. In the program control mode, the program card controls the format of the cards and the characters (alphabetic or numeric) to be punched.

1) Program control symbols

1	This punch allows punching of alphabetic characters
b	A blank column allows punching of numeric characters
0	This symbol causes duplicating from the column at the read station to the column at the punch station
	A 2-punch causes printing of leading zeros and all' characters
+	A + symbol in each column in the field, except the first, defines a field.

2) This example of a program card shows all common combinations or codes and their resultant products.



o The keypunch is designed for punching and duplicating only those characters contained on the keyboard. Heavily coded cards (e.g., cards with more than 3 punches per column) <u>cannot be duplicated</u> on the *machines.

o Please remember to consider the other users and clean up the machine before leaving it. Dispose of cards in a nearby "CARD disposal can."

o See the Dispatchers for assistance with keypunch machines and to report failures.

ALGOL W

ERROR MESSAGES

by

Henry R. Bauer
Sheldon Becker
Susan L. Graham

COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY
JANUARY 1968

ALGOL W ERROR MESSAGES

I. PASS ONE MESSAGES

All Pass One messages appear on the first page following the program listing. The message format is

CARD NO. (number) -- (message)

The (number) corresponds to the card number on which the error was found. The (message) is one of those listed below.

INCORRECT SPECIFTN	syntactic entity of a declaration is incorrect, e.g. variable string length.
INCORRECT CONSTANT	syntax error in number or bitstring.
MISSING END	an END needed to close block.
MISSING BEGIN	an attempt to close outer block before end of code.
MISSING)	) is needed.
ILLEGAL CHARACTER	a character, not in a string, is unrecognizable.
MISSING END .	program must conclude with the sequence END .
STRING LENGTH ERROR	string is of 0 length or length greater than 256.
BITS LENGTH ERROR	bits constant denotes no bits or more than 32 bits.
MISSING (	(is needed.
COMPILER TABLE OVERFLOW	terminating error - a compile time table has exceeded its bounds.

TOO MANY ERRORS	the maximum number of errors for Pass One records has been reached. Compilation continues but messages for succeeding errors detected by Pass One are suppressed.
ID LENGTH > 256	more than 256 characters in identifier.

II. PASS TWO MESSAGES

The format of Pass Two error messages is

(message), CARD NUMBER IS (number). CURRENT SYMBOL IS (incoming symbol).

If a $STACK card is included anywhere in the source deck, the SYNTAX ERROR message is followed by

```
STACK CONTAINS:
          (beginning of file)
     <symbol-1>

     <symbol-0 (top of stack)
```

The symbol names may differ somewhat from the metasymbols of the syntax.

If any Pass One or Pass Two errors occur, compilation is terminated at the end of Pass Two.

INCORRECT SIMPLE TYPE <number>	<simple type> of entity is improper as used. Number indicates explanation on list of simple type errors.

INCORRECT TYPE	a variable, label, procedure, record field, record, array, standard function, standard procedure or control identifier is used improperly.
MISMATCHED PARAMETER	formal parameter does not correspond to actual parameter.
MULTIPLY-DEFINED SYMBOL <identifier>	symbol defined more than once in a block.
UNDEFINED SYMBOL <identifier>	symbol is not declared or defined.
INCORRECT NUMBER OF ACTUAL PARAMETERS	the number of actual parameters to a procedure does not equal the number of formal parameters declared for the procedure.
INCORRECT DIMENSION	the array has appeared previously with a different number of dimensions.
DATA AREA EXCEEDED	too many declarations in the block.
INCORRECT NUMBER OF FIELDS	the number of fields specified in a record designator does not equal the number of fields the declaration of the record indicates.
INCOMPATIBLE STRING LENGTH	length of assigned string is greater than length of string assigned to.
INCOMPATIBLE REFERENCES	record class bindings are inconsistent.
BLOCKS NESTED TOO DEEP	blocks are nested more than 8 levels.
REFERENCE MUST REFER TO RECORD CLASS	reference must be bound to a record class.
EXPRESSION MISSING IN PROCEDURE BODY	body of typed procedure must end with an expression.

RESULT PARAMETER MUST BE <T VAR>	the actual parameter corresponding to a result formal parameter must be a <T VARIABLE>.
PROCEDURE BODY LACKS SIMPLE TYPE	proper procedure ends with an expression.
<SYMBOL-1> UNRELATED TO <SYMBOL-a	the symbol at the top of the stack (<SYMBOL-i,) should not be followed by the incoming symbol (<SYMBOL-a).
SYNTAXERROR	construction violates the rules of the grammar. The input string is skipped until the next END, ";", BEGIN, or the end of the program. More than one error message may be generated for a single syntax error.

Simple Type Errors

25. Upper and lower bounds must be integer.
29. Upper and lower bounds must be integer.
32. Simple type of procedure and simple type of expression in procedure body do not agree.
71. Substring index must be integer.
73. Variable before '(' must be string, procedure identifier, or array identifier.
74. Substring length must be integer
76. Field index must be reference or record class identifier.
77. Array subscript must be integer.
81. Array subscript must be integer.
84. Actual parameters and formal parameters do not agree.
88. Actual parameters and formal parameters do not agree.
93. Expressions in if expression do not agree.
94. Expressions in case expression do not agree.
95. Expression in if clause must be logical.

98. Expressions in case expression do not agree.

99. Expression in case clause must be logical.

101. Arguments of= or ¬ = do not agree.

102. Arguments of relational operators must be integer, real, or long real.

103. Argument before is must be reference.

106. Argument of unary + must be arithmetic.

107. Argument of unary - must be arithmetic.

108. Arguments of + must be arithmetic.

109. Arguments of - must be arithmetic.

110. Arguments of or must be both logical or both bits.

112. Record field must be assignment compatible with declaration.

117. Arguments of * must be arithmetic.

118. Arguments of / must be arithmetic.

119. Arguments of div must be integer.

120. Arguments of rem must be integer.

121. Arguments of and must be both logical or both bits.

123. Argument of ¬ must be logical or bits.

125. Exponent or shift quantity must be integer; expression to be shifted must be bits.

126. Shift quantity must be integer; expression to be shifted must be bits.

130. Actual parameter of standard function has incorrect simple type.

134. Argument of long must be integer, real, or complex.

135. Argument of short must be long real or long complex.

136. Argument of abs must be arithmetic.

148. Record field must be assignment compatible with declaration.

181. Expression cannot be assigned to variable.

182. Result of assignment cannot be assigned to variable.

188. -Limit expression in for clause must be integer.

190. Expression in for list must be integer.

191. Assignment to for variable must be integer.

193. Expression in for list must be integer.

[illegible]. Step element must be integer.

[illegible]. Expression in while clause must be logical.

III. PASS THREE ERROR MESSAGES

The form of Pass Three error messages is

```
***** (message)
***** NEAR CARD (number)
```

The number indicates the number of the card near which the error occurred. The message may be

PROGRAM SEGMENT OVERFLOW	the amount of code generated for a procedure exceeds 4096 bytes.
COMPILER STACK OVERFLOW	constructs nested too deeply.
CONSTANT POINTER TABLE TOO LARGE	too many literals appear in a procedure.
BLOCKS NESTED TOO DEEP	parameters in procedure call are nested too deeply; procedure calls in block nested too deeply.
DATA SEGMENT OVERFLOW	too many variables declared in the block.

Iv. RUN TIME ERROR MESSAGES

The form of run error messages is

```
(segment number) (message) RUN ERROR NEAR CARD (number)
```

SUBSTRING INDEXING	substring selected not within named string.
CASE SELECTION INDEXING	index of case statement or case expression is less than 1 or greater than number of cases.
ARRAY SUBSCRIPTING	array subscript not within declared bounds.

LOWER BOUND> UPPER BOUND	lower bound is greater than upper bound in array declaration.
ARRAY TOO LARGE	array must have fewer elements:
ASSIGNMENT TO NAME PARAMETER	assignment to a formal name parameter whose corresponding actual parameter is an expression, a literal, control identifier, or procedure name.
DATA AREA OVERFLOW	storage available for program execution has been exceeded.
ACTUAL-FORMAL PARAMETER MISMATCH INFORMAL PROCEDURE CALL	the number of actual parameters in a formal procedure call is different from the number of formal parameters in the called procedure,: or the parameters are not assignment compatible.
RECORD STORAGE AREA OVERFLOW	no more storage exists for records.
LENGTH OF STRING INPUT	string read is not assignment compatible with corresponding declared string.
LOGICAL INPUT	quantity corresponding to logical quantity is not true or false.
NUMERICAL INPUT	numerical input not assignment compatible with specified quantity.
REFERENCE INPUT	reference quantities cannot be read.
READER EOF	a system control card has been encountered during a read request.
REFERENCE	the null reference has been used to address a record, or a reference bound to two or more record classes was used to address a record class to which it was not currently pointing.