ALGOL W

REFERENCE MANUAL

JUNE 1972

PREFACE

This manual describes the ALGOL W language and the compiler constructed for the IBM 360 at Stanford University under the direction of Niklaus Wirth. The language is based upon "A Contribution to the Development of ALGOL" by Niklaus Wirth and C.A.R. Hoare. The compiler was written by Henry R. Bauer, Sheldon Becker, Susan L. Graham and Edwin H. Satterthwaite who also documented the system.

Subsequently a number of minor amendments and several extensions have been made to the language; substantial changes have been made to the compiler to improve its efficiency and to add to its capabilities. In particular, a debugging system has been added which is a significant improvement on the programming tools normally provided by compilers. Many of these changes, the work of Edwin Satterthwaite, have been described in the revised documentation of the language and compiler prepared by Richard L. Sites (Stanford University Technical Report STAN-CS-71-230, "ALGOL W REFERENCE MANUAL") others have been described in NUMAC Programming Notes 39 and 41. A few recent additions are documented here for the first time. In preparing this edition of the manual all of these sources have been used freely.

The manual consists of two distinct parts. In the first part of the manual, sections one to eight define the ALGOL W language. Sections nine to eleven form the "Programmer's Guide to ALGOL W". Section nine describes the compiler, sections ten and eleven deal with aspects of the operating systems, MTS and OS/360 respectively, which are relevant to the use of the ALGOL W compiler. The second part of the manual is a transcription of "Introduction to ALGOL W Programming" by Henry R. Bauer. Amendments have been made here, to reflect changes to the language and to simplify its transcription to machine readable form. The author's permission to make these changes is gratefully acknowledged.

This edition of the ALGOL W manual supercedes the 1970 edition of the NUMAC ALGOL W manual and replaces NUMAC Programming Notes 27, 39 and 41. Changes since the previous manual are summarised below.

1) The introduction of three new basic symbols, assert, algol and fortran, providing a new statement, the assert statement (cf.7.8) and the ability to invoke externally defined procedures (cf.5.3.2.4).
2) The use of the character "_" as a character in identifiers (cf.3.1).
3) Changes to the precision of arithmetic; products (other than of integer quantities) have the quality "long".
4) The precedence of operators has been changed (cf.6, 6.4). This obviates the need for the intuitively unnecessary parentheses in conditions involving relational and logical operators, but implies changed interpretation of bit and logical expressions involving these operators.
5) Block expressions are no longer restricted to defining function procedure bodies but are permitted in any expression (cf.6).

6) In comparing strings of equal lengths the shorter is (effectively) extended with blanks to the length of the longer before comparison. String assignments are done in a single action rather than character by character left to right, removing the anomalous behaviour on assigning strings to substrings of themselves (cf.6.4,7.2)
7) Facilities for creating formatted output have been added to the WRITE and WRITEON standard procedures (cf.7.9) using additional predeclared variables.
8) An additional exceptional condition, ENDFILE, is detected on input.
9) Sections 9, 10, 11 and Appendix II are new.
10) The standard functions COMPLEXSQRT and LONGCOMPLEXSQRT have been deleted.

Except in the case of the completely new sections (9, 10, 11, and Appendix II) changes (in content as opposed to layout or presentation) since the previous version of the manual are marked with vertical lines in the left margin.

The manual describes the 01JULY72 version of the compiler.

June, 1972. J. Eve

LANGUAGE DESCRIPTION

ALGOL W

LANGUAGE DESCRIPTION

1 TERMINOLOGY, NOTATION AND BASIC DEFINATIONS

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

(1) VT, the set of basic (or terminal) symbols of the language,
(2) VN, the set of syntactic entities (or nonterminal symbols),and
(3) P, the set of syntactic rules (or productions).

1.1 Notation

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

<a> ::= x

where <a> is a member of VN, and x is any possible sequence of basic symbols and syntactic entities, simply to be called a "sequence". In ALGOL W, the set P contains the syntactic rule

<bar> ::= |

implying that | is a basic symbol of the language. Adopting the convention that all references to this basic symbol in other syntactic rules shall be replaced by <bar> permits the unambiguous use subsequently of the notation

<a> ::= x | y | ... | z

as an abbreviation for the set of syntactic rules

<a> ::= x
<a> ::= y
.....
<a> ::= z

In the syntactic rule

<empty> ::=

the sequence Contains zero symbols, i.e. the empty sequence.

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 VN, x = u<a>w, y = uvw, and <a> ::= v is a rule in 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 VT, and x can be produced from the syntactic entity <program>.

The sets VT and VN - {|} are defined Through enumeration of their members in Section 2 (cf. also 4.4). The syntactic rules are given throughout sections 1 to 8. To provide explanations for the meaning of ALGOL W programs, lower case letter sequences used in syntactic entities have been chosen to be English words describing approximately the nature of the 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 T or Tn, where n is a digit, 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 T within one syntactic rule must be replaced consistently, and the replacing words are

integer	logical
real	bit
long-real	string
complex	reference
long-complex	

For example, the production

<T-expression-1> ::= <T-expression-2> (cf.6)

corresponds to

<integer-expression-1> ::= <integer-expression-2>
<real-expression-1> ::= <real-expression-2>
<long-real-expression-1> ::= <long-real-expression-2>
<complex-expression-1> ::= <complex-expression-2>
<long-complex-expression-1> ::= <long-complex-expression-2>

The production

<T4-expression-8> ::= long <T5-expression-8> (cf.6.3.1 and 6.3.2.7)

corresponds to

<long-real-expression-8> ::= long <real-expression-8>
<long-real-expression-8> ::= long <integer-expression-8>
<long-complex-expression-8> ::= long <complex-expression-8>

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 (VN-{11})

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 | 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 | assert | algol | fortran

All underlined words, which are called "reserved words", are represented by the same words in capital letters in an actual program, with no intervening blanks. Adjacent reserved words, identifiers (cf.3.1) and numbers (cf.4.1) must include no blanks and 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 (VT)

(with corresponding section numbers)

3 IDENTIFIERS

3.1 Syntax

```
<identifier> ::= <letter> | <identifier> <letter> |
                 <identifier> <digit> | <identifier> _
<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 for 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 (cf. 7.9 and 8) 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, as is a block expression (cf.6).

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 Examples

```
I
PERSON
ELDERSIBLING
X15, X20, X25
NEW_PAGE
```

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:

integer: the value is a 32 bit integer,
real: the value is a 32 bit floating point number,
long real: the value is a 64 bit floating point number,
complex: the value is a complex number composed of two numbers of type real,
long 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 least one and 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 a an ordered set of values, all of identical type,
record: the value is an ordered set of 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.6.2.2).

Subsequently, the reference denotation of constants is defined. The reference denotation of any constant consists of a sequence of characters. This, however, 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-constant> ::= <complex-constant>L
<complex-constant> ::= <imaginary-constant>
<imaginary-constant> ::= <real-constant>I |
        <integer-constant>I
<long-real-constant> ::= <real-constant>L |
        <integer-constant>L
<real-constant> ::= <unscaled-real> |
        <unscaled-real><scale-factor> |
        <integer-constant><scale-factor> |
        <scale-factor>
```

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

4.1.2 Semantics

Arithmetic constants are numbers 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 preceding it. Each number has a uniquely defined type. (Note that all <T-constant>s are unsigned.)

4.1.3 Examples

```
  1              .5                 1I
  0100         1'3                  0.671
3.1416         6.02486'+23          1IL
2.7182818284590 45L                 2.3'-6
```

4.2 Logical Values

4.2.1 Syntax

```
<logical-constant> ::=  true  |  false
```

4.3 Bit Sequences

4.3.1 Syntax

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

Note that 2 | ... | F correspond to 2^{10} | ... | 15^{10} .

4.3.2 Semantics

The number of bits in a bit constant is 32 or 8 hex digits. The bit constant is always represented by a 32 bit word with the specified sequence of bits 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-constant> ::= <string>
<string> ::= "<open-string>"
<open-string> ::= <character> | <open-string><character>
```

4.4.2 Semantics

Strings consist of any sequence of (at least one and 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. The characters accepted by the IBM System 360 are listed in Appendix I.

4.4.3 Examples

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

4.5 References

4.5.1 Syntax

<reference-constant> ::= 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-T-variable-declaration> |
<T-array-declaration> | <procedure-declaration> |
<record-class-declaration>

5.1 Simple Variable Declarations

5.1.1 Syntax

<simple-T-variable-declaration> ::= <T-type><identifier-list>
<integer-type> ::= integer
<real-type> ::= real
<long-real-type> ::= long real
<complex-type> ::= complex
<long-complex-type> ::= long complex
<logical-type> ::= logical
<bits-type> ::= bits | bits (32)
<string-type> ::= string | string (<integer-constant>)
<reference-type> ::=
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 (cf. 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. The value of this integer may not be less than 1 or greater than 256. If the simple type is given only as string, the length of the variable is 16 characters.

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> ::= <T-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 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. The bound pair expressions can depend only on variables and procedures global to the block in which the declaration occurs. If, for any bound pair, the value of the upper bound is less than the value of the lower bound, the array has no elements.

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>
<TO-function-procedure-declaration> ::=
```

```
            <T0-type> procedure <procedure-heading>;
            <T1-function-procedure-body>
    <proper-procedure-body> ::= <statement> |
            <external-reference>
    <T-function-procedure-body> ::=  <T-expression> |
            <external-reference>
    <procedure-heading> ::= <identifier> |
            <identifier> (<formal-parameter-list>)
    <formal-parameter-list> ::= <formal-parameter-segment> |
            <formal-parameter-list>;<formal-parameter-segment>
    <formal-parameter-segment> ::= <formal-array-parameter> |
            <formal-type><identifier-list>
    <formal-type> ::= <T-type> | <T-type> value |
            <T-type> result | <T-type> value result |
            <T-type>  procedure | procedure
    <formal-array-parameter> ::= <T-type>  array
            <identifier-list> (<dimension-specification>)
    <dimension-specification> ::= * |
            <dimension-specification> , *
|   <external-reference> ::=  fortran <string> |  algol <string>
```

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. The type of a function procedure body, T1, must be assignment compatible (cf.7.2.2) with the type, T0, of the procedure.

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 value and result 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 begin and end;

(2) For every formal parameter whose formal type contains the symbol value or result (or both),
    (a) a declaration followed by a semicolon is inserted after the first begin of the procedure body, with a 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 in the outermost block 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 the 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.

5.3.2.4 External references. Use of an external reference as a procedure body indicates that the actual procedure body is specified by the environment in which the program is to be executed. The information in the external reference is used to locate and interpret that procedure body. The details of such use depend upon the specific environment. (cf. 9.6, 10.3 and 11.3)

5.3.3 Examples

```
procedure INCREMENT; X := X+1

real procedure MAX (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 := N step -1 until 0 do S := S * X + A(I);
          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
```

```
procedure PLOTSUBROUTINE (integer value I); fortran "PLOTSB"
```

5.4 Record Class Declarations

5.4.1 Syntax

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

```
record NODE (reference (NODE) LEFT, RIGHT)

record PERSON (string NAME; integer AGE; logical MALE;
     reference (PERSON) FATHER, MOTHER, YOUNGESTOFFSPRING,
     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. The operands are either constants, variables or function designators, or other expressions, enclosed by parentheses if necessary. 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. Expressions are distinguished by a type and a precedence level, the former depending on the types of the operands and the latter resulting from the precedence hierarchy imposed upon operators in the syntactic rules which follow. The syntactic entities naming different kinds of expression in these rules display these attributes, the word "expression" being prefixed by a type and, usually, postfixed by an integer indicating the precedence level. (Higher precedence is implied by increasing magnitude of this integer). The operators and their precedence levels are:

level	operators
1	or
2	and
3	¬
4	< <= = ¬= >= > is
5	+ -
6	* / div rem
7	shl shr **
8	long short abs

When the types allow an operator at level i to be applied to operands, the resulting expression, which belongs to the syntactic class <T-expression-i>, has the intuitive meaning given in the second column of the table.

Syntactic Entity	Meaning	Definitions
<T-expression-1>	disjunction	6, 6.4, 6.5
<T-expression-2>	conjunction	6, 6.4, 6.5
<T-expression-3>	negation	6, 6.4, 6.5
<T-expression-4>	relation	6, 6.4
<T-expression-5>	sum	6, 6.3
<T-expression-6>	term	6, 6.3
<T-expression-7>	factor	6, 6.3, 6.5
<T-expression-8>	primary	6, 6.3, 6.7

The third column of the table indicates sections where definitions of these syntactic entities occur.

Throughout section 6 and its subsections the symbol T has to be replaced consistently as described in Section 1 and the triplets T0, T1, T2 have to be either all three replaced by the same one of the words

logical
bit
string
reference

or (subject to specification to the contrary) in accordance with the following "triplet rules".

1. Given the qualities (integer, real or complex) of T1 and T2, the corresponding quality of T0 is given in the table

T1 \ T2	integer	real	complex
integer	integer	real	complex
real	real	real	complex
complex	complex	complex	complex

2. T0 has the quality "long" either if both T1 and T2 have that quality, or if one has the quality "long" and the other is "integer".

Syntax:

<T-expression> ::= <T-expression-1> |
    <conditional-T-expression>
<T-expression-1> ::= <T-expression-2>
<T-expression-2> ::= <T-expression-3>
<T-expression-3> ::= <T-expression-4>
<T-expression-4> ::= <T-expression-5>
<T-expression-5> ::= <T-expression-6>
<T-expression-6> ::= <T-expression-7>
<T-expression-7> ::= <T-expression-8>
<T-expression-8> ::= <T-variable> |
    <T-function-designator> | <T-constant> |
    (<T-expression>) | <T-block-expression>
<T-block-expression> ::= <block body><T-expression> end

Semantics:

There are 8 levels of precedence; an expression at one level of precedence is a valid expression at each lower level of precedence.

A block expression introduces a new level of nomenclature and specifies the execution of a sequence of statements in the block body as described for blocks (cf.7.1). After execution of the block body, the final expression is evaluated and the value of that expression becomes the value of the entire block expression.

Variables, function designators and conditional expressions are defined in subsequent paragraphs of section 6.

6.1 Variables

6.1.1 Syntax

<simple-T-variable> ::= <T-variable-identifier> |

```
            <T-field-designator> | <T-array-designator>
    <T-variable> ::= <simple-T-variable>
    <string-variable> ::= <substring-designator>
    <T-field-designator> ::= <T-field-identifier>
            (<reference-expression>)
    <T-array-designator> ::= <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 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+K,I-J)
FATHER(JACK)        MOTHER(FATHER(JILL))
```

6.2 Function Designators

6.2.1 Syntax

```
    <T-function-designator> ::= <T-function-identifier> |
            <T-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, modified as indicated in steps 2-4, is executed. Execution of the expression which constitutes or is part of the modified procedure body consists of evaluation of that expression, and the resulting value is the value of the function designator. The type of the function designator is the type in the corresponding function procedure declaration.

6.2.3 Examples

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

6.3 Arithmetic Expressions

6.3.1 Syntax

```
<T3-expression-5> ::= + <T3-expression-6> |
          - <T3-expression-6>
<T0-expression-5> ::= <T1-expression-5> + <T2-expression-6> |
          <T1-expression-5> - <T2-expression-6>
<T0-expression-6> ::= <T1-expression-6> * <T2-expression-7> |
          <T1-expression-6> / <T2-expression-7>
<integer-expression-6> ::=
          <integer-expression-6> div <integer-expression-7> |
          <integer-expression-6> rem <integer-expression-7>
<T4-expression-7> ::=
          <T5-expression-7> ** <integer-expression-8>
<T4-expression-8> ::= abs <T5-expression-8> |
          long <T5-expression-8> | short <T5-expression-8>
<integer-expression-8> ::= <control-identifier>
```

6.3.2 Semantics

An arithmetic expression is a rule for computing a number. According to its 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.

For the operator *, the second "triplet rule" is modified so that T0 has the quality long unless both T1 and T2 are integer.

For the operator /, the "triplet rules" apply except when both T1 and T2 are integer, then T0 is long-real.

6.3.2.2 The operator "-" standing as the first symbol of an expression at priority level 5 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 such an expression denotes the monadic operation of identity.

In the relevant syntactic rules of 6.3.1, every occurrence of the symbol T3 must be systematically replaced by one of the following words (or word pairs):

integer
real
long-real
complex
long-complex

6.3.2.3 The operator div is defined (for B ¬= 0) as

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

where the fuction procedures SGN and D are declared as

```
integer procdure 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 defined as

A rem B = A - (A div B) * 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 T4 and T5 are to be replaced by any of the following combinations of words:

T4	T5
long-real	integer
long-real	real
long-complex	complex

T4 has the quality "long" whether or not T5 does.

6.3.2.6 The monadic operator abs yields the absolute value or modulus of the operand. In the relevant syntactic rule of 6.3.1 the symbols T4 and T5 have to be replaced by any of the following combinations of words:

T4	T5
integer	integer
real	real
real	complex

If T5 has the quality "long", then so does T4.

6.3.2.7 Precision of arithmetic. If the result of an arithmetic operation is of type real, complex, long real, or long complex, its value is defined by System/360 arithmetic and 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 T4 and T5 must be replaced by any of the following combinations of words (or word pairs):

Operator long

T4	T5
long-real	real
long-real	integer
long-complex	complex

Operator short

T4	T5
real	long-real
complex	long-complex

Note: It is illegal to apply long to an expression which is already long; similarly for short.

6.3.3 Examples

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

6.4 Logical Expressions

6.4.1 Syntax

In the following rules for <relation> the symbols T6 and T7 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 T8 or T9 must be identically replaced by string or must be replaced by any of real, long-real, integer.

```
<logical-expression-1> ::=
    <logical-expression-1> or <logical-expression-2>
<logical-expression-2> ::=
    <logical-expression-2> and <logical-expression-3>
<logical-expression-3> ::= ¬ <logical-expression-4>
<logical-expression-4> ::= <relation>
<relation> ::=
    <T6-expression-5><equality-operator><T7-expression-5> |
    <T8-expression-5><inequality-operator><T9-expression-5> |
    <reference-expression-5> is <record-class-identifier>
<equality-operator> ::= = | ¬=
<inequality-operator> ::= < | <= | >= | >
```

6.4.2 Semantics

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

6.4.2.1 The relational operators represent algebraic ordering for arithmetic arguments and EBCDIC ordering for string

arguments. If two strings of unequal length are compared, the shorter string is considered to be extended to the length of the longer (for the comparison only) by appending blanks to the right. The relational operators 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. The operator is yields the logical value true if the reference expression designates a record of the indicated record class; false otherwise. The reference value null fails to designate a record of any record class.

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

```
<bit-expression-1> ::=
          <bit-expression-1> or <bit-expression-2>
<bit-expression-2> ::=
          <bit-expression-2> and <bit-expression-3>
<bit-expression-3> ::= ¬ <bit-expression-4>
<bit-expression-7> ::=
          <bit-expression-7> shl <integer-expression-8> |
          <bit-expression-7> shr <integer-expression-8>
```

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 bit(s) 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 operand.

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

```
<substring-designator> ::= <string-variable>
          (<integer-expression><bar><integer-constant>)
```

6.6.2 Semantics

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

6.6.2.1 A substring designator denotes a sequence of characters of the string designated by the string variable. The integer expression preceding the bar selects the starting character of the sequence. 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 bar 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 Examples

```
S(4|3)
S(I+J|1)
STREET(J+1)(I|1)
NAME(FATHER(JACK))(0|8)
```

6.7 Reference Expressions

6.7.1 Syntax

```
<reference-expression-8> ::= <record-designator>
<record-designator> ::= <record-class-identifier> |
<record-class-identifier> (<expression-list>)
<expression-list> ::= <empty> | <T-expression> |
          <expression-list>, |
          <expression-list> , <T-expression>
```

6.7.2 Semantics

A reference expression is a rule for computing a reference to a record.

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 length of

the list must equal the number of fields specified in the record class declaration. Values of nonempty expressions in the expression list are assigned to the corresponding fields of the new record, and the types of the expressions must be assignment compatible with the types of the record fields (cf. 7.2.2).

6.7.3 Examples

```
PERSON ("JANE", 0, false, JACK, JILL, null, YOUNGESTOFFSPRING
   (JACK))
NODE ( , null)
```

6.8 Conditional Expressions

6.8.1 Syntax

```
<conditional-T-expression> ::=
          <case-clause> (<T-expression-list>)
<conditional-T0-expression> ::=
          <if-clause> <T1-expression> else <T2-expression>
<T-expression-list> ::= <T-expression>
<T0-expression-list> ::=
          <T1-expression-list> , <T2-expression>
<if-clause> ::= if <logical-expression> then
<case-clause> ::= case <integer-expression> of
```

6.8.2 Semantics

The construction

```
<if-clause> <T1-expression> else <T2-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 true, the expression following the if clause is selected; if the value is false, the expression following else is selected. If T1 and T2 are type string, the length of the resulting expression is the maximum of the lengths of the component string expressions; if necessary, blanks are appended on the right of the shorter string. The construction

```
<case-clause> (<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. If T is type string, the length of the resulting string expression is the maximum of the lengths of the strings in the expression list. If necessary, the length of any shorter element is increased by appending blanks on the right.

6.8.3 Examples

```
if A>B then A else B
case I of ("SPADES","HEARTS","DIAMONDS","CLUBS")
```

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> ::= <statement>. |
              <proper-procedure-declaration>. |
              <T-function-procedure-declaration>.
<statement> ::= <simple-statement> | <iterative-statement> |
              <if-statement> | <case-statement>
<simple-statement> ::= <block> | <T-assignment-statement> |
              <procedure-statement> | <goto-statement> |
              <standard-procedure-statement> |
              <assert-statement> | <empty>
```

Programs which are procedure declarations cannot be executed directly, but the corresponding procedure bodies can form part of the environment in which other ALGOL W programs are executed (cf. 5.3, 2.4, 9.6, 10.3 and 11.3)

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 except for occurrence in array bound expressions is systematically replaced by another identifier, say A', 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 (unless it is a goto statement leading to a label within the block) 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 T0 and T1 must be replaced by words which may be substituted for T as indicated in Section 1, subject to the restriction that the type T1 must be assignment compatible with the type T0 as defined in 7.2.2.

```
<T0-assignment-statement> ::= <T0-left-part><T1-expression> |
            <T0-left-part><T1-assignment-statement>
<T-left-part> ::= <T-variable> :=
```

7.2.2 Semantics

The execution of a simple assignment statement

<T0-left-part><T1-expression>

causes the assignment of the value of the expression to the variable. If a shorter string is to be assigned to a longer one, the shorter string is first extended to the right with blanks until the lengths are equal. In a multiple assignment statement

<T0-left-part> <T1-assignment-statement>

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

A type T1 is said to be assignment compatible with a type T0 if either

(1) the two types are identical (except that if T0 and T1 are string, the length of the T0 variable must be greater than or equal to the length of the T1 expression or assignment), or

(2) T0 is real or long real, and T1 is integer, real or long real or

(3) T0 is complex or long complex, and T1 is integer, real, long real, complex or long complex.

In the case of a reference, the reference to be assigned must be null or refer to a record of one of the classes specified by the record class identifiers associated with the reference variable in its declaration.

7.2.3 Examples

```
Z := AGE(JACK) := 28          C := I + X + C
X := Y + abs Z                P := X ¬= Y
```

7.3 Procedure Statements

7.3.1 Syntax

```
<procedure-statement> ::= <procedure-identifier> |
          <procedure-identifier>(<actual-parameter-list>)
<actual-parameter-list> ::= <actual-parameter> |
          <actual-parameter-list> , <actual-parameter>
<actual-parameter> ::= <T-expression> | <statement> |
          <T-subarray-designator> | <procedure-identifier> |
          <T-function-identifier>
<T-subarray-designator> ::= <T-array-identifier> |
          <T-array-identifier>(<subarray-designator-list>)
<subarray-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. The procedure statement is replaced by the copy of the procedure body.

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. In each subarray designator, any subscripts are evaluated and replaced by constants designating the resulting values.

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 The following table summarises the forms of actual parameters which may be substituted for each kind of formal parameter specification.

Formal type	Actual parameter
<T-type>	<T-expression>
<T0-type> value	<T1-expression>
<T1-type> result	<T0-variable>
<T1-type> value result	<T2-variable>
<T-type> procedure	<T-function-identifier> <T-expression>
procedure	<procedure-identifier> <statement>
<T-type> array	<T-subarray-designator>

The type T1 must be assignment compatible with the type T0. The types T1 and T2 must be mutually assignment compatible.

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(IP, N, A(I,*), B(*,J))
```

7.4 Goto Statments

7.4.1 Syntax

```
<goto-statement> ::= goto <label-identifier> |
                     go to <label-identifier>
```

7.4.2 Semantics

An identifier is 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.

Note: There is only one definition of a valid label. (cf. 7.1.2)

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

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 H(I) := -H(I);
     begin H(I-1) := H(I-1) + H(I); I := I-1 end;
     begin H(I-1) := H(I-1) * H(I); I := I-1 end;
     begin H(H(I-1)) := H(I); I := I-2 end
end
```

7.7 Iterative Statements

7.7.1 Syntax

```
<iterative-statement> := <for-clause><statement> |
       <while-clause><statement>
<for-clause> ::= for <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 (the identifier following for) cannot be changed by assignment within the controlled statement.

(a) An iterative statement of the form

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

has the same effect as the block

begin <statement-0>; <statement-1> ... ; <statement-I>;
... ; <statement-N> end

where, in the Ith statement, every occurrence of the control identifier is replaced by the value of the expression (E1 + I*E2). 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>. Therefore they cannot depend on the control identifier.

(b) An iterative statement of the form

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

is exactly equivalent to the iterative statement

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

(c) An iterative statement of the form

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

is exactly equivalent to the block

begin <statement-1>; <statement-2> ... <statement-I> ;
... <statement-N> end

where, in the Ith statement, every occurrence of the control identifier is replaced by the expression EI, enclosed by a pair of parentheses.

(d) An iterative statement of the form

while E do <statement>

is exactly equivalent to

```
   begin
L:      if E then
           begin <statement> ; goto L end
   end
```

where it is understood that L represents an identifier which is not defined at the place from which the while statement is entered.

7.7.3 Examples

```
for V := 0 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 Assert Statements

7.8.1 Syntax

```
<assert statement> ::= assert <logical expression>
```

7.8.2 Semantics

The execution of an assert statement causes the logical expression to be evaluated. If the value is false, execution of the program is terminated.

7.9 Standard Procedures

Standard procedures are provided in ALGOL W for the purpose of communication with the input/output system. A standard procedure differs from an explicitly declared procedure in that the number and type of its actual parameters need not be identical in every statement which invokes the standard procedure.

Syntax:

```
<standard-procedure-statement> ::=
          READ (<input-parameter-list>) |
          READON (<input-parameter-list>) |
          READCARD (<input-parameter-list>) |
          WRITE (<transput-parameter-list>) |
          WRITEON (<transput-parameter-list>) |
          IOCONTROL(<transput-parameter-list>)
<input-parameter-list> ::= <T-variable> |
          <simple-statement> |
          <input-parameter-list> , <T-variable> |
          <input-parameter-list> , <simple-statement>
<transput-parameter-list> ::= <T-expression> |
          <simple-statement> |
          <transput-parameter-list> , <T-expression> |
          <transput-parameter-list> , <simple-statement>
```

7.9.1 The Input/Output System

ALGOL W provides a single legible input stream and a single legible output stream. These streams are conceived as sequences of records, each record consisting of a character sequence of fixed length. The input stream has the logical properties of a sequence of cards in a card reader; records consist of 80 characters. The output stream has the logical properties of a sequence of lines on a line printer; records consist of 132 characters, and the records are grouped into logical pages. Each page consists of not less than one nor more than 60 lines.

Input records may be transmitted as strings without analysis. Alternatively, it is possible to invoke a procedure which will scan the sequence of records for data items to be interpreted as numbers, bit sequences, strings, or logical values. If such analysis is specified, data items may be

reference denotations of the corresponding constants (cf. Section 4). In addition, the following forms of arithmetic expressions are acceptable data items, and the corresponding types are those determined by the rules for expressions (cf. 6.3):

(1) <sign><T-constant>
where : T is one of integer, real, long real, complex, long complex;

(2) <T0-constant><sign><T1-constant> |
<sign><T0-constant><sign><T1-constant>
where : T0 is one of integer, real, long real, and
T1 is one of complex, long complex.

Data items are separated by one or more blanks. Scanning for data items initially begins with the first character of the input stream; after the initial scan, it normally begins with the character following the one which terminated the most recent previous scan. Leading blanks are ignored. The scan is terminated by the first blank following the data item. In the process, new records are fetched as necessary; character position 80 of one record is considered to be immediately followed by character position 1 of the next record. There exist procedures to cause the scanning process to begin with the first character of a record; if scanning would not otherwise start there, a new record is fetched.

Output items are assembled into records by an editing procedure. Items are automatically converted to character sequences and placed in fields as described below. The first field transmitted begins the output stream; thereafter, each field is normally placed immediately following the most recent previously transmitted field. If, however, the field corresponding to an item cannot be placed entirely within a non-empty record, that item is made the first field of the next record. In addition, there exist procedures to cause the field corresponding to an item to begin a new record. Each page group is automatically terminated after 60 records; procedures are provided for causing earlier termination.

7.9.2 Read Statements

Both READ and READON designate free field input procedures. Input records are scanned as described in 7.9.1. Values on input records are read, matched with the variables of the actual parameter list in order of appearance, and assigned to the corresponding variables. The type of each data item must be assignment compatible with the type of the corresponding variable. For each READ statement, scanning for the first data item is caused to begin with the first character of a record; for a READON statement, scanning continues from the previous point of termination as determined by prior use of READ, READON, or IOCONTROL (cf. 7.9.1).

READCARD designates a procedure transmitting 80 character input records without analysis. For each variable of the actual parameter list, the scanning process is set to begin at the first

character of a record (by fetching a new record if necessary), all 80 characters of that record are assigned to the corresponding string variable, and subsequent input scanning is set to begin at the first character of the next sequential record.

7.9.3 Write Statements

WRITE and WRITEON designate output procedures with automatic format conversion. Values of expressions in the transput parameter list (there must be at least one) are converted to character fields which are assembled into output records in order of appearance (cf. 7.9.1). For each WRITE statement, the field corresponding to the first value is caused to begin an output record; for a WRITEON statement, assembly continues from the previous point of termination.

The values of a set of predeclared editing variables control the field widths and the formats of numerical quantities printed by the standard Algol W output routines. These variables are initialized to appropiate default settings; their values can be inspected and modified in the course of the execution of an Algol W program. Their attributes are given by the following table:

Identifier	Type	Initial Value	Interpretation
I_W	integer	14	width of integer fields
R_FORMAT	string(1)	"F"	format of real, long real, complex, and long complex fields
R_W	integer	14	width of real and long real fields; width of complex and long complex fields (2*R_W + 2)
R_D	integer	0	places following the decimal point in real, long real, complex, and long complex fields
S_W	integer	2	width of the fields of blanks appended to the end of each field (excluding string fields).

Values of I_W and R_W control the output field widths used for numerical quantities, in conjunction with the values of S_W they determine the layout of each line of numerical output. Integer quantities are converted according to a standard format, but three different formats for the legible representations of real, long real, complex, and long complex values (strictly, rounded approximations to these values) are available. For a particular output value, the actual format is determined by interrogation of the variable R_FORMAT, which must specify one of the following:

(1) scaled format (R_FORMAT = "S"), in which the legible representation takes the form of a normalized mantissa followed by an explicit scale factor;

(2) aligned format (R_FORMAT = "A"), in which the

representation includes an integral part, a fractional part with a specified number of digits, but no scale factor;

(3) free-point format (R_FORMAT = "F"), in which the representation is chosen to use a specified number of significant digits, with the decimal point suitably positioned and with a scale factor only if necessary.

Scaled and aligned representations are sometimes said to use "scientific" and "fixed-point" notation respectively. If scaled or free-point format is specified, the number of significant digits printed is given by R_W - 7. If (but only if) aligned format is specified, the number of digits following the decimal point is controlled by the value of R_D, and the magnitude of the numerical quantity determines the number of significant digits printed.

The field in which an output item is placed depends upon the type of the item, as follows:

Type	Field-Description
integer	right justified in a field of I_W characters and followed by S_W blanks
real	right justified in a field of R_W characters and followed by S_W blanks
long real	right justified in a field of R_W characters and followed by S_W blanks
complex	right justifed in a field of (2*R_W+2) characters and followed by S_W blanks
long complex	right justified in a field of (2*R_W+2) characters and followed by S_W blanks
logical	right justified in a field of 6 characters and followed by S_W blanks
string	field length is exactly the length of the string
bits	right justified in a field of 14 characters and followed by S_W blanks

Parameters corresponding to the syntactic class <simple-statement>; are executed as they are encountered in the corresponding output lists; they cause no values to be transmitted but can (and normally should) serve to change the values of the editing variables or the state of the input/output system. Furthermore, the values of the five predeclared editing variables I_W, R_W, R_D, R_FORMAT and S_W are automatically saved at the beginning of execution of WRITE or WRITEON statements and restored at the end. Thus changes to the values of these variables within an output statement are localized and can affect only the editing of the remaining elements of that list, but assignments outside of such a list can affect all subsequent editing.

7.9.4 Control Statements

IOCONTROL designates a procedure which affects the state of the input/output system. Argument values with defined effect are listed below; other values currently have no effect but are explicitly made available for local use or future expansion.

Value	Action (cf. 7.9.1)
1	Subsequent input scanning begins with the first character of a record.
2	Subsequent output assembly begins with the first field of a record.
3	Subsequent output assembly begins with the first field of a record which, in turn, begins a new output page.
4	Subsequent output has no provision for automatic page skips.
5	Subsequent output contains carriage control characters providing automatic page skips. (Initial Option).

7.9.5 Examples

```
READ ( X, A(1) )
READCARD ( S, LINE(10|80) )
WRITE ( "AVERAGE =", SUM/N )
WRITEON ( X(1,J) )
IOCONTROL (2)
```

Execution of the program,

```
begin
  procedure SCALED (integer value N);
    begin R_FORMAT := "S"; R_W := N+7
    end;
  procedure ALIGNED (integer value N,D);
    begin R_FORMAT := "A"; R_W:= N+D+1; R_D:= D
    end;
  procedure FREE_POINT(integer value N);
    begin R_FORMAT := "F";R_W := N+7
    end;
  procedure NEW_LINE; IOCONTROL(2);

  FREE_POINT(5);I_W :=2; S_W := 1;

  for I:= -1, 0, 32 do
    begin  WRITE(S_W := 0, I,":", NEW_LINE,I/3);
    WRITEON("I  ",ALIGNED(3,2),I/3,"*",SCALED(12),I/3,"*")
    end
end.
```

will produce the following output lines:

```
-1:
    -0.33333I   -0.33 * -3.33333333333'-01 *
 0:
          0I    0.00 *                0     *
32:
      10.667I   10.67 *  1.06666666667'+01 *
```

Note that the setting of S_W when the corresponding quantity is transmitted determines the number of trailing blanks; also, edited values are always rounded.

Any values assigned to I_W, R_W or S_W in excess of 132 are treated as 132. In the event that values of I_W, R_W, R_D, S_D or R_FORMAT are erroneous or inconsistent with the magnitude or precision of the number to be transmitted, then alternative values are used. These values ensure that an approximation to the number is always transmitted and that not more digits than are warranted by the precision of the number are transmitted.

8 STANDARD FUNCTIONS AND PREDECLARED IDENTIFIERS

The ALGOL W environment includes declarations and initialization of certain procedures and variables which supplement the language facilities previously described. Such declarations and initialization are considered to be included in a block which encloses each ALGOL W program (with the terminating period eliminated). The corresponding identifiers are said to be predeclared.

8.1 Standard Transfer Functions

Certain functions For conversion of values from one type to another are provided. These functions are predeclared; the corresponding implicit declaration headings are listed below:

```
integer procedure TRUNCATE (real value X);
     comment the integer i such that
          |i| <= |X| < |i| + 1 and i*X > = 0
integer procedure ENTIER (real value X);
     comment the integer i such that
          i <= X < i + 1 ;
integer procedure ROUND (real value X);
     comment the value of the integer expression
          if X < 0 then TRUNCATE(X-0.5) else
          TRUNCATE(X+0.5) ;
integer procedure EXPONENT (real value X);
     comment 0 if X = 0,  otherwise the largest
     integer i such that
          i <= (log(|X|)/log16)+1
     This function obtains the exponent used in the S/360
     representation of the real number;
real procedure ROUNDTOREAL (long real value X);
     comment the properly rounded value of X ;
real procedure REALPART (complex value Z);
     comment the real component of Z ;
long real procedure LONGREALPART (long complex value Z);
real procedure IMAGPART (complex value Z);
     comment the imaginary component of Z ;
long real procedure LONGIMAGPART (long complex value Z);
complex procedure IMAG (real value X);
     comment the complex number 0 + Xi ;
long complex procedure LONGIMAG (long real value X);
logical procedure ODD (integer value N);
     comment the logical value
          N rem 2 = 1 ;
bits procedure BITSTRING (integer value N);
     comment two's complement representation of N ;
integer procedure NUMBER (bits value X);
     comment integer with two's complement representation X ;
integer procedure DECODE (string(1) value S);
     comment numeric code for the character S
     (cf. Appendix 1) ;
string(1) procedure CODE (integer value N);
     comment character with numeric code
     (cf. Appendix 1) given by
          abs (N rem 256) ;
```

In the following comments, the significance of characters in the prototype formats is as follows:

D	decimal digit in a mantissa or integer
E	decimal digit in an exponent
A	hexadecimal digit in a mantissa or integer
B	hexadecimal digit in an exponent
±	sign (blank for positive mantissa or integer)
ü	blank

Each exponent is unbiased. Decimal exponents represent powers of 10; hexadecimal exponents represent powers of 16. Each mantissa (except 0) Represents a normalized fraction less than one. Leading zeroes are not suppressed.

```
string(12) procedure BASE10 (real value X);
     comment string encoding of X with format
               ü±EE±DDDDDDD ;
string(12) procedure BASE16 (real value X);
     comment string encoding of X with format
               üü±BB±AAAAAA ;
string(20) procedure LONGBASE10 (long real value X);
     comment string encoding of X with format
               ü±EE±DDDDDDDDDDDDDDD ;
string(20) procedure LONGBASE16 (long real value X);
     comment string encoding of X with format
               üü±BB±AAAAAAAAAAAAAA ;
string(12) procedure INTBASE10 (integer value N);
     comment string encoding of N with format
               ü±DDDDDDDDDD ;
string(12) procedure INTBASE16 (integer value N);
     comment unsigned, two's complement string encoding
     of N with format
               üüüüAAAAAAAA ;
```

8.2 Standard Function of Analysis

The following functions of analysis are provided in the system environment. In some cases, they are partial functions; action for arguments outside of the allowed domain is described in 8.5. These functions are predeclared; the corresponding implicit declaration headings are listed below:

```
real procedure SQRT (real value X);
     comment the positive square root of X,
       domain : X >= 0 ;
long real procedure LONGSQRT (long real value X);
     comment the positive square root of X,
       domain : X >= 0 ;
real procedure EXP (real value X);
     comment e ** X ,
       domain : X < 174.67 ;
long real procedure LONGEXP (long real value X);
     comment e ** X ,
       domain : X < 174.67 ;
real procedure LN (real value X);
     comment logarithm of X to the base e,
       domain : X > 0 ;
```

```
long real procedure LONGLN (long real value X);
     comment logarithm of X to the base e, domain : X > 0 ;
real procedure LOG (real value X);
     comment logarithm of X to the base 10, domain : X > 0 ;
long real procedure LONGLOG (long real value X);
     comment logarithm of X to the base 10, domain : X > 0 ;
real procedure SIN (real value X);
     comment sine of X (radians),
       domain : -823550 < X < 823550 ;
long real procedure LONGSIN (long real value X);
     comment sine of X (radians),
       domain : -3.537'+15 < X < 3.537'+15 ;
real procedure COS (real value X);
     comment cosine of X (radians)
       domain : -823550 < X < 823550 ;
long real procedure LONGCOS (long real value X);
     comment cosine of X (radians),
       domain : -3.537'+15 < X <3.537'+15 ;
real procedure ARCTAN (real value X);
     comment arctangent (radians) of X,
       range : -PI/2 <  LONGARCTAN(X) < PI/2 ;
long real procedure LONGARCTAN (long real value X);
     comment arctangent (radians) of X,
       range : -PI/2 < ARCTAN(X) < PI/2 ;
```

8.3 Time Function

The ALGOL W environment includes a clock which measures elapsed time since the beginning of program execution. The resolution of that clock is at least 1/60 second and at most 1/38400 second. A predeclared function is provided for reading the clock

```
integer procedure TIME (integer value N);
     comment Argument                    Result Units

               -time of day              -
            -1                           seconds/60

               - elapsed execution time -
             0                           minutes/100
             1                           seconds/60
             2                           seconds/38400
```

The result for any other argument is not defined;

8.4 Predeclared Variables

The following variables are to be considered declared and initialized by assignment in the conceptual block enclosing The entire ALGOL W program. The values indicated for real and long real quantities are to be understood as decimal approximations to the actual machine-format values provided.

```
integer I_W;
     comment initialized to 14 , controls output field size
       for integers (cf. 7.9.1);
```

```
integer R_W;
     comment initialised to 14, controls output field size
       for real, long real, complex and
       long complex quantities (cf.  7.9.1);
integer R_D;
     comment initialised to 0, specifies the number of
       fraction digits in aligned formats (cf.7.9.1);
string(1) R_FORMAT;
     comment initialised to "F", controls output format
       for real, long real, complex and long complex
       quantities (cf.  7.9.1);
integer S_W;
     comment initialised to 2, specifies the number of
       blanks appended to the end of an output numeric field
       (cf.  7.9.1);
integer MAXINTEGER;
     comment initialized to 2147483647, the maximum
       positive integer allowed by the implementation;
real EPISILON;
     comment initialized to 9.536743'-07 ,
       the largest positive real number e provided by the
       implementation such that
                    1 + e = 1 ;
long real LONGEPSILON;
     comment initialized to 2.22044604925031'-16L ,
       the largest positive long real number e provided by
       the implementation such that
                    1 + e = 1 ;
long real MAXREAL;
     comment initialized to 7.23700557733226'+75L ,
       the largest positive long real number provided by
       the implementation;
long real PI;
     comment initialized to 3.14159265358979L ;
```

8.5 Exceptional Conditions

The facilities described below are provided in ALGOL W to allow detection and control of certain exceptional conditions arising in the evaluation of arithmetic expressions and standard functions.

Implicit declarations:

```
record EXCEPTION (logical XCPNOTED;
     integer XCPLIMIT, XCPACTION;
     logical XCPMARK; string(64) XCPMSG);
reference(EXCEPTION)
     ENDFILE,
     OVFL, UNFL, DIVZERO, INTOVFL, INTDIVZERO,
     SQRTERR, EXPERR, LNLOGERR, SINCOSERR;
```

Associated with each exceptional condition which can be processed is a predeclared reference variable to which references to records of the class EXCEPTION can be assigned. Fields of such records control the processing of exceptions. The association between conditions and reference variables is as follows:

Reference Variable	Conditions
ENDFILE	end of file detected on input
OVFL	real, long real, complex, long complex (exponent) overflow
UNFL	real, long real, complex, long complex (exponent) underflow
DIVZERO	real, long real, complex, long complex division by zero
INTOVFL	integer overflow
INTDIVZERO	integer division by zero
SQRTERR	negative argument for SQRT, LONGSQRT
EXPERR	argument of EXP, LONGEXP out of domain (cf. 8.2)
LNLOGERR	argument of LN, LOG, LONGLN, LONGLOG out of domain (cf. 8.2)
SINCOSERR	argument of SIN, COS, LONGSIN, LONGCOS out of domain (cf.8.2)

When one of the conditions listed above is detected, the corresponding reference variable is interrogated, and one of the alternatives described below is chosen.

If the value of the reference variable interrogated is null, the condition is ignored and execution of the ALGOL W program continues. In such situations, a value of 0 is returned as the value of a standard function or input operation. For other conditions the result is that provided by the underlying hardware (cf. IBM System/360 Principles of Operation, IBM Systems Library, Form A22-6821). In determining such a result, it is to be noted that in those cases in which the detection of exceptional conditions can be inhibited at the hardware level, namely integer overflow and exponent underflow, detection is so inhibited when the corresponding reference is null.

If the value of the reference variable interrogated is not null, the fields of the record designated by that reference are interrogated, and the processing action is that described by the algorithm given below in the form of an extended ALGOL W procedure. Identifiers in lower case represent quantities which transcend the ALGOL W language; they are explained subsequently

```
procedure PROCESSEXCEPTION
(reference(EXCEPTION) value CONDITION);
  begin
     XCPNOTED(CONDITION) := true;
     XCPLIMIT(CONDITION) := XCPLIMIT(CONDITION) - 1;
     if (XCPLIMIT(CONDITION) < 0) or XCPMARK(CONDITION) then
       WRITE("*** ERROR NEAR COORDINATE nnnn - ",
```

```
        XCPMSG(CONDITION));
      if XCPLIMIT(CONDITION) < 0 then endexecution else
      if specialcondition then
      resultant := default else
      resultant := if XCPACTION(CONDITION) = 1
                   then adjustment else
                   if XCPACTION(CONDITION) = 2
                   then OL else
                   default
  end PROCESSEXCEPTION
```

This procedure is invoked with the value of the reference variable appropriate to the condition as actual parameter. The significance of the special identifiers used is as follows:

nnnn — approximate coordinate number of the source code which was being executed when the exceptional condition was detected

endexecution — procedure to terminate execution of the ALGOL W program

specialcondition — logical value which is true if, and only if, the condition being processed is one of those listed below

default — result of the operation or function provided by the ALGOL W system prior to invocation of the exception processing procedure; this is defined by the hardware for arithmetic operations and is the value 0 for standard functions and for input operations. (cf. IBM System/360 Principles of Operation, IBM Systems Library, Form A22-6821)

resultant — value to be returned as the result of the arithmetic evaluation, standard function invocation, or input operation

adjustment — adjusted result of the operation according to the following table

Specialcondition	Adjustment
exponent overflow, division by zero	if default < 0 then -MAXREAL else MAXREAL
exponent underflow	0L
argument X out of domain for :	
SQRT, LONGSQRT	SQRT(abs X), LONGSQRT(abs X)
EXP, LONGEXP	MAXREAL
LN, LONGLN	-MAXREAL
LOG, LONGLOG	-MAXREAL
SIN, LONGSIN	0L
COS, LONGCOS	0L

endfile on input; according to type:

numerical	0
logical	false
string	" "
bits	#0

The reference variable UNFL is initialized by the system to null. All other reference variables listed above are initialized to references to a special record. Interrogation of this record by the procedure described above causes the ALGOL W program to be terminated with a message indicating the type of exception. Any other attempt to access any field of this record will result in a reference error.

Condition	XCPACTION ≠1 or 2	XCPACTION=1	XCPACTION=2	Reference=NULL
ENDFILE[1]	0	0	0	0
OVFL	exponent 128 too small	±MAXREAL	0	exponent 128 too small
UNFL	exponent 128 too large	0	0	0
DIVZERO	dividend	±MAXREAL	0	dividend
INTOVFL	true result ± 2**32	true result ± 2**32	true result ± 2**32	true result ± 2**32
INTDIVZERO	dividend	dividend	dividend	dividend
SQRTERR	0	SQRT(abs x)	0	0
EXPERR	0	MAXREAL	0	0
LNLOGERR	0	-MAXREAL	0	0
SINCOSERR	0	0	0	0

[1]when an endfile condition occurs on attempting to read a string, a string of blanks is supplied; for a logical value, false is returned.

Table of Results for Exceptional Conditions

8.5.1 Example

```
OVFL := EXCEPTION(FALSE, 10, 1, TRUE, "OVERFLOW FIXED UP");
```

The field values and their effects are:

```
XCPNOTED  FALSE  becomes TRUE if an overflow occurs.
XCPLIMIT  10     allows up to ten overflows before termination.
XCPACTION 1      replace the result with ±MAXREAL.
XCPMARK   TRUE   print XCPMSG each time an overflow occurs.
XCPMSG    "OVERFLOW FIXED UP"
```

ALGOL W

PROGRAMMER'S GUIDE

9. THE ALGOL W COMPILER

The compiler for the ALGOL W language is re-entrantly coded in PL360; when used it is augumented with an interface which provides communication with the host operating system. Currently two interfaces exist which provide, in effect, two compilers meeting different objectives. Differences between the two compilers lie in the disposition of the compiled program and in the program testing and library facilities which are available when the compiled program is loaded and executed.

The XALGOL W compiler is intended for use in program development and provides facilities for the compilation and (when compilation is successful) execution of one or more ALGOL W programs. A standard library is provided which cannot be augmented by the user. It does however support extensive optional aids to the debugging and analysis of programs; in particular, it is possible to obtain a summary of statement execution frequencies, a post-mortem dump of variable storage after a run-time error and a selective trace of executed statements and their effects. There is no provision for saving compiled programs; each run involves recompilation of the source program.

The ALGOL W compiler is intended for translating "production" programs, i.e., relatively large programs which are likely to be run several times before they require modification. In addition to a standard library, independently compiled procedures (coded in FORTRAN, PL360, ASSEMBLER as well as ALGOL W) may be called from libraries administered by the operating system, by means of the ALGOL W external reference facilities (cf.5.3.2.4). Most of the debugging and program analysis aids are <u>not</u> available.

Invocation of the compilers is described in sections 10 and 11.

Subsequent reference to "the compiler", unqualified by either of the names XALGOL W or ALGOL W implies reference to the compiler proper, unaugmented by either interface.

9.1 <u>The Language</u>

The language accepted by the compiler is that described in sections 2-8 of this manual subject to limitations implied in the following paragraphs of section 9.1.

9.1.1 Symbol Representation

Only capital letters are available. Basic symbols which consist of underlined letter sequences in the Language Definition are denoted by the same letter sequences without further distinction. As a consequence, they cannot be used as identifiers. Such letter sequences are called reserved words. Embedded blanks are not allowed in reserved words, identifiers and numbers. Adjacent reserved words, identifiers and numbers must be separated by at least one blank; otherwise, blanks may be used freely. The basic symbols, other than those appearing in

identifiers or numbers, are:

```
+ - * / ** ( ) = ¬= < <= > >= ¬
, ; : . # " | :: :=

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

9.1.2 Standard Identifiers

The following identifiers are predeclared, but may be redeclared due to block structure. The reference alongside each identifier is to the subsection in which the predeclared meaning is defined.

ARCTAN	8.2	LN	8.2	READCARD	7.9.2
BASE10	8.1	LNLOGERR	8.5	READON	7.9.2
BASE16	8.1	LOG	8.2	ROUND	8.1
BITSTRING	8.1	LONGARCTAN	8.2	ROUNDTOREAL	8.1
CODE	8.1	LONGBASE10	8.1	R_D	8.4
COS	8.2	LONGBASE16	8.1	R_FORMAT	8.4
DECODE	8.1	LONGCOS	8.2	R_W	8.4
DIVZERO	8.5	LONGEPSILON	8.4	SIN	8.2
ENDFILE	8.5	LONGEXP	8.2	SINCOSERR	8.5
ENTIER	8.1	LONGIMAG	8.1	SQRT	8.2
EPSILON	8.4	LONGIMAGPART	8.1	S_W	8.4
EXCEPTION	8.5	LONGLN	8.2	SQRTERR	8.5
EXP	8.2	LONGLOG	8.2	TIME	8.3
EXPERR	8.5	LONGREALPART	8.1	TRACE	9.4.3
EXPONENT	8.1	LONGSIN	8.2	TRUNCATE	8.1
IMAG	8.1	LONGSQRT	8.2	UNFL	8.5
IMAGPART	8.1	MAXINTEGER	8.4	WRITE	7.9.3
INTBASE10	8.1	MAXREAL	8.4	WRITEON	7.9.3
INTBASE16	8.1	NUMBER	8.1	XCPACTION	8.5
INTDIVZERO	8.5	ODD	8.1	XCPLIMIT	8.5
INTFIELDSIZE	8.4	OVFL	8.5	XCPMARK	8.5
INTOVFL	8.5	PI	8.4	XCPMSG	8.5
IOCONTROL	7.9.4	REALPART	8.1	XCPNOTED	8.5
I_W	8.4	READ	7.9.2		

9.1.3 Restrictions

The implementation imposes the following restrictions:

1) Identifiers consist of at most 256 characters.
2) Not more than 15 record classes can be declared.
3) Approximately 256 constants are allowed in a procedure or the outermost block.
4) Not more than 255 procedures or blocks containing declarations are allowed.
5) The data area excluding array elements for each procedure or block with declarations is limited to 4096 bytes.
6) The total amount of space occupied by the constants and machine code in any procedure or block containing

declarations may not exceed 8192 bytes.

7) The total number of blocks, procedure declarations and for statements may not exceed 511.
8) No block may be included in more than 29 other blocks.
9) Blocks with declarations, blocks associated with procedures and actual parameter lists may not be nested within one another to a depth of more than eight (counting the initial BEGIN).
10) References to not more than 63 procedures are allowed within a single procedure.

9.2 Input Format

The compiler accepts input records of 80 characters. The first 72 characters are processed as part of an ALGOL W program; characters 73 through 80 are listed but are not processed otherwise. Character 72 of one record is considered to be immediately followed by character 1 of the next record. Strings and comments should be arranged so that the character '@' does not appear in character position 1.

9.3 Compiler Directives

The compiler accepts directives inserted anywhere in the sequence of input records; these directives affect subsequent records. A directive record is marked by the character '@' in character position 1 followed by the directive starting in character position 2. The admissable directives and their functions are:

@LIST	List source records. (Initial option).
@NOLIST	Do not list source records.
@TITLE,<string>	Continue any subsequent listing on a new page. The comma and string are optional; if present, the string (of up to 30) characters stripped of the enclosing quotes is used as a title in the centre of the heading line of the new page and subsequent pages.
@SYNTAX	Check the program for syntax errors but do not execute.
@STACK	Dump the current contents of the parsing stack if a pass 2 error should occur (cf. Appendix II) with the most recent syntactic element listed last.
@NOCHECK	Omit checks on subscript ranges and reference compatibility and the initialisation of variables to "undefined".
@DEBUG,n(m)	Activate the debugging facilities. (cf. 9.4).

9.4 Debugging System

If the execution of a program terminates abnormally, a

message indicating the cause and location of the failure is always produced (cf. Appendix 2). The XALGOL W compiler optionally provides further facilities which are designed to aid in the debugging and analysis of programs. These facilities are described below; details of the debugging output and its interpretation are deferred to section 9.5.

9.4.1 Debugging Facilities

(1) Post-Mortem Dump
A dump of active storage is produced if and when any error causing abnormal termination of execution is detected. For each segment (i.e., each procedure or block with declarations), beginning with the one most recently activated, the identifiers and values of all local variables are printed. An indication of the point of activation of that segment is also given. The dump displays no more than 8 elements of each array. Unless the "@NOCHECK" directive is included, local variable storage is initialized to a special bit pattern, which is interpreted as "undefined" and printed as "?" by the dump and trace routines. Tests for the use of such values are not made as the program is executed, however, and, for most data types, the bit pattern is also a valid representation of a permissible (but relatively unusual) data value.

(2) Statement Counts
After the execution of the program, a listing is produced showing the number of times each statement in the program has been executed. The syntactic structure of the source program is used to display the source text in an edited format which emphasizes the control structure.

(3) Store Trace
The effect of executing each statement is displayed the first n times that the statement is encountered. The value of n can be selected by the programmer and modified under program control as described below. Output for each traced flow unit (normally, an elementary statement or clause) includes the following information:

(a) The source coordinate.
(b) The current frequency count for the flow unit.
(c) The source text.
(d) An indication of all assignments to variables, calls of procedures, and accesses to parameters.
(e) The values of anonymous expressions (labeled by "*") directly governing the flow of control.

Frequency counts are used to control the automatic suspension (with the printing of "...") and reinitiation (with an appropriate message) of the trace output.

(4) Fetch/Store Trace
This trace is similar to the store trace, but the identifications and values of all variables used in the evaluation of expressions are also included in the trace output.

9.4.2 The DEBUG Directive

The desired debugging options are selected by the compilation directive "@DEBUG,m(n)". The parameter m must be a single digit, with $0 \leq m \leq 4$. The facilities selected for each value of m are indicated by the following table:

m	0	1	2	3	4
post-mortem dump		x	x	x	x
flow summary			x	x	x
store trace				x	
fetch/store trace					x

The parameter n must be an unsigned integer; it is relevant only when one of the trace options has been specified and then determines the number of times each flow unit is to be traced.

The following default options and abbreviations have been established for the DEBUG directive:

(1) "@DEBUG,m" implies n=2
(2) "@DEBUG" implies m=4,n=2
(3) The default option for the XALGOL W compiler is @DEBUG,1
(4) The ALGOL W compiler ignores DEBUG directives precluding any change to the default "@DEBUG,0".

9.4.3 The TRACE Routine

The standard procedure TRACE is also provided to allow explicit control of the trace output; it has the following implicit declaration heading:

```
procedure TRACE( integer value N );
    comment if N ≥ 0, the trace limit is set to N.
    Otherwise, that limit is reset to the value implied by
    the "@DEBUG" compilation directive;
```

Calls of TRACE have no effect unless one of the trace options is selected when the program is compiled; if all tracing is to be controlled explicitly, the compilation directive "@DEBUG,4(0)" should be used.

Space required for the execution of a program is minimized by option 0; time, by option 0 or 1. Options 3 and 4 produce approximately 2n lines of output for every traced flow unit and should be used with discretion. In practice, option 4 does not produce significantly many more lines of printing than option 3.

9.5 Compiler Output

The compiler listing consists of:

1) A source program listing produced at compile time.
2) Error messages produced at both compile time and run time. (In the case of the XALGOL W compiler, error messages can also occur at the time the compiled program is loaded into core prior to execution).

3) Optional diagnostic and program analysis information produced by the debugging system at run time.

9.5.1 The Source Program Listing

The source program listing comprises four columns of output. These contain, from left to right, a four digit coordinate number, a two character block nesting level indicator, an image of the text on each source record and finally an image of the source record sequence number (if these are provided). These columns of output are separated by blank fields of widths one, six and eight characters.

(a) The Coordinate Number

A coordinate count is maintained by the compiler while scanning the source text. The count starts at zero and is incremented by one for each a semi-colon (except one ending a comment) or BEGIN passed. The coordinate number displayed at the beginning of each line is the value of this count after completion of the scanning of the preceding lines.

(b) The Block Nesting Level Indicator

The block nesting level count, L, maintained by the compiler starts at zero and is incremented by one for each BEGIN, and is decremented by one for each END, scanned. After the first BEGIN (last END) symbol on each source record, L REM 10 is evaluated and the resulting decimal digit is displayed as the first (second) character of the block nesting level indicator on the same line as the source record image. If no BEGIN or END symbol appears in a source record, then the corresponding indicator character is '-'.

(c) The Source Record Image

Characters 1-72 of each source record are displayed exactly as read. Compiler directives, i.e., records with the character '@' in position one of the record, are not listed. (Note that @LIST, @NOLIST and @TITLE affect the output of the source program listing cf.9.3.).

(d) The Source Record Sequence Number

Character positions 73-80 of each source record are displayed exactly as read. These positions are available for sequence numbers or other indicators but are frequently left blank, in which event, inadvertent typing of program beyond character position 72 (cf.9.2) of a record is clearly signalled on the source listing due to the eight blank separation between the source record and sequence number fields. Apart from listing characters 73-80, the compiler ignores them.

The source listing terminates with the messages

```
EXECUTION OPTIONS: DEBUG,m TIME=t PAGES=p
ddd.dd SECONDS IN COMPILATION, (a,b) BYTES OF CODE GENERATED
  DIRECTORY SIZES (x,y,z)
```

m is one of the integers $0 \leq m \leq 4$ (cf.9.4.2); the units of time are seconds and a,b,x,y,z are five decimal digit integers with the following significance:

a is the total amount of space occupied by the compiled machine code,
b is the amount of space occupied by compressed source text used by the debugging system,
x,y and z are the sizes of various symbol tables needed by the debugging system; the line containing these values does not appear if the debug parameter m<2.

Any errors detected during compilation, result in messages which are printed following the source program listing. Loader messages (XALGOL W only), if any, follow next (cf.Appendix II).

9.5.2 Debugging System Output

It is convenient to describe the debugging system output in terms of increasing values of m in the DEBUG,m directive.

1) Error Messages

In the event of a run time error, one of the messages described in Appendix II is printed. If the debug parameter m=0, only this information is printed.

2) Post-Mortem Dump

If the debug parameter m=1, the error message is followed by a post-mortem dump. The post-mortem commences with the message

```
=> TRACE OF ACTIVE SEGMENTS
```

A segment is either the body of a procedure or a block containing declarations. Segments associated with procedures are distinguished by the name of the procedure. Blocks are ambiguously referenced by the name "<BLOCK>" except the outermost which has the name "(MAIN)".

For each active segment, starting with that in which the error was detected and working backwards to the segment which invoked it and then the segment which invoked that, etc., the following information is printed:

a) The name of the segment, in the format:

```
=> SEGMENT NAME: name
```

If this message is followed by "(DEPTH n)", where n is an integer greater than or equal to two, it signifies the tracing of a recursive procedure segment at depth n. If n=1 the depth is not explicitly specified.

b) Except in the case of the segment (MAIN), a message indicating the point of invocation. This message is in one of the forms:

```
name WAS ACTIVATED FROM invoker, NEAR COORDINATE xxxx
name WAS REENTERED FROM invoker, NEAR COORDINATE xxxx,
     TO ACCESS A PARAMETER
```

The second message arises in circumstances like the following. Consider a program skeleton

```
BEGIN
  PROCEDURE A(REAL P)
    BEGIN...; R := P;...
    END;
  PROCEDURE X;
    BEGIN
      REAL B;
      ...; A(B * SQRT(-1));...
    END;
  ...; X;...
END.
```

When X is invoked it eventually calls A passing B*SQRT(-1) as an unevaluated parameter, but when A encounters the formal parameter P, the substitution of the actual parameter B*SQRT(-1) entails reentering X because the actual parameter B*SQRT(-1) is local to X. When the SQRT procedure fails on the negative argument, the second of the above messages will occur in the dump, where 'name' would be X and 'invoker' would be A. Subsequently in the dump a message of the first type would be output where 'name' is X and 'invoker' is (MAIN).

In the event that the message output is of the first type it is preceded by:

c) The names and values of local variables in the segment. These are printed in the format,

```
name = value
```

usually four to a line. Unless @NOCHECK was specified during compilation, values of variables which have not been initialised appear as "?". Values of parameters for which local copies have been created are identified by a name which is the corresponding formal parameter name with a prime appended. Strings are printed with initial and terminal quotes but internal quotes are not doubled. Reference values are printed in the format,

```
name = recordclassname.integer
```

where the integer is the ordinal number of the record in order of creation. At most eight elements of an array are listed (the seven smallest values of the first subscript with other subscripts taking their minimum value and the element where all subscripts take their maximum value). Control variable values which are followed by an asterisk indicate that the value is the last value prior to exit from the for statement. If an asterisked entry appears in the dump the dump terminates with

```
    * LAST VALUE OF CONTROL IDENTIFIER PRIOR TO NORMAL EXIT
```

3) Statement Counts.

If the debug parameter m=2, then between the error message and the post-mortem dump a listing is produced of the program text suitably edited to display the control structure of the program clearly. Coordinate numbers are shown on this listing as on the source program listing. The execution count information is represented between the column of coordinate numbers and the program text.

Apart from irrelevant details of layout, figure 1 shows the run time output of a small program with m=2

```
=> EXECUTION FLOW SUMMARY

0000        1.--|       BEGIN
0001            |       INTEGER SUM, COUNT, NUMB;
0002            |       WHILE  TRUE  DO
0002          2.--|       BEGIN  SUM := 0;   COUNT := 0;
---- ERROR ----------------------------------------------
0005                |       READON(NUMB);
---- ERROR ----------------------------------------------
0006                |       WRITEON(NUMB);
0007                |       WHILE  NUMB ¬= -1  DO
0007              3.--|          BEGIN  SUM := SUM + NUMB;
0009                  |          COUNT := COUNT + 1;
0010                  |          READON(NUMB);  WRITEON(NUMB)
0011                  |          END;
0012                |       IF  COUNT = 0  THEN
0012              0.--|          WRITE("EMPTY GROUP")
0012                |       ELSE
0012              1.--|          WRITE("AVERAGE", SUM/COUNT);
0013                |       IOCONTROL(2)
0013                |       END
0013            |       END

=> TRACE OF ACTIVE SEGMENTS

=> SEGMENT NAME:  (MAIN)

  VALUES OF LOCAL VARIABLES:
    SUM = 0       COUNT = 0       NUMB = -1
EXECUTION TERMINATED
```

Figure 1.

The statement count listing follows the message

```
=> EXECUTION FLOW SUMMARY
```

If a run time error has occurred, horizontal lines delimit the approximate location of the error. Each line of source text is preceded by the symbol "|" and the algorithm for interpreting the flow summary is:

(a) Select any statement and locate the "|" to the left of the corresponding source text.
(b) If the "|" is labeled by a count, terminate with that count.
(c) Otherwise, read up to find the first "|" which is directly above or to the left of the original; ignore all those to the right.
(d) Repeat from step (b).

The resulting count will be one too high if the algorithm above passes the point of a terminating error or a procedure call leading to that error.

4) The Store/Fetch Trace.

If m=3 or 4, in addition to obtaining statement counts and (following an error) an error message and post-mortem dump,during the execution of the program, values of variables are output whenever they are fetched for use (m=4, only) or whenever a new value is assigned.

As each statement (or clause) is executed under the tracing facility the following are printed,

a) the coordinate number,
b) the number of times it has been executed,
c) the text of the statement (or clause),
d) the trace information, notably the names and values of variables, and possibly expressions, within the statement.

Items a),b) and c) are printed on one line in the format used in the statement count listing; printing associated with d) is on the succeeding line, indented one position to the right of the text on the preceding line.

Examples.

```
0017        1.--|       R2 := ANSWER
                         ANSWER = RNODE.1; R2 := RNODE.1
0018        1.--|       IF N<7 THEN
                         N = 5; * = TRUE
```

As in the case of the post-mortem dump, naming conventions and notation are introduced to name values and specify the occurrence of events which are not explicitly named in the source program. Wherever possible, the conventions that are used in the post-mortem dump are employed. The following message formats are used in the trace information

name = value	(m=4 only), 'name' is the name of a variable used in the statement.
name := value	'name' is the name of a variable to which a new value is assigned by the statement.
name' := value	'name' is the name of a formal parameter with the VALUE OR RESULT

attribute. A new value is assigned to the 'local copy' of this parameter.

recordclassname.integer — is used to denote a reference value. The integer is the ordinal number (in the sequence in which records of this class are created) of the record which is the value.

* = value — * is used as the name of the (anonymous) expressions in if, while or case clauses.

-> name — indicates a call to the procedure with identifier 'name'.

=> TRACING name — indicates that a new segment is being traced.

+ RESUMING name — indicates that the trace of a segment is continuing after return from another segment.

<PARAMETER ASSIGNMENT> — indicates the performance of operations which bind a formal parameter to an actual parameter.

name(..) = value — indicates the value returned by a function procedure with identifier 'name'.

<<PARAMETER IN name AT coordinate: trace>>
if an actual parameter of the procedure 'name', called at 'coordinate' is an expression or statement then 'trace' is a trace of the parameter evaluation. In that parameters may be procedures which can invoke procedures this message often is nested within itself.

formalname :- actualname — indicates formal-actual parameter assignments.

— is used as a name for anonymous expressions, e.g., as 'actualname' in the preceding message type when the actual parameter is an expression.

... — indicates that tracing has been suspended either because the next statement has already been traced n times (cf.@DEBUG,m(n) in 9.4.2) or because of the action of the TRACE procedure (cf.9.4.3).

Return of control to the beginning of an iterated statement is indicated by repeating part of the program text, such as,

```
FOR control identifier
WHILE condition
```

but within parentheses to indicate continuation of a previously activated statement. This text is followed by the trace of the values involved.

Whenever a new record is read by a READ or READON statement, the complete card image is printed as a string in the message

```
INPUT RECORD: string
```

Strings are printed with an initial and a final quote but internal quotes are not doubled.

9.6 Externally Defined Procedures

Procedures, which are defined externally to the ALGOL W programs which invoke them, can use either ALGOL W or FORTRAN linkage and parameter conventions. The former are obtained automatically when an ALGOL W procedure declaration is compiled. The corresponding machine code is subject to change and will not be documented here. FORTRAN linkages are identical to the standard IBM S-type linkages, which are described in detail elsewhere (See, for example, MTS Volume 3, pages 15-24 or FORTRAN IV (G and H) Programmer's Guide, IBM SRL Form GC28-6817, Appendix C). They are produced by the IBM FORTRAN compilers and also used by many PL360 and assembly language programs.

9.6.1 ALGOL W Procedures

ALGOL W procedure declarations which stand as programs (cf.7.0) must satisfy the following restrictions:

(1) No unbound (global) identifiers, except those considered to be declared in the standard environment, are allowed.

(2) Declarations of record classes (and thus of reference quantities) are subject to special rules. They should normally be avoided.

Independently compiled procedures are known in the system environment (i.e., to the loader) by the names of their entry points; these are formed by extending (with "#"'s) or truncating the ALGOL W procedure identifier to 5 characters and appending "001".

9.6.2 FORTRAN Subroutines

A FORTRAN subroutine or subprogram can be used as an ALGOL W procedure. The type correspondence between ALGOL W and FORTRAN is given by the following table:

ALGOL W	IBM FORTRAN IV
integer	INTEGER*4
real	REAL*4
long real	REAL*8
complex	COMPLEX*8
long complex	COMPLEX*16
logical	LOGICAL*1
string (n)	(LOGICAL*n)
bits	LOGICAL*4
reference	- - -

String functions are not implemented. The permitted formal parameter specifications follow with their interpretations:

(1) <T-type>
The corresponding actual parameter is examined. If that parameter is a variable, the address of that variable is computed (once only) and transmitted. Otherwise, the expression which is the actual parameter is evaluated, the value is assigned to an anonymous local variable, and the address of that variable is transmitted.

(2) <T-type> value , <T-type> result , <T type> value result
As in ALGOL W procedures, a local variable unique to the call is created, and the address of that variable is transmitted.

(3) <T-type> array
The address of the actual array element with unit indices in each subscript position is computed and transmitted, even if that element lies outside the declared bounds of the ALGOL W array. Arrays with only one dimension and arrays with unit lower subscript bounds will have elements with indices which are identical in ALGOL W and FORTRAN routines. Array cross-sections should not normally be used as actual parameters of FORTRAN subprograms.

If FORTRAN input/output or FORTRAN error handling facilities are to be used, the subroutine package IBCOM, or a suitable substitute, is required.

9.6.3 External References

An external reference (cf. 5.3.2.4) standing as a procedure body is used to establish the connection between an ALGOL W program and an independently prepared procedure. The symbols algol and fortran in that reference indicate the use of ALGOL W and S-type linkage conventions respectively; the associated string is extended (with blanks) or truncated to 8 characters and taken as the entry point name of the external procedure. For a FORTRAN external procedure the entry point name is simply the name of the FORTRAN subroutine or function. For an ALGOL W external procedure the entry point name is derived from the procedure name as described in 9.6.1.

9.6.4 Example

The first program (outline) is an ALGOL W procedure which is invoked as an external procedure in the second program.

```
INTEGER PROCEDURE EXTFUN (REAL VALUE X);
  BEGIN...
  END.

BEGIN
  INTEGER I; REAL A,B;
  INTEGER PROCEDURE INTFUN(REAL VALUE Y);
    ALGOL "EXTFU001";
  ...; I := INTFUN(A * B); ...
END.
```

10 ALGOL W IN MTS

The two versions of the compiler referred to in section 9 are located in files *XALGOLW and *ALGOLW.

10.1 Summary

The needs of most users will be met by one of the "recipes" given below. More detailed information about these and other processing options and facilities is contained in sections 10.2 and 10.3.

MTS will compile and execute ALGOL W programs if the input is arranged as follows:

(1) For batch processing, construct a card deck (or a file of card-image records) which includes the following sequence:

```
$RUN  *XALGOLW
$ALGOL  T=t  P=p     ---ð
<program>               |  repeat
$DATA                   |  0 or more times
<data>               ---┘
$ENDFILE
```

All output will be directed to the line printer.

(2) From a terminal, issue a command of the following form:

```
$RUN *XALGOLW SCARDS=sourcef 1=listingf
```

The file sourcef must contain a sequence of source programs and data; it should be arranged as follows:

```
$ALGOL  T=t  P=p     ---ð
<program>               |  repeat
$DATA                   |  0 or more times
<data>               ---┘
```

Note that lines containing more than 80 characters will be truncated (with a warning message) and that only the first 72 characters of a line will be examined for ALGOL W source text. The file listingf (normally, a temporary file) will receive the compilation listing. All compilation diagnostics and all output from the execution of user programs will be directed to the terminal.

In both cases, the parameters on the corresponding "¢ALGOL" control line will limit the resources allowed for the execution (not compilation) of an ALGOL W program to t seconds of (problem state) CPU time (default: 10 seconds) and p pages of printed output (default: 10 pages). In batch, the corresponding global limits take precedence if, and only if, they are exceeded first. Note that the diagnostics produced when ALGOL W forces program termination usually are more helpful than the corresponding MTS messages.

10.2 MTS *XALGOLW Specifications

*XALGOLW contains a monitor which supervises the compilation and immediate execution of a sequence of ALGOL W programs. No object files are created; programs are compiled directly into main memory and then executed. *XALGOLW is invoked by a command of the following form:

```
$RUN *XALGOLW [SCARDS=sourcef] [SPRINT=outputf] [SERCOM=errorf]
                              [1=listingf]
```

The logical devices are used as follows:

(1) SCARDS supplies source programs and data. Each program to be compiled and executed must be delimited by control lines; the required format is described below. Library files, data files, and the like can be inserted into the source stream by use of the facilities provided by MTS for explicit or implicit file concatenation.

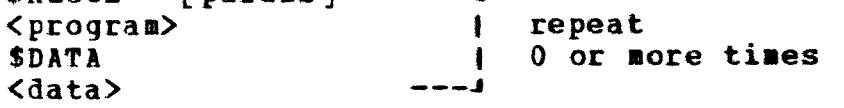
(2) SPRINT receives all output resulting from execution of the compiled program(s). If logical device 1 is not specified, SPRINT also receives the compilation listing(s).

(3) SERCOM receives any diagnostic messages generated by the compilation step(s).

(4) If logical device 1 is specified, it receives the compilation listing(s); otherwise, this output is directed to SPRINT.

In all output files, the first character of each line is a logical carriage control code supplied by the system. In the absence of explicit specifications, the logical devices SCARDS, SPRINT, and SERCOM are associated with the pseudofiles *SOURCE*, *SINK*, and *MSINK* respectively.

The file sourcef must contain a sequence of ALGOL W programs and data. Any line beginning with the string "$ALGOL" or the string "$DATA" is considered to be a control line, and control lines must be used to delimit each program according to the following scheme:



The "$DATA" control line can be omitted if there is no input data. Note that lines of sourcef containing more than 80 characters are truncated (with a warning message) and that only the first 72 characters of a line are examined for ALGOL W source text. The "$ALGOL" control line can be used to specify optional keyword parameters. A SIZE parameter controls the amount of working storage available for compilation and execution of the program. TIME and PAGES parameters establish limits on the problem state CPU time to be used and the number of pages to be printed during the execution (not compilation) of the program. Any MTS limits take precedence over these if (and only if) they

are exceeded first. Note that execution time rationing is not exact; execution time can overrun the specified limit by an unpredictable amount not exceeding 0.5 second. Details of parameter specification are given by the following table.

Format:

```
[SIZE=m[K|P]] [{TIME|T}=t[M|S]] [{PAGES|PGS|P}=p]
```

where m, t, and p are unsigned integers.

Interpretation:

Parameter	SIZE	TIME	PAGES
Abbreviations		T	P, PGS
Units	bytes	seconds	pages
Scale Factors (Values)	K (1024)	S (1)	
	P (4096)	M (60)	

All parameters are optional; the default values are equivalent to the specification

```
SIZE=48K  TIME=10S  PAGES=10
```

10.3 MTS *ALGOLW Specifications

*ALGOLW contains a routine which calls the compiler to process a single source program. A standard MTS object file is created. When the object program is subsequently executed, the standard ALGOL W library is made available automatically; other libraries can be provided explicitly. *ALGOLW is invoked by a command of the following form:

```
$RUN *ALGOLW [SCARDS=sourcef] [SPRINT=listingf] [SERCOM=errorf]
                SPUNCH=objectf [PAR=param]
```

The logical devices are used as follows:

(1) SCARDS supplies the source program. The associated file should contain exactly one ALGOL W source program (without data or delimiting control lines). Note that only the first 72 characters of any line are inspected for source text.

(2) SPRINT receives the compilation listings.

(3) Diagnostic messages produced by the compiler are directed to SERCOM

(4) SPUNCH receives the object records containing the text of the compiled program.

In the files associated with SPRINT and SERCOM, the first character of each line is a logical carriage control code. In the absence of explicit specifications, the logical devices SCARDS, SPRINT, and SERCOM are associated with the pseudofiles *SOURCE*, *SINK*, and *MSINK* respectively. The file associated with SPUNCH will receive card image records. Object files obtained by compilation of ALGOL W main programs include a record specifying implicit concatenation (i.e., a line beginning

"$CONTINUE WITH"). The concatenated file is sequential and cannot be processed correctly by most programs which operate upon line files. Thus care is necessary in specifying operations (such as copying) upon ALGOL W object files; normally, the modifier "@¬IC" should be used. If errors are detected in the source program, the object file will be empty or incomplete.

The optional parameter specifies the amount of working storage, in bytes, available for compilation. It has the form

SIZE=m[K|P]

where m is an unsigned integer. K and P are scale factors with values of 1024 and 4096 respectively. Omission of the parameter is equivalent to the specification

PAR=SIZE=56K

Execution of an ALGOL W object program contained in a file objectf is specified by a command of the following form:

$RUN objectf [SCARDS=inputf] [SPRINT=outputf] [PAR=params]

Implicit concatenation must be enabled when the file is loaded. Explicit concatenation of object files can be used to include the object code for precompiled procedures. Alternatively, such code can be selectively loaded from a public or private library (see the descriptions of *GENLIB, LOAD, etc.). In any case, the effective input to the MTS loader must contain the object code for exactly one ALGOL W main program, i.e., the object code obtained by compiling a statement (cf.7. And 9.6). The logical devices are used as follows:

(1) SCARDS is the standard input stream. Lines longer than 80 characters will be truncated (with a warning message on SERCOM). Implicit and explicit concatenation of input files can be used.

(2) SPRINT is the standard output stream. Unless the parameter NOCC is specified (see below), the first character of each line is a logical carriage control code supplied by the system and the maximum line length is 133. If NOCC is specified, the control character is omitted and the maximum line length is 132.

In the absence of explicit specifications, the logical devices SCARDS and SPRINT are associated with the pseudofiles *SOURCE* and *SINK* respectively.

Optional parameters can be supplied to control program execution. A SIZE parameter determines the amount of working storage available; TIME and PAGES parameters place bounds upon the resources available to the executing program. Details are given by the following table:

Format:

[SIZE=m[K|P]] [{TIME|T}=t[M|S]] [{PAGES|PGS|P}=p] [CC|NOCC]
where m, t, and p are unsigned integers.

Interpretation:

Parameter	SIZE	TIME	PAGES
Abbreviations		T	P, PGS
Units	bytes	seconds	pages
Scale Factors (Values)	K (1024)	S (1)	
	P (4096)	M (60)	

The TIME parameter establishes a limit upon the problem state CPU time used. This limit can be overrun by an unpredictable amount not exceeding 0.5 second. Any MTS limits take precedence over the TIME and PAGES limits if (and only if) they are exceeded first. The NOCC parameter suppresses the carriage control codes normally supplied with each line output by the program. This suppression is independent of the attributes of the output file or pseudofile; if those attributes specify logical carriage control, the first character of the actual output line will be used as the control code. In this way, explicit carriage control can be programmed. When NOCC is specified, each group of 60 output lines is considered to be a page for the purpose of page limit monitoring. All parameters are optional. The execution time and printed output are not monitored unless limits are explicitly specified; the default parameter specification is equivalent to

PAR=SIZE=36K CC

10.4 MTS System Error Messages

On occasion one or other of the messages

```
**SYSTEM ERROR.  JOB ABORTED
PROGRAM INTERRUPT.  PSW = hhhhhhhh hhhhhhhh
```

may appear; h indicates a hexadecimal digit. If the source of the trouble is not obvious, take the listing and card deck, if available, to a consultant.

If the time or page limits specified in a batch mode SIGNON command are exceeded an error message is printed indicating which of the limits has been exceeded then the job is terminated.

11. ALGOL W IN OS

The two versions of the compiler referred to in section 9 are invoked respectively by the catalogued procedure XALGOLW or one of the catalogued procedures ALGWC, ALGWCL and ALGWCLG.

11.1 OS Summary

The needs of most users will be met by the "recipe" given below. More detailed information about this and other processing options and facilities is contained in sections 11.2 and 11.3.

For batch processing, construct a card deck (or a file of card image records for submission to the batch stream) which includes the following sequence:

```
//        EXEC  XALGOLW
//X.SYSIN  DD  *
%ALGOL  :t,p          ---ð
<program>                 |   repeat
%EOF                      |   0 or more times
<data>                ---ȷ
/*
```

All output will be directed to the line printer. Note that only the first 72 characters of an input record will be examined for ALGOL W source text. The job should be run in a region of at least 120K bytes.

The parameters on the "%ALGOL" control record will limit the resources allowed for the execution (not compilation) of the corresponding ALGOL W program to t seconds of CPU time (default: 10 seconds) and p pages of printed output (default: 10 pages). Any limits upon these quantities which are specified in the OS JOB or EXEC statement take precedence if, and only if, they are exceeded first. Note that the diagnostics produced when ALGOL W forces program termination usually are more helpful than the corresponding OS messages.

11.2 OS XALGOLW Specifications

The catalogued procedure XALGOLW, invokes a monitor which supervises the compilation and immediate execution of a sequence of ALGOL W programs. No object files are created; programs are compiled directly into main memory and then executed.

The maximum size of ALGOL W programs which can successfully be compiled and executed is determined by the amount of main storage available. The minimum requirement for compilation is 100K bytes, and capacity increases quickly with additional storage. 120K bytes will be adequate for most programs not exceeding 300 to 400 source records; larger programs will usually require a larger region, as will those using large amounts of storage during execution.

Data definition (DD) statements with the names SYSIN and SYSPRINT must be provided; the corresponding data sets are used as follows:

(1) The SYSIN stream supplies card-image records containing the source programs and data. Each program to be compiled and executed must be delimited by control records; the required format is described below. Library files, data files, and the like can be inserted into the source stream by use of the facilities provided by OS for the concatenation of data sets.

(2) The SYSPRINT stream receives the compilation listing(s), any diagnostic messages generated by the compilation step(s), and all output resulting from execution of the compiled program(s). The associated data set will contain line-image records, in which the first character of each logical record is an ANSI carriage control character automatically supplied by the system.

These data sets have the following OS attributes:

DD Name	SYSIN	SYSPRINT
Format (RECFM)	FB	FBA
Record Length (LRECL)	80	133

A corresponding DD statement or data set label can supply the physical block size (BLKSIZE) and number of buffers (BUFNO). The block size must be an integral multiple of the logical record length. If these attributes are not otherwise specified, the record length and block size are assumed to be identical, and two buffers are provided. QSAM is used for all input/output operations referencing these streams.

The stream SYSIN must contain a sequence of ALGOL W programs and data. Any record beginning with the string "%ALGOL" or the string "%EOF" is considered to be a control record, and control records must be used to delimit each program according to the following scheme:

```
%ALGOL    [params]    ---δ
<program>                |  repeat
%EOF                     | 0 or more times
<data>                   |
                         ---┘
```

The "%EOF" record can be omitted if there is no input data. Note that only the first 72 characters of a record are examined for ALGOL W source text. The "%ALGOL" control record can be used to specify optional parameters. These establish limits on the problem state CPU time to be used and the number of pages to be printed during the execution (not compilation) of the program. Any OS job or step limits take precedence if (and only if) they are exceeded first. Details of parameter specification are given by the following table.

Format:

```
                    [ {m|[m]:s} ][,p]
```

where m, s, and p are unsigned integers.

Interpretation:

Parameter	m	s	p
Limit	time	time	pages
Units	minutes	seconds	pages

Parameters must be coded between columns 8 and 72, inclusive, of the corresponding control record. All parameters are optional; the default values are equivalent to the specification

0:10,10

11.2.1 Examples

(1) The cataloged procedure XALGOLW contains the following JCL statements (Newcastle):

```
//X        EXEC  PGM=XALGOLW
//STEPLIB  DD    DSN=SYS2.ALGOLW,DISP=SHR
//SYSPRINT DD    SYSOUT=A
```

The input stream must contain a definition of SYSIN.

(2) The following job step specifies the processing of a source program in a private disk file, XXX99.SOURCE on the volume UNE020, plus some input data on cards. Output is to be directed to a new data set, XXX99.RESULTS on the volume UNE030.

```
//G        EXEC  PGM=XALGOLW
//STEPLIB  DD    DSN=SYS2.ALGOLW,DISP=SHR
//SYSPRINT DD    DSN=XXX99.RESULTS,UNIT=2314,
//                VOL=SER=UNE030,DISP=(NEW,KEEP),
//                DCB=BLKSIZE=1330,SPACE=(1330,(100,25))
//SYSIN    DD    DSN=XXX99.SOURCE,UNIT=2314,
//                VOL=SER=UNE020,DISP=(OLD,KEEP)
//           DD  *
<data>
/*
```

In this example, the source file must include the necessary "%" control records. Its blocking factor is obtained from the data set label. In the output data set, the blocking factor is 10, and 1000 lines of output are anticipated.

11.3 OS ALGOLW Specifications

The cataloged procedures ALGWC, ALGWCL, and ALGWCLG, call the ALGOL W compiler to process a single source program. A data set containing standard OS object modules is produced. When the object modules are subsequently edited and executed, the standard ALGOL W library is made available automatically; other libraries can be explicitly provided.

The maximum size of ALGOL W programs which can successfully be compiled is determined by the amount of main storage available. The minimum storage requirement is 100K bytes, and

capacity increases quickly with additional storage. 120K bytes will be adequate for most programs not exceeding 400 to 500 source records; larger programs will usually require a larger region.

Data definition statements for the input stream SYSIN and the output streams SYSPRINT and SYSLIN are required. The corresponding data sets are used as follows:

(1) The SYSIN stream supplies the source program. The associated data set should contain exactly one ALGOL W source program (without data or "%" control records). Note that only the first 72 characters of any record are inspected for source text.

(2) The SYSPRINT stream receives the compilation listing and any diagnostic messages produced by the compiler. The first character of each logical record is an ANSI carriage control code.

(3) The SYSLIN stream receives the object records containing the text of the compiled program.

The data sets have the following attributes:

DD Name	SYSIN	SYSPRINT	SYSLIN
Format (RECFM)	FB	FBA	FB
Record Length (LRECL)	80	133	80

A corresponding DD statement or data set label can supply the physical block size (BLKSIZE), which must be an integral multiple of the record length, and the number of buffers (BUFNO). In the absence of explicit specifications, the assumed values are the record length and 2, respectively. Note that the linkage editor will accept only a limited range of blocking factors, and the attributes of the SYSLIN data set should be chosen according to installation standards.

The SYSLIN stream will receive card image records. Object files obtained by compilation of ALGOL W main programs include the following linkage editor control statements:

```
INCLUDE  SYSLIB(ALGOLX)
ENTRY    ALGOLX
```

If errors are detected in the source program, or if the "@SYNTAX" compilation directive (cf.9.3) is used, the object file will be empty or incomplete, and the return code supplied will be 16. Otherwise, the return code will be 0.

The object modules produced must be processed by the linkage editor before they can be loaded and executed. A complete discussion of all the relevant facilities provided by the linkage editor and OS library management utilities is well beyond the scope of this manual. Such a discussion can be found in the IBM Systems Reference Library publication, Linkage Editor and Loader, Form GC28-6538. The following points should be observed:

(1) A DD statement defining SYSLIB must be provided; the corresponding data set must be partitioned and must include the standard ALGOL W library modules among its members.

(2) Normally, the object modules corresponding to the ALGOL W main program will be contained in the SYSLIN data set, which will be passed from the preceding compilation step(s).

(3) Object code for precompiled procedures can be contained in the SYSLIN data set, in the SYSLIB data set, or in auxiliary data sets specified by LIBRARY or INCLUDE statements in the SYSLIN stream.

(4) In any case, the effective input to the linkage editor must contain the object code for exactly one ALGOL W main program, i.e., the object code obtained by compiling a statement (cf. 7 and 9.6).

A complete ALGOL W program can be loaded and executed after it has been converted to a load module by the linkage editor. DD statements with names SYSIN and SYSPRINT must be provided; the corresponding data sets are used as follows:

(1) SYSIN is the standard input stream and should contain card-image records. Concatenation of data sets is permitted. If there is no input stream, the definition DUMMY should be used.

(2) SYSPRINT is the standard output stream. The length of all logical records is 133. Unless the parameter MYCC is specified (see below), the first character of each logical record is an appropriate ANSI carriage control code, which is automatically provided by the system. If MYCC is specified, the first character of each output record constructed by the ALGOL W program itself is assumed to be the ANSI carriage control code, and a blank is appended as the last character of the record.

These data sets have the following OS attributes:

DD Name	SYSIN	SYSPRINT
Format (RECFM)	FB	FBA
Record Length (LRECL)	80	133

A corresponding DD statement or data set label can supply the physical block size (BLKSIZE) and number of buffers (BUFNO). If these attributes are not otherwise specified, the record length and block size are assumed to be identical, and two buffers are provided. QSAM is used for all input/output operations referencing these streams.

DD statements for any data sets referenced in (non-ALGOL W) precompiled procedures are also required.

Optional parameters can be supplied as PARM information in the OS EXEC statement which invokes a compiled ALGOL W program. These parameters establish limits upon the CPU time to be used and the number of pages to be printed during execution of the

program. Any limits upon these quantities which are specified (implicitly or explicitly) in the OS job control language take precedence if, and only if, they are exceeded first. Details are given by the following table:

Format:

```
[{m|[m]:s}][,[p][,MYCC]]
```

where m, s, and p are unsigned integers.

Interpretation:

Parameter	m	s	p
Limit	time	time	pages
Units	minutes	seconds	pages

The MYCC parameter suppresses the carriage control codes normally supplied with each line output by the program (see above). The default limits are equivalent to the specification

```
PARM='1:00,60'
```

i.e., 60 seconds of CPU time and 60 pages of printed output. Note that all parameters are positional; commas are required even if operands are omitted.

11.3.1 Examples

(1) The cataloged procedure ALGWCLG contains the following JCL statements (Newcastle):

```
//C        EXEC  PGM=ALGOLW,REGION=150K
//STEPLIB  DD    DSN=SYS2.ALGOLW,DISP=SHR
//SYSPRINT DD    SYSOUT=A
//SYSLIN   DD    DSN=&TEMP2,UNIT=2314,
//               VOL=SER=UNE999,
//               SPACE=(400,(&SIZE,15)),DISP=(NEW,PASS),
//               DCB=BLKSIZE=400
//L        EXEC  PGM=IEWL,PARM='LIST,MAP',
//               COND=(4,LT,C),REGION=100K
//SYSPRINT DD    SYSOUT=A
//SYSUT1   DD    DSN=&TEMP1,UNIT=2314,
//               SPACE=(1024,(&SIZE,2)),
//               VOL=SER=UNE999,DCB=BLKSIZE=1024
//SYSLIB   DD    DSN=SYS2.ALGWLIB,DISP=SHR
//SYSLMOD  DD    DSN=&&G(MAIN),UNIT=2314,
//               SPACE=(CYL,(2,1,1)),DISP=(,PASS)
//SYSLIN   DD    DSN=*.C.SYSLIN,
//               DISP=(OLD,DELETE)
//G        EXEC  PGM=*.L.SYSLMOD,
//               COND=((4,LT,C),(4,LT,L))
//SYSPRINT DD    SYSOUT=A
```

The input stream can contain definitions of SYSIN for the compiler (C.SYSIN), linkage editor (L.SYSIN), and ALGOL W program (G.SYSIN). The cataloged procedures ALGWC and ALGWCL are similar but include job control statements for just the first one or two job steps respectively.

(2) The following example uses the cataloged procedures ALGWC and ALGWCLG to compile and execute an ALGOL W program. One of the procedures, written in ALGOL W, is to be compiled separately. Additional precompiled procedures, originally written in FORTRAN and located in the data set SYS2.LOAD.SSP, are also referenced by the main program and are made available to the linkage editor by concatenation to the standard library.

```
//STEP1  EXEC  ALGWC
//C.SYSIN  DD  *
<Algol W procedure (source code)>
/*
//STEP2  EXEC  ALGWCLG
//C.SYSLIN DD  DISP=(MOD,PASS)
//C.SYSIN  DD  *
<Algol W main program>
/*
//L.SYSLIB DD  DSN=SYS2.ALGWLIB,DISP=SHR
//         DD  DSN=SYS2.LOAD.SSP,DISP=SHR
//G.SYSIN  DD  *
<data>
/*
```

11.4 OS System Error Messages

On occassion one or other of the messages

```
*** ABNORMAL JOB END ***  SYSTEM CODE=XXX
COMPLETION CODE - SYSTEM=XXX
```

may appear. If XXX is 222, 322 or 722 the job was terminated respectively by the operator, by the system on exceeding the time limit or by the system on exceeding the line limits. (These messages imply exceeding time and line limits in JCL, (not %ALGOL) records.

APPENDIX I - CHARACTER ENCODINGS

The following table presents the correspondence between printable string characters and their (EBCDIC) integer encodings. This encoding establishes the ordering relation on characters and thus on strings. Those characters in parentheses are not normally available on the line printer. Integer codes not listed below do not correspond to any established character. (cf. CODE, DECODE in section 8.1).

64	space	129	(a)	193	A	240	0
74	(c)	130	(b)	194	B	241	1
75	.	131	(c)	195	C	242	2
76	<	132	(d)	196	D	243	3
77	(	133	(e)	197	E	244	4
78	+	134	(f)	198	F	245	5
79	|	135	(g)	199	G	246	6
80	&	136	(h)	200	H	247	7
90	(!)	137	(i)	201	I	248	8
91	¢	145	(j)	209	J	249	9
92	*	146	(k)	210	K		
93	)	147	(l)	211	L		
94	;	148	(m)	212	M		
95	¬	149	(n)	213	N		
96	-	150	(o)	214	O		
97	/	151	(p)	215	P		
107	,	152	(q)	216	Q		
108	%	153	(r)	217	R		
109	_	162	(s)	226	S		
110	>	163	(t)	227	T		
111	?	164	(u)	228	U		
122	:	165	(v)	229	V		
123	#	166	(w)	230	W		
124	@	167	(x)	231	X		
125	'	168	(y)	232	Y		
126	=	169	(z)	233	Z		
127	"						

APPENDIX II - ALGOL W ERROR MESSAGES

Only error messages generated by the compiler are listed here. Occassionally contravention of operating system requirements lead to the generation of system error messages (cf. 10.4 and 11.4).

The compiler is divided into three passes: pass 1 reads the program, lists it, and saves it in memory in a compressed (tokenized) form; pass 2 parses the program, examining each statement to see if it is correctly formed; pass 3 generates the 360 machine code for the program. Each pass is capable of detecting a different set of errors. (There is also a fourth, loader, pass that on rare occasions may generate messages.) Errors may also occur while a compiled program is executing; these are called run-time errors.

All error messages from passes 1, 2 and 3 are of the form:

ERROR zxxx NEAR COORDINATE yyyy - message

where zxxx is the error number, z is 1, 2 or 3 according to the pass which generates the message and yyyy corresponds to one of the coordinate numbers in the first column on the program listing. If there are several statements on a line, only the coordinate of the first one appears on the program listing.

II.1 Pass One Error Messages

1001 INCORRECTLY FORMED DECLARATION

a) STRING(x) or BITS(x), where x is not a number.

b) STRING(0) or STRING(>256).

c) BITS (not 32).

1002 INCORRECT CONSTANT

a) More than 256 digits.

b) a bad exponent.

1003 MISSING "END"

A final "." or a control card encountered before an END matching each BEGIN. (Check the block numbers in the second column of the program listing.)

1004 UNMATCHED "END" (DELETED)

An END encountered after what appeared to be the final END. When possible, the innermost END is deleted. (Check the block numbers in the second column of the program listing.)

1005 MISSING ")"

STRING(x or BITS(x with no closing ")".

1006 ILLEGAL CHARACTER

An erroneously punched or overpunched character. Overpunched characters may print as blanks; the card should be inspected in this case.

1007 WARNING: MISSING FINAL "."

A control card encountered without a preceding ".".

1008 INVALID STRING LENGTH

A string constant of length >256, or a completely empty string; a quote may have been omitted.

1009 INVALID BITS LENGTH

a) "#" not followed by hex digits.

b) "#" followed by more than 8 hex digits.

1010 MISSING "("

REFERENCE not followed by "(".

1011 ERROR TABLE OVERFLOW

More than 50 error messages from pass 1. Subsequent errors are not listed.

1012 COMPILER TABLE OVERFLOW

The program is too big to fit in memory during compilation. There is no more room in one of the tables constructed by the compiler. On re-compiling with more memory, the tables will be bigger.

1013 ID LENGTH > 256

Overlength identifier.

1014 UNEXPECTED "."

An apparently final"." not followed by a control card, such as in a constant with an inadvertant space: . 123

1015 TOO MANY RECORD CLASSES

Only 15 are allowed.

1016 WARNING: "ELSE" PRECEDED BY (DELETED) ";"

The sequence ";ELSE" has been replaced by "ELSE"

1017 TOO MANY BLOCKS

Either a block is enclosed in more than 29 other blocks or the total number of blocks, procedure declarations and for statements exceeds 500.

II.2 Pass Two Error Messages

All pass 2 error messages are supplemented by:

(FOUND NEAR "...")

where "...." indicates a pair of symbols. In general, the first symbol is the input symbol or phrase after which the error was detected; the second is the next symbol to be scanned.

If any pass one or pass two error messages occur (other than the warnings 1007, 1016, 2013 and 2031), then compilation stops at the end of pass two. Several error messages may be generated for what is essentially a single mistake.

2001 MORE THAN ONE DECLARATION OF "XXXX" IN THIS BLOCK

The variable XXXX has been declared more than once in the same block.

2002 "XXXX" IS UNDEFINED

The variable or label XXXX has not been declared in the current block or in one containing it.

2003 Currently there is no error with this number

2004 Currently there is no error with this number.

2005 MISMATCHED PARAMETER

An actual parameter in a procedure statement is not of a type compatible with the formal parameter in the procedure declaration.

2006 INCORRECT NUMBER OF ACTUAL PARAMETERS

The number of actual parameters in a procedure call does not equal the number of formal parameters in the procedure declaration.

2007 INCORRECT DIMENSION

a) The number of dimensions of an actual parameter does not equal the number of dimensions declared for the corresponding formal parameter.

b) The wrong number of subscripts have been used in an array element reference.

2008 DATA AREA EXCEEDED

The data space for each PROCEDURE or block with declarations is limited to 4096 bytes. Read the

suggestions for 3001.

2009 INCORRECT NUMBER OF FIELDS

In creating a record, too many or too few initial values have been specified.

2010 INCOMPATIBLE STRING LENGTHS

a) In STRING1 := STRING2 , STRING2 is longer than STRING1.

b) In STRING3(x|y) , y is larger than the declared size of STRING3.

c) A long string has been passed to a shorter formal string parameter.

2011 INCOMPATIBLE REFERENCES

A reference variable refers to a record class to which it is not bound.

2012 BLOCKS NESTED TOO DEEPLY

Non-trivial blocks (i.e., blocks with declarations, or the blocks associated with a PROCEDURE) or actual parameter lists are nested more than eight deep. The error is detected early in the ninth block.

2013 WARNING: ";" SHOULD NOT FOLLOW EXPRESSION

In BEGIN ...<expression>; END the semi-colon is incorrect but ignored.

2014 REFERENCE MUST REFER TO RECORD CLASS

In REFERENCE (xyz)... , xyz is not a record class.

2015 EXPRESSION MISSING IN PROCEDURE BODY

A function PROCEDURE must have its final value specified by an expression standing alone immediately before the END.

2016 IMPROPER COMBINATION OF TYPES

Mixing incompatible types as alternatives of a conditional or case expression.

2017 RESULT PARAMETER MUST BE A VARIABLE

In a procedure declaration, a formal parameter is declared ... RESULT xyz, but a call to that procedure has passed an expression which is not a variable.

2018 PROPER PROCEDURE ENDS WITH AN EXPRESSION

A procedure which returns no value nonetheless ends with an expression. (This will happen if a final assignment

statement is using =, instead of :=).

2019 "XXXX" CANNOT FOLLOW "YYYY" HERE

The input up to the symbol denoted YYYY is part of a valid ALGOL W program, but no valid ALGOL W program can continue with the symbol XXXX.

2020 ARRAY USED INCORRECTLY

A simple variable must be used here.

2021 TOO MANY CONSTANTS IN PROCEDURE

Only 256 different constants (approximately) are allowed.

2022 INCORRECT STRING LENGTH

In S(x|y) , y is zero, or greater than 256.

2023 COMPILER TABLE OVERFLOW

The program is too big to fit into memory during compilation -- there is no more room for the parse trees that represent the program. Re-compile with more memory or compile some procedures separately.

2024 TOO MANY PROCEDURES

Only 255 different procedures or blocks with declarations are allowed by the compiler.

2025 CONSTANT OUT OF RANGE

a) The absolute value of an integer is greater than (2**31)-1 (9+ digits).

b) The absolute value of the adjusted exponent in a real number is greater than 75. The exponent written is first adjusted to include the number of digits written in front of the decimal point.

2026 INDEX OF ARRAY OR STRING MUST BE INTEGER

a) In S(x|y) , x is not an expression of integer type.

b) An array subscript is not an expression of integer type.

2027 INCORRECT OPERAND TYPE(S) FOR XXXX

XXXX is a unary operator.

a) LONG is applied to something which is already LONG, or to STRING, BITS, LOGICAL, or REFERENCE.

b) SHORT is applied to something which is neither LONG REAL nor LONG COMPLEX.

c) ¬ (not) is applied to something which is neither LOGICAL nor BITS.

d) Prefix + or - is applied to something which is LOGICAL, STRING, BITS, or REFERENCE.

e) ABS is applied to something which is LOGICAL, STRING, BITS, or REFERENCE.

f) In recordvariable(x) , x is not a REFERENCE.

g) In FOR I:=x... , x is not an integer expression.

h) In various other contexts, an INTEGER or LOGICAL operand is required.

2028 INCORRECT OPERAND TYPE(S) FOR XXXX

XXXX is a binary operator. Even when the error is in the first operand, the error is detected after both operands are inspected.

a) AND or OR is applied to expressions which are not both BITS or both LOGICAL.

b) A relational operator (like >) is applied to something which is COMPLEX, LOGICAL, or REFERENCE.

c) SHL or SHR is applied to something which is not BITS, or is followed by either an expression not enclosed in parentheses or a value which is not of integer type.

d) In x IS recordclass , x is not of type REFERENCE.

e) In x**y , y is not of type INTEGER.

f) In a FOR statement, the UNTIL expression is not of type INTEGER.

g) In various other contexts, an INTEGER type operand is required.

2029 Currently there is no error with this number.

2030 ASSIGNMENT INCOMPATIBILITY

An attempt to assign an expression of one type to a variable of a different type (or pass an actual parameter to a formal parameter of a different type). The only automatic conversions allowed are INTEGER to REAL, INTEGER to LONG REAL, REAL to/from LONG REAL, INTEGER/REAL/LONG REAL to COMPLEX/LONG COMPLEX, COMPLEX to/from LONG COMPLEX. (REAL cannot be assigned to INTEGER without using TRUNCATE, ENTIER, or ROUND.)

2031 WARNING: NAME PARAMETER SPECIFIED

In PROCEDURE declarations, it is more often the case that

formal parameters have VALUE specified. Check that the name specification is necessary.

2032 SIMPLE VARIABLE ID USED INCORRECTLY

The identifier in a substring designator is not type STRING.

2033 ... FURTHER MESSAGES SUPPRESSED

More than 64 errors detected, compilation continues with further messages suppressed.

2999 DEBUG TABLE OVERFLOW

If @DEBUG,x is specified with x equal to 2, 3, or 4, then a table is created with a fixed maximum of 448 entries, where one entry is used for each GROUP of statements that all occur together with no labels, branches or conditional expressions. All the statements in such a group are guaranteed to be executed the same number of times. Also, this message occurs if the compressed form of the program occupies more than 65536 bytes of memory (the compressed form is used to generate the listing with the statement counts attached).

II.3 Pass Three Error Messages

All pass 3 errors are disastrous, so compilation terminates immediately. After any pass 3 error, a table of triples, (coordinate number, byte offset, byte length), is listed, indicating how much code was generated for each statement in the current program segment. The last entry of this table and the last two byte lengths are occassionally not meaningful.

3001 PROGRAM SEGMENT OVERFLOW

This error message occurs because of a design constraint of the compiler: the total amount of machine code and constants for any PROCEDURE or other block with declarations must be less than 8192 bytes (a segment of code). All of the constants for a block are allocated in front of the first statement. Therefore, if the byte offset of the first statement is very large, constants are taking up too much space. This sometimes happens in programs with many string constants (ten 80-character string constants take up 800 bytes). It is necessary to reduce the number of statements and/or constants in the block; this can be achieved by introducing new procedures or by inserting at least one declaration into some internal block(s), thereby forcing part of the block that was too big into more than one segment of code.

3002 COMPILER STACK OVERFLOW

A push-down stack, used by the compiler while generating code, has overflowed. A program segment overflow probably was imminent. The remedies suggested in the case of the

message PROGRAM SEGMENT OVERFLOW (3001) apply.

3003 COMPILER LOGIC ERROR

Internal consistency checks performed by the compiler have failed. Take the program listing and (if one exists) matching card deck, exactly as it is, to a consultant.

3004 PROGRAM AREA OVERFLOW

There is insufficient space in memory to contain the compiled program. Re-compile with more memory.

3005 DATA SEGMENT OVERFLOW

The data for each PROCEDURE or BEGIN block with declarations is limited to 4096 bytes. Read the suggestions for 3001.

3006 COORDINATE TABLE OVERFLOW

The table being constructed to supply the coordinate number in run-time error messages has overflowed. Re-compile with more memory.

3007 TOO MANY PROCEDURE CALLS

References to only 63 procedures are allowed within any single procedure.

II.4 Loader Error Messages (XALGOL W only)

Loader error messages are all of the form:

*** LOADING ERROR-message

Like pass 3 messages, these are disastrous and terminate processing.

INSUFFICIENT STORAGE

Insufficient space to load the program. Re-run with more memory.

NO EXECUTABLE STATEMENTS

No main program was loaded, only external procedures.

TOO MANY PROCEDURES

Only 96 program segments are allowed by the loader.

UNDEFINED GLOBAL NAME - XXX

An external procedure was declared, but not loaded.

II.5 Run Time Error Messages

All run error messages are of the form:

RUN ERROR NEAR COORDINATE yyyy IN procedure name - message

After a run error, a post-mortem dump of all of the program variables is produced, unless it is explicitly suppressed with a @DEBUG,0 card. To keep the dump reasonably small, at most eight values are dumped from an array. If the same identifier is declared in many blocks (note that the index variable in a FOR loop is considered to be declared in a block around just the FOR statement), then that identifier will be listed many times. Variables which have never been assigned any meaningful value are printed as "?".

ACTUAL-FORMAL MISMATCH IN PROCEDURE CALL, PARAMETER #xx

The actual parameter passed is not assignment compatible with the formal parameter.

ARRAY SUBSCRIPTING

An array subscript is not within the declared bounds.

ARRAY TOO LARGE

The first n-1 dimensions of an array declaration define too many elements. The product of the first n-1 dimension lengths (upper bound - lower bound + 1) multiplied by the size of a single element must be strictly less than 32768. The element sizes are:

logical	1
integer, real, bits, reference	4
long real, complex	8
long complex	16
string	length of a single string

ASSERTION x FAILED

An assertion is not true, x is a running count of how many prior assertions were true.

ASSIGNMENT TO NAME PARAMETER

Attempt to assign to an actual parameter which is not a variable, but is instead an expression, a constant, or a control identifier.

CASE SELECTION INDEXING

An index in a case statement or case expression is less than 1 or greater than the number of cases.

DATA AREA OVERFLOW

No more storage is left for variables. This can happen if a procedure gets in a loop calling itself recursively, or if there really is not enough memory.

DIVISION BY ZERO

May also be caused by 0**(-n).

EXP ERROR

The argument to EXP must be less than 174.67.

INCOMPATIBLE FIELD DESIGNATOR

An attempt to access a field of a record using a reference which does not designate a record of the corresponding class. (It might be null or undefined).

INCORRECT NUMBER OF PARAMETERS

The number of actual parameters in a procedure call is different from the number of formal parameters declared in the called procedure.

INTEGER OVERFLOW

An integer operation produced a number with an absolute value greater than (2**31)-1.

LENGTH OF STRING INPUT

The string read is longer than the declared length of the receiving string variable. Possibly a quote has been omitted in the data or two adjacent strings in the data have no separating blank causing the double quote to be interpreted as a single quote inside the first string. (Note that quotes in columns 80 and 1 of succeeding cards are adjacent).

LN/LOG ERROR

A negative or zero argument.

LOGICAL INPUT

The quantity read was not TRUE or FALSE.

NULL OR UNDERFINED REFERENCE

An attempt to access a record field using a reference which is null or undefined.

NUMERICAL INPUT

The next data item either is not a correctly formed number or is not assignment compatible with the variable in a READON or READ statement.

OVERFLOW

A real operation produced a number with an absolute value greater than 7.2'+75. This may occur when dividing by a very small number, such as in 1'+50/1'-50.

PAGE ESTIMATE EXCEEDED

The page estimate on the control card (cf. 10 or 11) is exceeded. Note that any tracing (@DEBUG,3 or 4) output is included in this page limit.

PROGRAM CHECK #nn

The compiler or the code it generated is in error. If this happens, take the listing and (if one exists) matching card deck, exactly as it is, to a consultant.

READER EOF

No more data cards. A control card (cf. 10 or 11) was read instead. This is one way to terminate programs, but not a recommended one.

RECORD STORAGE AREA OVERFLOW

No more storage exists for records.

REFERENCE INPUT

References cannot be read.

SIN/COS ERROR

See the domain restrictions in Section 8.2

SQRT ERROR

A negative argument.

STRING INPUT

The next data item is not a correctly formed string.

SUBSTRING INDEXING

The substring selected extends off one end of the string.

TIME ESTIMATE EXCEEDED

The control card time estimate is exceeded (cf. 10 or 11).

UNDERFLOW

A real operation produced a number with an absolute value less than 5.4'-79, but not exactly zero. This may occur when dividing by a very large number, such as in 1'-50/1'+50.