920 CORAL IN OUTLINE

CORAL 66 was developed from CORAL 64 by Currie and Griffiths of the Mathematics Division of R. R. E. (Malvern). The line of development has been towards ALGOL 60 whilst still retaining the useful on-line facilities of CORAL 64. In particular, fixed point working and table manipulation has been retained and the block structure of ALGOL has been incorporated. The present provisional specification is issued by courtesy of the Royal Radar Establishment.

The specification of a programming language down to the finest details of interpretation is not a minor undertaking, but the syntax rules and accompanying comments given here are sufficient to define the language fairly exactly. The reader is assumed to be familiar with ALGOL, and the usual liberties can be taken with the actual choice of basic symbols (the hardware representation). The character set used here is one which happens to be convenient for description purposes.

A program is understood to be one element in a whole system of titled programs sharing a "compool" and having priorities assigned to them for control purposes when more than one program is simultaneously active. Compool declarations are no more restricted than those within a program and apply to all programs within the system, but all these aspects are here regarded as external to the language itself, being part of the general operating system.

In order to economize on storage space, non-dynamic storage allocation is assumed. Unless a computer is well equipped with address modification facilities, dynamic allocation also consumes valuable running time.

In the following syntax rules, class-names are written as single words in block capitals reserved for this purpose alone, and therefore do not need special brackets. The equals sign after a left-hand class name is, of course, outside the language, and hence distinct from the basic symbol = which occurs under STRINGITEM (67) and RELATIONALOPERATOR (60). Alternative expansions for a class name are written on separate lines. The symbol ϕ denotes a void.

(1) BLOCK = begin DECLARATIONLIST STATEMENTLIST end

The body of a program is a block. The end of a block limits the scope of identifiers declared at its head, as in ALGOL. At each entry to a block, simple variables and table items can be initialized dynamically. Though storage allocation is not dynamic, advantage is taken of limited scopes in the implementation, as the working space of successive blocks can be automatically overlaid.

(2) DECLARATIONLIST = DECLARATION ;
DECLARATION ; DECLARATIONLIST

No identifier may be used before it has been completely declared. The scope of an identifier thus extends from the conclusion of its declaration to the end of its block. The rules of scope are otherwise the same as in ALGOL.

(3) STATEMENTLIST = STATEMENT
STATEMENT ; STATEMENTLIST

(4) DECLARATION = NUMBERDECLARATION
PROCEDUREDECLARATION
ARRAYDECLARATION
TABLEDECLARATION
SWITCHDECLARATION
DEFINEDECLARATION

(5) NUMBERDECLARATION = NUMBERTYPE IDENTLIST NUMBERPRESET

(6) NUMBERTYPE = fixed FIXEDSCALE
integer for significance of integer, see (7)

(7) FIXEDSCALE = (INTEGER, SIGNEDINTEGER)

The first integer is the total number of binary digits representing the fixed point number, including a sign digit, and must not exceed the wordlength of the computer (18 bits). The second integer is the number of binary digits after the point i. e. between the binary point and the end of the stored representation. This may exceed the first integer or be negative. Neither integer nor fixed point numbers are closed-packed. The scale of an integer is effectively (18, 0).

(8) IDENTLIST = IDENTIFIER
IDENTLIST, IDENTIFIER

(9) IDENTIFIER = LETTER
IDENTIFIER LETTER
IDENTIFIER DIGIT

(10) NUMBERPRESET = ϕ
$\leftarrow$ NUMBER

An example of a NUMBERDECLARATION would be

fixed (18, 17) x, y, z$\leftarrow$0

Here x, y and z would lie in the range (-1, +1) excluding +1 and would all take initial value zero dynamically at every entry to the block. Variables needing different pre-setting are thus declared separately.

(11) NUMBER = UNSIGNEDNUMBER
ADDOPERATOR UNSIGNEDNUMBER

(12) UNSIGNEDNUMBER = DECIMALNUMBER
$10^{SIGNEDINTEGER}$
DECIMALNUMBER$_{10}$ SIGNEDINTEGER

(13) DECIMALNUMBER = INTEGER
. INTEGER
INTEGER . INTEGER

(14) INTEGER = DIGIT
INTEGER DIGIT

(15) SIGNEDINTEGER = INTEGER
ADDOPERATOR INTEGER

(16) ARRAYDECLARATION = NUMBERTYPE array ARRAYLIST

For example, see (19) below.

(17) ARRAYLIST = ARRAYITEM
ARRAYITEM, ARRAYLIST

(18) ARRAYITEM = IDENTLIST [SIZELIST]

(19) SIZELIST = SIGNEDINTEGER : SIGNEDINTEGER
SIZELIST, SIGNEDINTEGER : SIGNEDINTEGER

This, of course, is where the lack of dynamic storage allocation shows in the language: array bounds must be numerical values. All array items in one array declaration have the same scaling, and each item occupies one whole computer word. Example of ARRAYDECLARATION,

fixed (6, 0) array a, b [1 : 3] , c [1 : 3, 1 : 4, 1 : 4]

Here a and b are simple three-component vectors, and c is a three-dimensional array of 48 elements. All 54 elements are integers with 5 bits plus sign.

(20) TABLEDECLARATION = table IDENTIFIER [INTEGER,INTEGER]
[ENTRYPART]

The table declaration does not exist in ALGOL 60, though it has been proposed for a future ALGOL. The identifier is the name of the table, the first integer is the number of entries in the table, the second integer the number of computer words per entry. For further explanation and example, see (26) below.

(21) ENTRYPART = ENTRYSEGMENT
ENTRYPART ; ENTRYSEGMENT

(22) ENTRYSEGMENT = SEGMENTDESCRIPTION SEGMENTPRESET

(23) SEGMENTDESCRIPTION = IDENTIFIER DESCRIPTION INTEGER, INTEGER

(24) SEGMENTPRESET = ϕ
← CONSTLIST

(25) CONSTLIST = NUMBER
CONSTLIST, NUMBER

(26) DESCRIPTION = (INTEGER)
(INTEGER unsigned SIGNEDINTEGER)
(INTEGER signed SIGNEDINTEGER)

A table is analogous to a one-dimensional array. For example, a table called "squad" might have eight entries, one per man. Each entry can occupy more than one computer word, but each is of the same length, a whole number of words. An entry is made up of packed data segments, and the segments are named. A segment is referenced by its name and a subscript denoting the entry. For example, each entry in squad might have a segment called "height", and height[7] would refer to the height of the seventh man. Notice that this reference to a component of the table "squad" does not involve the identifier squad. An example of a table declaration is:

```
table squad[8, 2]
    [armynumber (18 signed 0) 1, 1;
     age (7 unsigned 1) 2, 1;
     height (9 unsigned 2) 2, 8;
     status (2) 2, 17←0, 0, 0, 0, 0, 0, 0, 0;
     trait (16) 2, 1]
```

Explanation:

8 entries in squad

2 words per entry

18 bit segment for army numbers starting at word 1, bit 1, of entry. Contents of segment to be treated as an integer (i.e. 0 bits after the point), with 17 bits plus sign bit. Note: an unsigned segment of 18 bits could not be handled arithmetically in the 920 computer, since it has an 18 bit word length.

7 bit segment for age, starting at word 2, bit 1, of entry. Contents of segment to be treated as a positive number in the range 0.0 to 63.5 inclusive.

9 bit segment for height, similarly treated.

2 bit segment for status, a bit pattern starting at word 2, bit 17. For explanation of presetting see below.

16 bit segment for trait, to be treated simply as a bit pattern. Note: segments can be defined in an overlapping manner; in this instance "trait" is the union of age and height.

It would be undesirable to permit segments to overlap joins between computer words, and in CORAL 66 this is barred. If a segment of data having DESCRIPTION of type (INTEGER) is used arithmetically, it will be treated as an unsigned integer, i.e.

$$(n) = (n \text{ unsigned } 0).$$

Any segment can be initialized (i.e. preset) dynamically, as shown for status in the example, which would be set to zero for all eight entries. The presetting values would be converted to the appropriate binary form, in this case an unsigned integer.

(27) SWITCHDECLARATION = switch IDENTIFIER←IDENTLIST

This is more restricted than ALGOL, which permits designational expressions in the switch list. Here the list is of labels only.

(28) PROCEDUREDECLARATION =
ANSWERSPECIFICATION procedure PROCEDUREHEADING ;
PROCEDUREBODY

Examples of procedure declarations are given after rules (43) and (73).

(29) ANSWERSPECIFICATION = ϕ
NUMBERTYPE

This is the type of value, if any, assigned to the procedure in its body. See (43).

(30) PROCEDUREHEADING = IDENTIFIER PARAMETERPART

The identifier is the name of the procedure.

(31) PARAMETERPART = ϕ
(PARAMETERLIST)

(32) PARAMETERLIST = PARAMETERSET
PARAMETERLIST ; PARAMETERSET

(33) PARAMETERSET = SPECIFIER IDENTLIST
ANSWERSPECIFICATION procedure PROCLIST
TABLESPECIFICATION

Unlike the formal parameter list in ALGOL, the specification of parameters is included with the formal parameters where these first appear, and not in a separate list afterwards. For the detailed expansion of syntax of PROCLIST, see (70).

(34) SPECIFIER = CALLTYPE NUMBERTYPE
NUMBERTYPE array
label
switch

(35) CALLTYPE = value
location

These declarations describe the type of parameter call for fixed and integer type numbers. The CALLTYPE value is the standard ALGOL "call by value". The CALLTYPE location is what is sometimes known as "address by value", i.e. it is suitable for output from the procedure, but occurrences of the formal parameter in the procedure body do not repeatedly refer back to the actual parameter for re-evaluation. Arrays and tables are called in this way without option.

(36) TABLESPECIFICATION
= table IDENTIFIER [INTEGER, INTEGER] [SEGMENT LIST]

For explanation, compare (20) and see below.

(37) SEGMENT LIST = SEGMENT DESCRIPTION
SEGMENT LIST ; SEGMENT DESCRIPTION

Note the similarity between SEGMENT LIST and ENTRYPART (21), the only difference being the absence of SEGMENTPRESET.

(38) PROCEDUREBODY = COMPOUNDSTATEMENT
BLOCK
ANSWERSTATEMENT

Note the differences from ALGOL and see (43) for assignment of a value to the procedure. The procedure body is like a block rather than a compound statement, e.g. it is not permitted to jump into it from outside. Rule (39) is void.

(40) . COMPOUNDSTATEMENT = begin STATEMENT LIST end

(41) STATEMENT = FREESTATEMENT
FORSTATEMENT
IFSTATEMENT
IDENTIFIER : STATEMENT

(42) FREESTATEMENT = ANSWERSTATEMENT
ASSIGNMENTSTATEMENT
GOTOSTATEMENT
PROCEDURECALL
CODESTATEMENT
COMPOUNDSTATEMENT
BLOCK
ϕ

(43) ANSWERSTATEMENT = answer EXPRESSION

When a procedure declaration starts with an answer specification which is not void, the procedure body (38) is either an answer-statement alone or has an answer-statement immediately before its end. The expression will be evaluated to the specified type and assigned as the value of the procedure. This arrangement seems more satisfactory than the exceptional assignment statement in ALGOL. The following is an example of a procedure declaration with assorted parameters:

```
fixed (12, 5) procedure
        example (value fixed (12, 5) a, b, c;
                 location fixed (6, 5) x;
                 location integer i, j;
                 fixed (12, 5) array g, h;
                 integer array k;
 1               label s1, s2, s3;
                 table t [16, 1] [head(9) 1,10; tail (9) 1, 1] );

        begin    integer p, q;
                 STATEMENTLIST;
                 answer a + b + c

        end
```

(44) FORSTATEMENT = for IDENTIFIER←FORLIST do STATEMENT

The identifier is a simple arithmetic variable. It is not permitted to jump to or into the controlled statement from outside. The value of the controlled variable upon exit by exhaustion is undefined. Upon exit by jumping out, its value is what it was at the time. The further sub-division of the syntax follows much the same pattern as in ALGOL except that the expressions in the for-list are called by value as soon as each element of the for-list is reached.

(45) FORLIST = FORELEMENT
FORELEMENT, FORLIST

(46) FORELEMENT = EXPRESSION
EXPRESSION step EXPRESSION until EXPRESSION
EXPRESSION while BOOLEANEXPRESSION

The expression or expressions in a for-element (except the boolean) are all evaluated immediately before the for-statement is obeyed for that for-element. No further reference to the original expressions is made after this.

(47) EXPRESSION = TERM
EXPRESSION ADDOPERATOR TERM
ADDOPERATOR TERM

(48) ADDOPERATOR = +
-

(49) TERM = FACTOR
TERM MULTOPERATOR FACTOR

(50) MULTOPERATOR = x
/

(51) FACTOR = PRIMARY
FACTOR ↑ PRIMARY

(52) PRIMARY = (EXPRESSION)
VARIABLE
UNSIGNEDNUMBER
PROCEDURECALL

The procedure call must refer to a procedure with an answer part.

(53) VARIABLE = IDENTIFIER
IDENTIFIER [EXPRESSIONLIST]

The expression list is the list of subscripts. Arithmetic variables are those which occur in (52) and (62), and they refer to numbers, array elements or segments of a table entry. The variable occurring in (63) is a "designational variable" which is a label or a switch.

(54) EXPRESSIONLIST = EXPRESSION
EXPRESSIONLIST, EXPRESSION

(55) PROCEDURECALL = IDENTIFIER
IDENTIFIER (EXPRESSIONLIST)

When a formal parameter is substituted by value, the FIXEDSCALE of the actual parameter, where this is a variable, need not agree with that specified for the formal parameter. An assignment takes place, which looks after conversion of scaling. Where the formal parameter is substituted by location, the scaling must agree.

(56) IFSTATEMENT

= if BOOLEANEXPRESSION then LABFREESTATEMENT
if BOOLEANEXPRESSION then LABFREESTATEMENT else STATEMENT

(57) BOOLEANEXPRESSION = BOOLEANEXPRESSION or BOOLEANONE
BOOLEANONE

(58) BOOLEANONE = BOOLEANONE and BOOLEANTWO
BOOLEANTWO

(59) BOOLEANTWO = EXPRESSION RELATIONALOPERATOR EXPRESSION

A "booleantwo" cannot be bracketed. All brackets within boolean expressions are those around or within the arithmetic expressions on each side of the relational operator.

(60) RELATIONALOPERATOR = $<$
$>$
$\leq$
$\geq$
$=$
$\neq$

(61) LABFREESTATEMENT = FREESTATEMENT
IDENTIFIER : LABFREESTATEMENT

The identifier is a label.

(62) ASSIGNMENTSTATEMENT = VARIABLE ← EXPRESSION
VARIABLE ← ASSIGNMENTSTATEMENT

The expression is evaluated in an undefined sequence with undefined intermediate scaling and is converted to the scaling of the variable on the left-hand side before assignment. Where there are two or more left-hand sides, their NUMBERTYPE must agree.

(63) GOTOSTATEMENT = goto VARIABLE

The variable, syntactically similar to a simple or subscripted arithmetic variable, is here a label or a switch.

(64) CODESTATEMENT = code begin CODESTREAM end

The stream of machine code is in a form similar to SIR. The identifiers within the CODESTREAM refer exactly to the corresponding language items.

(65) DEFINEDECLARATION = define IDENTIFIER as ⟨STRING⟩

Within the scope of the declaration, the identifier is supposed to be replaced by the string whenever it occurs.

(66) STRING = STRINGITEM
STRING STRINGITEM

(67) STRINGITEM = LETTER
DIGIT
+ - x / .10 < > ≤ ≥ _ = ≠ ↑ , : ; ( ) [ ]
⟨STRING⟩

The symbols are shown on one line to save space. The underscore is assumed to be "non-escaping".

(68) 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

(69) DIGIT = 0 1 2 3 4 5 6 7 8 9

(70) PROCLIST = PROCITEM
PROCLIST, PROCITEM

This rule continues from (33) the syntactic description of formal procedure parameters which are themselves procedures. For an example, see below (73).

(71) PROCITEM = IDENTIFIER (PROCSPECLIST)

An illustration of a procitem is indicated in the example below (73).

(72) PROCSPECLIST = PROCSPECITEM
PROCSPECLIST, PROCSPECITEM

(73) PROCSPECITEM = SPECIFIER
table
ANSWERSPECIFICATION procedure

Unlike ALGOL, a formal parameter of a procedure which is itself the name of a procedure must be accompanied by a specification of its own parameters, but as these occur only as actuals in the procedure body, they do not have to be named in the PROCSPECLIST. The following is an example of a PROCEDUREDECLARATION containing parameters of procedure type.

To show the difference between the specification of parameters in a procedure declaration with those in a procedure parameter, outragel and outrage2 have been assumed to have parameters of identical types. The procedure heading is clumsy, though not outrageously so. Procedure type parameters are not a common occurrence in ordinary programs, and when they do occur, their own parameters would probably be simpler than are shown in this example.

```
fixed (12, 0) procedure

outragel (location integer i, j;
          value fixed (12, 0) x;
          fixed (12, 0) procedure

                outrage2  (location integer,          )
                          location integer,           )
                          value fixed (12, 0),        )
                          fixed (12, 0) procedure,    )  PROCITEM;
                          procedure,                  )
                          procedure);                 )

          procedure p (value fixed (12, 6)), q (value fixed (12, 6) ) ) ;
PROCEDUREBODY
```

Implementation

920 CORAL is being implemented using a new technique of compiler generation. This technique allows modifications to the language to be implemented very quickly and cheaply. Some suggested modifications are listed below.

The compiler requires 16K words of store although the object code it produces will run on any size 920.

The following facilities are included although they are not mentioned in the syntax.

Octal constants, e.g. octal ( )
String macro facility with parameters
A limited form of recursion

String handling and own variables may be added later if the need arises.

Summary

More than ALGOL	Tables Fixed point scaling String macro facility Presetting
Less than ALGOL	No dynamic arrays No conditional expressions No Boolean variables No Boolean brackets Fewer Boolean operators No recursion yet
Different from ALGOL	For-elements called by value Controlled for-variable never subscripted Parameter specification different Calls of parameters by value or location

INDEX OF SYNTAX RULES