This section describes certain features of KDF9 ALGOL which are not part of the Revised ALGOL 60 language. It should be noted that throughout this manual, KALGOL and WALGOL are abreviations for Kidsgrove ALGOL and Whetstone ALGOL respectively.

2.1 USERCODE PROCEDURES

This section gives, by means of formal syntax, and informal semantics, (to be read as the required expansion of section 5.4.6 of the Revised ALGOL 60 Report), a set of rules for procedure bodies in code. The code given is KDF9 USERCODE as described in the KDF9 programming manual.

2.1.1 Syntax

The specification part for code procedures is restricted as follows:-

<specifier> :: = string|<type>|array|<type>array|label

The procedure body is made up as follows:-

<description> ::= <number of nests used> / <number of Q-stores used> /
<number of SJNS used> / <number of V-stores used>

<instruction> ::= <KDF9 USERCODE instruction> | EXIT |
'<formal parameter>' | = '<formal parameter>' |
J'<formal parameter>'

<instruction sequence> ::= <instruction> | <instruction sequence>;
<instruction>

<code> ::= KDF9 <description>; <instruction sequence>; ALGOL;

2.1.2 Description

The description gives information to the translator concerning storage requirements of the code procedure.

Q-stores will be used from Q15 downwards. Q1, Q2, Q3, Q4 will not be available.

The use of the W-stores is not allowed. The number of SJNS used is limited to 14 cells, and the maximum number used is entered in this part of the description.

The number of the highest numbered V-store belonging to the code body must be given, for use later by the USERCODE compiler. V-stores set in the code body belong to that body regarded as a USERCODE subroutine. They preserve their values between calls; they may be regarded as analogous to "own" variables.

If no V-stores are used zero must be entered

2.1.3 Instructions

USERCODE procedure bodies will be written in the style of P-subroutines in that each will have its own V-stores and labels. The label Pp or PpVv will be provided by the ALGOL compiler, not by the author of the procedure. The procedure body is delimited by the special symbols KDF9 and ALGOL. USERCODE EXIT instructions must be distinguished; those ending the procedure dynamically must be written underlined, as EXIT; those occurring internally must be written normally, as EXIT. USERCODE library subroutines may be called. "Private" P-routines may not be used.

If library subroutines (L-routines) are used they must be requested in the normal way by the use of library. This request must follow the last instruction of the code body (subroutines in USERCODE may not be nested). A code body may refer to such routines in another code body without restriction.

Any library routine used in this way must of course comply with the rules given above.

Communication between a code body and its ALGOL environment will be solely through the parameter list. Since this scheme is being implemented on two entirely different translators, code procedures must not use any internal properties of a translator or of the translated ALGOL program that it has produced. In particular the use of Y,YA..YZ stores is prohibited except in the case of Y-addresses, which are modified by a modifier which is derived from an array parameter. Such Y addresses are limited to the range ± 1200.

The formal parameters can be used in a code procedure in fetch, store, and unconditional jump instructions, which will be translated into one or more USERCODE instructions.

If necessary, in the case of <type> or label parameters called by name, the translator will preserve any necessary nests or Q-stores or SJNS locations so that parameters can be used regardless of the complexity of their corresponding actual parameters.

2.1.4 Calls of Formal Parameters by Name

Type Parameters. The instruction 'a' (That is, a formal parameter enclosed in string quotes) will cause the corresponding actual parameter to be evaluated and the value to be placed in N1. Real/Integer conversions will be performed.

The instruction = 'a' will cause the address of the corresponding actual parameters, which could be a simple or a subscripted variable, to be evaluated, and N1 to be stored at this address. Real/integer conversions are not allowed.

Label Parameters. The instruction J 'a' will cause the corresponding designational expression to be evaluated, and the code procedure to be left to return to the resulting label.

Array Parameters. The instruction 'a' will load N1 with the appropriate array word. An array word contains addressing information for an array.

(a) Arrays are stored by columns.

For example: $\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix}$ is stored in successive words as

a_{11}	a_{21}	a_{12}	a_{22}	a_{13}	a_{23}	a_{14}	a_{24}

(b) Each array has associated with it an additional word which contains information about where the array is stored and where the information is stored which enables the array to be indexed. This additional word is the word which is loaded into the formal location.

(c) This word is in three parts corresponding to the counter, increment and modifier divisions of a Q-store.

The counter position contains the address (relative to Y0) of $A(l_1, l_2 \ldots\ldots l_n)$ where l_i are the values of the lower bounds in the array declaration.

The increment position contains the address (relative to Y0) of the start of the index information. (dope vector)

The modifier position contains the address (relative to Y0) of $A(0, 0 \ldots\ldots 0)$. That is,

$C - \sum_{i=1}^{n} l_i . \Delta_i$ where C is the address given in the counter position as defined above, and Δ_i is defined below.

(d) The Δ_i are defined as follows, where u_i are the values of the upper bounds in the array declaration.

$\Delta_1 = 1$

$\Delta_2 = (u_1 - l_1 + 1) . \Delta_1$

$\Delta_n = (u_{n-1} - l_{n-1} + 1) . \Delta_{n-1}$

$\Delta_{n+1} = (u_n - l_n + 1) . \Delta_n$ = total number of elements.

The index information ("Dope Vector") is first the total number of elements for the array, and secondly, the list of the n-1 increments required for a step of 1 in any given suffix position. The step for suffix position 1 is always 1 (stored by columns).

The Dope Vector then is as follows:-

$DV_0 = \Delta_{(n+1)}$

$DV_1 = \Delta_2$

$\vdots$

$DV_{n-1} = \Delta_n$

the Δ_i are located as the least significant 16 bits of the DV_{i-1} element.

The other 32 bits of the DV_i words are used for information to the translator - this information is not always there and cannot be used by the writer of a code body - it does, however, mean that the Δ_i must be 'masked' out of the DV_{i-1} word.

(e) It will be seen, therefore, that $C - \sum_{i=1}^{n} l_i.\Delta_i$ is the address relative to which a suffix can be evaluated in a general manner. That is, $A(i_1\ i_2\ .....i_n)$ is given by $M + i_1.\Delta_1 + ... + i_n.\Delta_n$, where M is the value of the modifier position, as defined above.

For example: $\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix}$ gives rise to the following information

index: $\Delta_1 = 1$
$\Delta_2 = (2 - 1 + 1).\ 1 = 2$
$\Delta_3 = 8$

and $DV_0 = 8$
$DV_1 = 2$

(f) DV_0 (the number of elements) may not necessarily be positive, but its absolute value is the number of elements in the array.

String Parameters. The instruction 'a' will load N1 with the address, relative to EØ, of the opening string quote. The basic symbols of the string (including its bounding string quotes) are stored as 8-bit characters according to the table in Appendix C, 6 per word, beginning with DØ-7 of the word whose address is given.