7. PROCEDURES

7.1. INTRODUCTION

A procedure in ALGOL is used to specify an independent section of a program (which usually represents an algorithm) that can be called or executed at different points throughout the same program or may be used in other programs. The operations to be performed are fixed, but a list of parameters makes it possible for a procedure to be used with varying values and/or variables.

A procedure must be declared in the declaration part of the block in which the procedure is referenced. More than one procedure may be defined at the beginning of a block. During program execution when a block is entered, the first statement executed is the first executable statement following the procedures (if any).

The procedure declaration consists of a procedure heading and a procedure body. The heading consists of a procedure identifier, a formal parameter list, if any, a value list, if any, and specifications, if any. The procedure body follows the specifications and consists of a statement, compound statement, or a block.

Example:

```
PROCEDURE NFACT (ARG1, ARG2) $
          INTEGER ARG1, ARG2 $
          BEGIN
              INTEGER I $
              ARG2 = 1 $
              FOR I = 1 STEP 1 UNTIL ARG1 DO
                  ARG2 = ARG2*I$
          END
```

In the above example NFACT is the identifier for a procedure that calculates the value of N factorial. ARG1 and ARG2 are the formal parameters (arguments) for the procedure. ARG1 is the number whose factorial is to be calculated, and ARG2 is the result after the procedure has been executed. ARG1 and ARG2 are **INTEGER** variables. The **BEGIN–END** pair sets off the body of the **PROCEDURE.** The **BEGIN–END** pair can be dropped if the procedure body is just one statement. Since I is declared as an **INTEGER,** it is local to NFACT.

A "call" of the above procedure would be of the form:

```
NFACT (N,FACT) $
•
•
•
NFACT (N1,FACT1) $
```

In the first call, the actual parameters N and FACT are substituted for the formal parameters ARG1 and ARG2. Later the parameters N1 and FACT1 are substituted for ARG1 and ARG2 in the same fashion. Thus procedure is a closed subroutine, and the call establishes a linkage to the subroutine.

An alternate form of the parameter list allows comments to be inserted between the formal parameters since the comma separating formal parameters is equivalent to:
) < string > : (

```
PROCEDURE NFACT (ARG1, ARG2)
```

could be written as:

```
PROCEDURE NFACT (ARG1) AND STORE RESULT IN: (ARG2) $
```

or

```
PROCEDURE NFACT (ARG1) ARG1 INPUT AND ARG2 OUTPUT: (ARG2) $
```

The following is a procedure for the multiplication of two matrices:

```
PROCEDURE MATMUL (A,B,C,N,M,P) $
REAL ARRAY A,B,C, $
INTEGER N,M,P $
BEGIN INTEGER I,J,K $
REAL TEMP $
FOR I=1 STEP 1 UNTIL N DO
FOR J=1 STEP 1 UNTIL P DO
BEGIN TEMP=0.0 $ FOR K=1 STEP 1 UNTIL M DO
TEMP=TEMP+B(I,K)*C(K,J) $
A(I,J)=TEMP END I,J LOOP
END MATMUL $
```

A procedure statement calls for the execution of a procedure body.

Given the declaration

```
REAL ARRAY A1(1:10,1:7), A2(1:10,1:15), A3(1:15,1:7) $
```

then the procedure statement

```
MATMUL (A1,A2,A3,10,15,7) $
```

has the effect of multiplying the two matrices A2 and A3 and storing the results in A1.

Expressions may also be used as actual parameters. Care must be taken to match the type and kind of each formal and actual parameter in any call.

7.2. VALUE ASSIGNMENT (CALL BY VALUE) AND NAME REPLACEMENT (CALL BY NAME)

The above procedure NFACT makes use only of the value of ARG1 whereas it changes the value of the actual parameter which replaces ARG2. Thus NFACT could be rewritten as follows:

```
PROCEDURE NFACT (ARG1, ARG2)
   VALUE ARG1 $
   INTEGER ARG1, ARG2 $
   BEGIN
       INTEGER I $
       ARG2 = 1 $
       FOR I = 1 STEP 1 UNTIL ARG1 DO
           ARG2 = ARG2*I
       END$
```

The procedure statement

```
NFACT (NUMBER,FACTORIAL) $
```

has this effect: the value of the actual parameter, NUMBER, replaces ARG1 when the procedure statement is encountered and NFACT does not have access to the location assigned to NUMBER. ARG1 is known as a < Call by value> parameter. A value parameter must also have a type specification and cannot appear to the left of the replacement operator in the procedure.

Any parameter (such as ARG2) which is not listed in the **VALUE** part of the procedure declaration is said to be a <Call by name> parameter. The name FACTORIAL replaces the name ARG2. The value of FACTORIAL is changed as the procedure is executed.

In the following examples, numeric values or expressions are used as actual parameters:

```
NFACT (15,FACT1) $
NFACT (J+K+M,FACT2) $
```

It should be noted that a value parameter which is an array or string identifier requires that the entire array or string supplied by the procedure call be copied locally within the procedure. As a result large amounts of working storage may be used unexpectedly when the procedure is called. All calculations in the procedure use that temporary copy. As an example, suppose it is necessary to find the determinant of a matrix without destroying the matrix. The usual computational methods for finding determinants destroy the matrix with which they are working. Thus the original matrix must be copied somewhere. Specifying the array as **VALUE** accomplishes this:

```
REAL PROCEDURE DET(A) SQUARE MATRIX WHOSE DIMENSION IS:(N)$
VALUE A,N $
REAL ARRAY A $
INTEGER N $
BEGIN
(STATEMENTS)
DET=... END DET $
```

This is an example of a function procedure explained in 7.4. If an expression is used as an actual parameter, and if the parameter is called by name, then the expression is reevaluated at each occurrence of the formal parameter in the procedure.

7.3. SPECIFICATIONS

The <type> of all formal parameters defined by a procedure declaration must be specified in the specification part of a procedure heading. The format of the specification part is as follows:

<specification> <identifier list>;

The specification may be in any one of the following forms:

<type>

ARRAY

<type> **ARRAY**

PROCEDURE

<type> **PROCEDURE**

LABEL

SWITCH

FORMAT

LIST

and< type> is one ALGOL type: **INTEGER, REAL, REAL 2, BOOLEAN, COMPLEX,** or **STRING**

The <identifier list> consists of the formal parameter identifiers contained in the procedure declaration separated by commas.

The reason that all formal parameters must be specified is that the compiler must know the type and kind or class of all parameters in order to compile proper machine code.

Examples:

```
INTEGER I, K ;
REAL X, Y ;
REAL ARRAY Z ;
BOOLEAN PROCEDURE F ;
STRING S ;
```

Specifications do not include information about lengths of strings, the dimensions and bounds of arrays, the formal parameter parts of procedures, or the contents of formats and lists. The actual declarations of these exist elsewhere in the program. The details of constructing a procedure can be illustrated by an example:

```
PROCEDURE TRANSPOSE (A) ORDER:(N) $
    VALUE N $            (A, N)
    ARRAY A $
    INTEGER N $
    BEGIN
       REAL W $
       INTEGER I, K $
       FOR I = 1 STEP 1 UNTIL N DO
       FOR K = 1+I STEP 1 UNTIL N DO
           BEGIN
              W = A(I,K) $
              A(I,K) = A(K,I) $
              A(K,I) = W
           END
    END TRANSPOSE$
```

7.4. FUNCTION PROCEDURES

Procedures which are to be used as functions (e.g., SIN, EXP) must have a type associated with the procedure identifier (i.e. procedure name). This type declaration must be the first symbol of the procedure declaration. Also for the function procedure to have a value associated with it, the procedure identifier must occur at least once as the left part of an assignment statement in the procedure body. In addition, at least one of these assignment statements must be executed on a given procedure call for a value to be assigned to the procedure. If more than one such assignment statement is executed within the body, then the last one executed before exiting from the procedure determines the value associated with the procedure. Any other occurrences of the procedure identifier within the body of the procedure are considered as recursive calls on the procedure.

The procedure NFACT could be written so that the only parameter would be N and the value of NFACT would be N factorial.

```
INTEGER PROCEDURE NFACT (ARG) $
   INTEGER ARG $
   BEGIN
       INTEGER I, TEMP $
       TEMP = 1 $
       FOR I = 1 STEP 1 UNTIL ARG
           DO TEMP = TEMP*I$
       NFACT = TEMP
   END NFACT $
```

The call for the above procedure would be of the form

```
COMMENT SET FACT = N FACTORIAL$
FACT = NFACT(N) $
```

A function procedure is referenced by a function designator which defines a single numerical or logical value. NFACT(N) is a function designator which will have an integral value and can thus be used in any expression in which an integer variable could be used.

7.5. RECURSIVE PROCEDURES

In the example above, a new variable TEMP was used to store the intermediate result of the calculation of N factorial. Then the extra statement NFACT = TEMP was needed to give NFACT the proper value. The reason for this is that inside the procedure body, whenever the name of the procedure occurs on the left-hand side of an assignment statement, it is a <procedure assignment> statement, that is, the statement which assigns the value to the procedure. Wherever else the name occurs, it is a call on the procedure.

This kind of construction can be used to produce another version of NFACT which is even simpler to write. In fact, it requires only one statement in the procedure body:

```
INTEGER PROCEDURE NFACT(N) $
INTEGER N $
NFACT = IF N EQL 0 THEN 1 ELSE N*NFACT(N-1) $
```

which is equivalent to the recursive definition

$$\text{factorial}(n) = 1 \qquad n = 0$$
$$= n*\text{factorial}(n-1) \qquad n > 0$$

A procedure call with the actual parameter 4 (FACT = NFACT(4)) has the following effect. After the subroutine linkage is set up, the procedure body is virtually changed to:

```
NFACT=IF 4 EQL 0 THEN 1 ELSE 4*
NFACT (3)
```

NFACT(3) is another call of the functional procedure having the result that the call is replaced with the procedure body:

```
NFACT=IF 4 EQL 0 THEN 1 ELSE 4 *
   (IF 3 EQL 0 THEN 1 ELSE 3*NFACT(2))
```

This produces another call on NFACT resulting in another change of the statement. This goes on until finally:

```
NFACT=IF 4 EQL 0 THEN 1 ELSE 4*
(IF 3 EQL 0 THEN 1 ELSE 3*(IF 2
EQL 0 THEN 1 ELSE 2* (IF 1 EQL 0
THEN 1 ELSE 1*(IF 0 EQL 0 THEN 1
ELSE 1* (NFACT(0)))))) $
```

The process is terminated when 0 **EQL** 0 occurs in the relation of the conditional expression. The usual expression for the factorial is obtained after the unnecessary parts of the above statement have been removed:

4 factorial=4*3*2*1

All procedures written in ALGOL may be called from within themselves. But it should be mentioned that recursive procedures are not always put to good use. For example, using the recursive properties of procedures makes a much neater looking NFACT, but also a much less efficient one. However, recursive procedures may have practical uses. For example, multiple integration programs use a quadrature procedure to evaluate the inner function as well as to do the integration. (See ACM Algorithm 233 ''Simpson's Rule for Multiple Integration,'' Communications of the ACM Vol. 7, No. 6, June 1964.)

7.6. EXTERNAL PROCEDURES

External procedures are procedures whose bodies do not appear in the main program. They are compiled separately and linked to the main program at its execution. The **EXTERNAL** declaration serves the purpose of informing the compiler of the existence of these procedures, their types (if any), and the proper manner to construct the necessary linkages. The general form of the external declaration is:

EXTERNAL <kind> <type> **PROCEDURE** <identifier list>

where <type> is the arithmetic type or is empty, <identifier list> is a list of identifiers of external procedures, and

<kind> ::= empty/FORTRAN/NON–RECURSIVE

The words 'FORTRAN' and 'NON–RECURSIVE' have special significance only in this context. Procedures of kind <empty> are ALGOL procedures and are treated exactly like an ordinary procedure declared within the program. However, they need not be written in ALGOL language. Procedures of kind 'FORTRAN' are FORTRAN subroutines or functions and procedures of kind 'NON–RECURSIVE' are necessarily written in machine language. In the following paragraphs we assume a knowledge of the UNIVAC 1108 Executive System, FORTRAN (see 7.6.2), and the UNIVAC 1108 Assembler (see 7.6.3).

7.6.1. ALGOL External Procedures

An ALGOL program which consists entirely of a procedure is nonexecutable because it contains only a procedure declaration (see 7.1.). When such a program is compiled, the name of the procedure is marked as an entry point when the program is entered into the program file. Like all names in the program file the first six characters of the procedure name must define it. Such a procedure may be referenced from another ALGOL program as an external procedure.

Example:

```
PROGRAM 1

BEGIN REAL PROCEDURE DET(A,N)$
REAL ARRAY A $
INTEGER N $
VALUE A,N $
BEGIN
COMMENT THIS PROCEDURE FINDS THE DETERMINANT OF A REAL N BY N
MATRIX A, LEAVING A UNCHANGED AND ASSIGNING THE VALUE TO DET $
•
•
DET =-. . . END DET
END $

PROGRAM 2

BEGIN REAL ARRAY MATRIX (1:10,1:10) $
      EXTERNAL REAL PROCEDURE DET $
      •
      •
WRITE(DET(MATRIX,10)) $
      •
      •
END $
```

A user could build a library of procedures that are useful to him and then refer to whichever he needed by merely declaring them as external procedures in his main program.

7.6.2. FORTRAN Subprograms

A FORTRAN subroutine or a FORTRAN function may be made available to an ALGOL program by the declaration:

EXTERNAL FORTRAN < type > **PROCEDURE** < identifier list >

Actual parameters in calls on such procedures may be either expressions or arrays. (Labels, string expressions, and string arrays are specifically excluded.) The FORTRAN subprogram is a subroutine or function depending on the absence or presence of < type > in the external declaration. A FORTRAN function is used like an ALGOL functional procedure i.e., as an expression. For example, if DET (above) were a FORTRAN subroutine:

```
PROGRAM 1

SUBROUTINE DET(A,N,D)
DIMENSION A(N,N)
C  DET FINDS THE DETERMINANT OF A REAL NXN
C  MATRIX A AND LEAVES THE RESULT IN D,
C  DESTROYING A.
   •
   •
   D=...

END
```

```
PROGRAM 2

BEGIN REAL ARRAY MATRIX(1:10,1:10) $
   REAL DETVALUE $
   EXTERNAL FORTRAN PROCEDURE DET $
   •
   •
   DET(MATRIX,10,DETVALUE) $
END $
```

7.6.3. Machine Language Procedures

A procedure written in 1108 Assembler language may be referenced in either of two ways. The more difficult manner occurs when the procedure is declared exactly as an ALGOL external procedure. In this case the Assembler procedure must behave like an ALGOL procedure (that is, it must be able to handle recursive calls). Here the nonrecursive case is considered. The form of the declaration for these is:

EXTERNAL NON–RECURSIVE<type> **PROCEDURE** <identifier list>

To understand how to write such procedures consider the coding produced by the ALGOL compiler as the result of a call in the following program:

```
EXTERNAL NON-RECURSIVE PROCEDURE PUNCH $
INTEGER Q, S $

PUNCH(Q,S) $
```

The statement PUNCH (Q,S) results in the four lines of coding:

```
LMJ                  11,PUNCH
+                    2
F                    00,01,01,Q
F                    00,01,01,S
```

The second line states the number of parameters being handed through and the following lines provide information about each parameter in turn. The actual form of F is defined elsewhere in the system by a FORM directive

```
F  FORM 6, 3, 3, 24
```

which specifies the number of bits in each field of F. (See "UNIVAC 1108 Assembler Programmers Reference Manual," UP-4040.)

The four fields of F are defined and encoded as follows:

KIND

00 = Expression
10 = Array

TYPE

01 = INTEGER
02 = REAL
03 = COMPLEX
04 = BOOLEAN
05 = STRING
06 = REAL 2

REFERENCE

00 = Constant
01 = Name
02 = Indirect
06 = Result

LOCATION

The location field specifies the location of the parameter. Indirect addressing may be specified and index register 11 is usually designated in this 24–bit field. With this in mind, the following rules should be followed in writing an Assembler procedure:

(1) The return point for a call with N parameters is (X11)+N+1.
(2) The value of the procedure (if any) must be left in register A2 (and A3 for **COMPLEX** and **REAL** 2), absolute 14.
(3) Registers 1–4 may not be used without saving and restoring.
(4) Register 10 must never be destroyed.
(5) The Ith parameter should be referenced by an indirect command, e.g.,

```
LA   A2,*I,X11
```

If the parameter under reference is a double-word quantity (**COMPLEX** or **REAL** 2) its second half is in the next location, and the parameter should be referenced with a DL A2,*I,X11

```
DL      A2,*I,X11
```

(6) When using an arithmetic or a Boolean expression, the word referenced in rule (5) is the value of the expression.
(7) When using either an arithmetic or a Boolean array, the word referenced in (5) is itself the address of the following information:

number of elements	, address of first element
precision	, N
lower bound (N)	, length (N)
	.
	.
lower bound (1)	, length (1)

where N is the number of dimensions of the array, lower bound (I) is the value of the lower bound of subscript position (I) and length (I) is its length.

(8) When using a string expression, the word referenced in (5) is a string descriptor of the form:

```
F     FORM    12,6,18
```

```
F     length, start, address
```

'length' is the length of the string which starts at character position 'start' of the word at 'address.' For this purpose 'start' is coded as 0–5 to select a sixth of a word, S1–S6.

(9) When using a string array, the word referenced in (5) consists of two addresses, the left one being the address of a string descriptor and the right one the address of the array information (as in 7). The address field of the string descriptor must be added to the first element address to find the first element of the string array. Both arrays and string arrays are stored by column.

(10) The name of the external procedure must be the entry point of the Assembler procedure.