PROC's

PROC ia a directive which is used in conjunction with the END directive to delineate a structure of a symbolic code. When referenced, this delineated structure of code will be generated by UTMOST at the point of reference.

The term PROC is short for procedure. The structure of code delineated by the PROC is a procedure which functions at object time, i.e. when the program containing it is being run. This will be called the object procedure.

The process of using a PROC and an END directive to delineate code and the process of referencing the PROC so that generation might take place is also a procedure. This procedure is active at coding and assembly times. This is the procedure which is the subject of the following text.

THE PROCEDURE: SECTION I

A. GENERAL

Coding is a tedious and exacting function with which the programmer must contend. It becomes quickly apparent to a coder that every so often certain sequences of instructions become repetitive. It becomes apparent that these sequences are exactly alike except for a few minor changes. Consider the following compare operation:

EXAMPLE 1: STANDARD SYMBOLIC CODE

```
A.  *LOAD . REGISTER   WITH X
     COMPARE           TO   Y
     JUMP . EQUAL      TO   B
     JUMP . UNEQUAL    TO   OUT

B.   LOAD REGISTER     WITH X + 1
     COMPARE           TO   Y + 1
     JUMP . EQUAL      TO   C
     JUMP . UNEQUAL    TO   OUT

C.   LOAD REGISTER     WITH X + 2
     COMPARE           TO   Y + 2
     JUMP . EQUAL      TO   D
     JUMP . UNEQUAL    TO   OUT

D.   ET CETERA
```

It can be seen that A, B, C, D are repetitive. They are exactly alike in structure, slightly different in address references. Coding such as this becomes clerical and subject to clerical error. If the programmer could write "COMPARE X TO Y" and be assured that this could generate the LOAD . REGISTER and the COMPARE instructions and if the programmer could write "$ + 2 or OUT" ($ signifies the current address) and the first address given ($ + 2) was predetermined to be the JUMP . EQUAL operand and the second address (OUT) was the JUMP . UNEQUAL operand, then programmer would be more content with coding the above example as follows:

EXAMPLE 2: PROCEDURE REFERENCE CODING

```
A.  COMPARE   X      TO  Y       $ + 2 OR OUT
B.  COMPARE   X + 1  TO  Y + 1   $ + 2 OR OUT
C.  COMPARE   X + 2  TO  Y + 2   $ + 2 OR OUT
D.  ET CETERA
```

Example 2 demonstrates reference lines which would produce the structure of code described in Example 1. Normally, the programmer would only write this reference in his program; the structure of code, or the object procedure itself, would exist in a library. If it did not exist, the programmer would have to write his own and prefix it to his symbolic assembly deck.

It would look like this:

EXAMPLE 3: GENERAL FORMAT OF A PROCEDURE

```
COMPARE  PROC
         LA    8, Compare (1,1) (Load Register with X)
         C     8, Compare (3,1) (Compare to Y)
         JE    Compare    (4,1) (Jump equal to $ + 2)
         J     Compare    (6,1) (Jump unequal out)
         END
```

The basic structure is delineated with a PROC and an END line. The label of the PROC identifies it. A parameter reference form ''Compare (n,e)'' indicates by n,e what given expression in the reference line will replace it at generation time. Generation occurs when the assembler encounters the reference line. Every reference line in Ex. 2 will cause the four symbolic lines delineated in Ex. 3 to be appropriately modified and generated in place of the reference line. The result would be equivalent to Example 1.

B. DEFINITIONS

1. PROC is a directive which when used in conjunction with the END directive delineates a structure of symbolic code. The PROC structure must appear physically before the reference in the program.

2. A procedure is a method employed by the UTMOST assembler to allow the automatic generation and modification of a structure of code defined by a PROC. Generation occurs when the reference line is encountered.

3. A Reference Line is a symbolic line of code which employs the label of a Procedure for its operation code. It informs the assembler that generation and modification of an intended structure of code should begin at this point. The Reference Line provides the needed expressions for modification in its operand field.

4. A Paraform is a coined word for ''Parameter Reference Form''. A paraform is a device to inform the assembly system what expression in the operand of the reference line is to be substituted for the paraform when encountered. Use of the Paraform is restricted to the operand field of any symbolic line located within the bounds of a PROC.

   a. Constitution of a Paraform:

   A paraform consists of the operation code of the reference line to which it pertains, followed directly by a set of parenthesis. The parenthesis contain a double coordinate reference system expressed as n,e. n refers to a particular field in the reference line, and e refers to a particular subfield within that field.

   EXAMPLE 4: Paraform: Compare (3,1)

   Compare is the operation code found in the reference line of E.g. 2 The coordinate system is defining field 3, subfield 1 of the operand. This is Y. The resulting generation from line 2 Ex. 3 would be:

   ```
           C  8 , Y.
   ```

   b. Constitution of Operand of Reference Line

   (1) The operand consists of one or more fields separated by one or more blanks not preceded by an asterisk or comma.

   EXAMPLE 5: Operand of a Reference Line

   ```
   3 fields:  A  Compare  X  TO  Y
   1 field:   A  Compare  X, TO,*Y
   ```

   (2) A field may consist of one or more subfields separated by commas. Any expression may constitute a subfield. Any group of items joined by operators may constitute an expression. (See Ex. 5) Fields and subfields are counted sequentially from left to right.

C. SEQUENCE OF EXPRESSIONS IN A REFERENCE LINE

The order of expressions given in a reference line is optional. Generally, however, some sort of grouping or logical sequence should be employed. This aids the programmer, reduces errors, and allows for simplier coding methods. In Ex. 2 the operand has been set up to read like English, but the fields "TO" and "OR" are not essential.

D. FORMATS OF A PROCEDURE

1. Reference Line: Label Operation Operand

   a. Label: Any normal label is acceptable, it refers to the first line of code generated.

   b. Operation: The label of any Procedure is acceptable.

   c. Operand: Any number of fields or subfields, in any sequence desirable is acceptable.

2. PROC Line: Label PROC Operand

   a. Label: Any normal label not exceeding 8 characters is acceptable as identification of a PROC structure.

   b. PROC: This directive occupies the operation field and signals the assembler.

   c. Operand: Three operands are possible: blank, value, period. They are used to restrict the number of parameters that may be used from a reference line. Blank means indeterminate, value means maximum number of fields utilized, period means any variable up to the period encountered in the reference line's operand. It should be noted that column 72 contains an automatic period which can be interpreted to mean:

      (1) Blank in a PROC operand does not exist.

      (2) Omitting a coded period in the reference line is ineffectual.

3. Symbolic Line: Label Operation Operand

   a. Label: Any normal label may be employed, however, its definition will be restricted to the bounds of the PROC, unless it is an entry point*. Any label may be made available immediately outside the bounds of the PROC that contains* it by appending an asterisk to the label as a suffix.

      (* Multiple entry points are permissable and multilevels are permissable. These will be described in the text of the Procedure: Section II.)

   b. Operation: Any mnemonic, designated special character, label of a PROC, or directive is permissable.

   c. Operand: Any operand appropriate to the operation code is acceptable.

4. END Line: Label END Operand

   a. Label: None, no purpose.

   b. END is coded in operation field.

   c. Operand: None, no purpose.

E. ILLUSTRATIONS

```
Procedure:        LOAD  PROC
                        LA      8, LOAD (3,1)
                        END

Reference:        LOAD  Register with X
```

Note: The above procedure is being used to translate English into the requires UTMOST code. The first and second fields in the operand of the reference line are ignored.

```
Procedure:        TYPE  PROC    1                                 (1)
                          LA    1 , TYPE (1,1)                    (2)
                          OR    1 , (0,1)                         (3)
                          +     03600000                          (4)
                          +     037777                            (5)
                          J     $ - 4                             (6)
                          END                                     (7)
```

Reference: TYPE ((('START $\overset{+}{\underset{-}{0}}$)) – 1) (8)

Note: The above PROC illustrates how to achieve a one or two typeout under the BOSS II System Line two provides the location of the first word to be typed out. Line 3 superimposes index 1; line 4 is an illegal operation which causes interrupt. Line 5 is a flag checked by the executive to determine that a typeout is desired. Line 6 provides a loop if the typewriter is busy. The reference line contains a line item which will generate an alphabetic two word literal; and it will generate the first word address of that literal which is substituted in line 2 by reason of the Paraform. If a one word typeout is desired the ' 1' and one set of parenthesis is omitted; the carriage return symbol $\overset{+}{0}$ is a plus – zero multipunch.

THE PROCEDURE: SECTION II

A. BASIC PREMISES OF PROCEDURES

To more clearly understand the mechanics of procedures, it is necessary to demonstrate three principles.

1. Locality of Labels:

A label is a symbolic representation of some value. The extent, the region, the locale within which this label is known to represent this value is either limited or universal. A universal label is one whose value extends beyond the program. A local label is one whose value is restricted to the program, or to a procedure. While labels at a program level are available to procedures, labels at a procedure level are not normally available outside of the procedure. The nesting of procedures creates a hierarchy of levels or regions within which labels are or are not available. The innermost nest or the minor procedures can call on labels from the outermost or major procedures. The reverse is generally not true. The star, or asterisk may be appended as a suffix to a label. It is not a part of the label, it is to be considered only as a flag. Any label having a star suffix is available one region higher. A label at program level having a star becomes universal i.e. its value can be ascertained outside of the program. Such a label is called an external definition and is the counterpart of the external reference which is a label undefined at program level. A standard label in a procedure would become available to program level and a label in a nested procedure becomes available one procedure higher.

EXAMPLE: Locality of LABELS in Procedure

```
Major    PROC
Mediate  PROC
Minor    PROC
M1       EQU   10
M1A*     EQU   15
         END
M2       EQU   20
M2A*     EQU   25
         END
M3       EQU   30
M3A*     EQU   35
         END
         MAJOR
         MEDIATE
         MINOR
```

Note: Value M1 is available only to Minor PROC.

Value M1A* and M2 ia available to Minor PROC and Mediate PROC.

Value M2A* and M3 is available to Minor, Mediate, and Major PROCs.

Value M3A is available to the PROCs and the program.

The program calls Major, Mediate and Minor PROCS on 3 successive reference lines. Because of physical restriction the reference to Mediate and Minor would be errors: they are not available to the program. If the PROCS were physically outside of each other, these references would not be in error, but the locale of all labels would be inferred as above.

2. Redefinition of Labels

Normally, if a label has two definitions that conflict, it is flagged as a duplicate error. Labels within procedures, however, may be purposefully redefined, continuously or intermittently without incurring an error. The locality of labels can be temporal as well as physical. While the same label may have different values in different PROCS and not be confused, the same label may have different values in the same PROC because time is a sequence that imposes boundaries.

EXAMPLE: Temporal Regions

```
SPINE   PROC
P       EQU     O
SPIRAL  NAME
P*      EQU     P + SPINE (1, 1)
        +       P
        DO      P < 10 , GO SPIRAL
        END
        SPINE 2
        SPINE 3
```

Note: The above example displays Recursive and Recurrent references employing time boundaries. P is referenced and redefined recursively until the proper value is achieved. Spine is referenced recurrently and each time P is recursively redefined.

3. Forward References

When a label is referenced, and it has not yet been defined either because of physical location, or because it is dependent on other values not yet available (time) then this situation is termed a forward reference. When dealing with directives or references to directives, forward references, as a general rule, are prohibited.

EXAMPLE: FORWARD REFERENCES

```
        CONE    33
        RES     ((+1))
CONE    PROC
        DO   CONE = 2, COIL
N*      EQU  CONE (1, 1) **3
        END
COIL    PROC
M*      EQU  CONE (2, 1)//CONE(1, 2)
        END
        CONE   34  4
```

Note: (1) This procedure produces no code and would cause an error if assembled.

(2) CONE 33 is a forward reference and is an error. RES is an error because it seeks a value of a location from the literal table which is not yet formed. COIL, however, is a forward reference which is not an error because COIL is defined before CONE 34, 4 is encountered. UTMOST employs a sampling technique which lists all labels associated with PROC and NAME lines and star suffixes when it encounters a procedure. The availability of this list allows the forward reference just explained.

B. MODES OF PROCEDURES

Procedures can be developed in three different modes. Simple, Generative, Interpretative.

1. Simple modes occur when the object procedure developed is equivalent to the object procedure declared.

   E.g. Simple Mode Procedure

```
CTR     PROC
        LA    8, CTR (1,1)
        BA    8, CTR (1,2)
        SA    8, CTR (1,1)
        END
        CTR   Tally, 13
```

   Note: The above described Procedure declares and generates 3 lines of code which add a given value to a given counter.

2. Generative mode occurs when the object procedure developed is a multiple of the object procedure declared.

   E.g. Generative Mode Procedure

```
CTR     PROC
SUBCT   PROC
        LA      8, SUBCT (1,1)
        BA      8, SUBCT (1,2)
        SA      8, SUBCT (1,1)
        END
Q       DO      CTR , SUBCT CTR(Q,1), CTR(Q,2)
        END
        CTR     Tally, 13  Toll, 5 Total, 2
```

   Note: The call line supplies 3 fields for which 3 lines each of code will be generated.

3. Interpretative mode occurs when the object procedure declared will interpret the fields given and generate code based on the interpretation.

   E.g. Interpretative Mode Procedure

```
JPS     PROC
T       EQU JPS (1,1)
        JP   (T=8)+(T=4)+3*(T=2)+4*(T=1), JPS (1,2), JPS(1,3)
        END
        JPS 8, END
```

   Note: In UNIVAC III, the status of an arithmetic register may be ascertained by testing the indicator with it. The indicator, however, has a representative value that differs from the value of the register. The JPS (Jump Positive Sign) PROC allows the programmer to write the register value which will be translated into the proper indicator value. The scheme is displayed above and is based on the veracity of the T=8, T=4, T=2, T=1 statements.

C. POTENTIAL OF PROCEDURES

The potential of procedures may be described as those abilities and discretions that may be employed when using them.

1. Procedures may be nested; i.e., they may be included within each other. Nesting may be physical or it may be implied.

   a. Physical nesting means that the procedure is physically located in the bounds of another procedure.

   b. Inferred nesting means that though a procedure is not physically contained within another, it may be temporarily considered so by implication: i.e. its reference line is contained within another PROC.

   The purpose of nesting procedures is to restrict labels, if they interfere with labels from other procedures or the main program. Another purpose is to re-equate labels for homogeneity. Nesting allows simpler block building techniques but requires longer assembly time.

2. Decision Making: The Conditional DO statement allows symbolic lines to be created or negated. The GO may be also employed to include or skip lines of code.

3. Random Entrance: The name directive allows random entrance into a procedure. This is a method for qualifying a procedure.

4. PROCS Employ All Directives: Since procedures allow the presence of all directives their power is enhanced. Of special value are the NAME, GO, DO, and EQU directives.

5. PROCS are Reflexive: They may fall back on themselves for introspection.

6. PROCS are Restrictive: Because procedures do employ certain restrictive measures, they become all the more powerful, by avoiding possible ambiguous situations.

D. RESTRICTIONS IN PROCEDURES:

Restrictions help develop unique situations, they also hinder general methods. They are like two edged blades which must be employed deftly.

1. The procedure may restrict the maximum number of fields it will employ.

2. Labels are local to a procedure but may be starred to make them more universal by levels.

3. Nesting further restricts the locale of labels of inner procedures, but enlarges the locale of labels of outer procedures.

4. The redefinition of labels is a restrictive process since it destroys previous values. It may not always be intentional.

5. Forward Referencing is a restrictive process which allows the redefinition of labels.