<u>UNIT 3</u>

PROCEDURE DIVISION

PROCEDURE DIVISION

```
PROCEDURE DIVISION[USING <DATA-ITEM1>, <DATA-ITEM2>.
MAIN-PARA.
        DISPLAY  ‘ENTER  VALUE OF A:’.
        ACCEPT A.
        DISPLAY  ‘ENTER  VALUE OF B:’.
        ACCEPT A.
        MOVE A TO B.
        ADD A TO B.
        DISPLAY ‘A VALUE :’  A.
        DISPLAY ‘B VALUE :’  B.
        ---------------------------------
         --------------------------------




         STOP RUN.
```

Figure 3-1 PROCEDURE DIVISION

Notes :

Procedure Division can consists of

- Sections (Optional)
- Paragraphs(Optional)
- Statements.

While coding, we must follow the following Hierarchy:

SECTION-------→ PARAGRAPHS ------→ STATEMENTS

Or

PARAGRAPH-------→ STATEMENTS

Or

STATEMENTS

COBOL VERBS

All instructions are coded in Procedure division.

<u>BASIC COBOL VERBS</u>

- **MOVE**
- **ACCEPT**
- **DISPLAY**
- **PERFORM**
- **GO TO**
- **STOP RUN**
- **CALL**
- **COPY**
- **SORT**
- **MERGE**
- **FILE OPERATIONS**
- **CHARACTER HANDLING**
- **TABLE HANDLING**
- **CONDITIONS**
- **ARITHMETIC VERBS**

Figure 3-2 COBOL Verbs

Notes:

Arithmetic Verbs	: ADD, SUBTRACT, MULTIPLY, DIVIDE, COMPUTE
Conditions	: IF….ELSE, EVALUATE
File handling	: READ, WRITE, REWRITE, DELETE
Character handling	: INSPECT, STRING, UNSTRING
Table handling	: SET, SEARCH

Paragraphs

- **Paragraphs are building blocks of the PROCEDURE DIVISION**

```
PROCEDURE DIVISION.
MAIN-PARA.
          STATEMENT1.
          STATEMENT2.
          ---------------------
          ---------------------
          ----------------------
PARA-100.
         -----------------------
        -------------------------
```

Figure 3-3   Paragraphs

Notes:

A paragraph-name must begin in Area A and must be followed by a separator period.

A paragraph-name need not be unique because it can qualified by a SECTION name.

Paragraph-names need NOT contain any alphabetic character (i.e. can be all numeric).
A paragraph ends at:

- The next paragraph-name or section header
- The end of the PROCEDURE DIVISION
- The Scope terminator END-PARAGRAPH

Terminator Statements

- **EXIT PROGRAM.**
  The EXIT PROGRAM statement specifies the end of a called program and returns control to the calling program

- **STOP RUN.**
  The STOP RUN statements halts the execution of the object program, and returns control to the system

- **GOBACK.**
  The GOBACK statement functions like the EXIT PROGRAM statement
  When it is coded as part of a called program and like the STOP RUN when coded in a main program

Figure 3-4 Terminator Statements

Notes:

If these statements are not the last statements in a sequence, statements following them will not be executed.

Scope Terminators

Explicit scope terminators mark the end of certain PROCEDURE DIVISION statements.

Explicit scope terminators are COBOL Reserved Words.

END-ADD
END-MULTIPLY
END-PERFORM
END-READ
END-EVALUATE
END-IF
END-SEARCH
END-START
END-STRING
END-DIVIDE
END-REWRITE
END-CALL
END-COMPUTE
END-DELETE
END-UNSTRING
END-WRITE

An explicit Scope Terminator is paired with the unpaired occurrence of the verb.

An implicit Scope Terminator is a separator period.

Figure 3-5 Scope Terminators

Notes:

Example:

```
  PERFORM   PARA-1 UNTIL A > 10
        STATEMENT1
        STATEMENT2
        -------------------
         -------------------
        --------------------
END-PERFORM.
```

Period(.) should not encounter in between PERFORM and END-PERFORM. Since it indicates end of the PERFORM statement, then compiler error will raise.

DISPLAY Verb

The function of the DISPLAY statement is to display low-volume results on the operator's console or some other hardware device.

Syntax :

```
>>____DISPLAY_____  __identifier-1___  __ | ________________________________________>
                                 | _ literal-1______|
```

e.g:

```
PROCEDURE DIVISION.
DISP-PARA.
          DISPLAY  SRCH-ARG  'NOT IN TABLE.'.
          ----------------------------------
          -------------------------------
          DISPLAY 'HELLO HOW ARE YOU'.
```

Figure 3-6 DISPLAY Statement

Notes:

The DISPLAY statement transfers the contents of each operand to the output device. The contents are displayed on the output device in the order, left to right, in which the operands are listed.

WITH NO ADVANCING When specified, the positioning of the output device will not be changed in any way following the display of the last operand.

ACCEPT Verb

- **Format 1 transfers data from an input/output device into identifier-1.**
- **When the FROM phrase is omitted, the system input device is assumed.**
- **Format 1 is useful for exceptional situations in a program when operator intervention (to supply a given message, code, or exception indicator) is required.**

Format 1 :

```
>>__ACCEPT______identifier-1___  ______________________________   ______________><
                                 |_  FROM__  _mnemonic-name-1___  _|
                                             |_ environment-name _  |
```

```
77    SEARCH-VALUE       PIC     X(10).
….
      ACCEPT SEARCH-VALUE  FROM SYSIN.
```

Figure 3-7   ACCEPT Statement – Format 1

Notes:
The ACCEPT statement transfers data into the specified identifier. There is no editing or error checking of the incoming data.

If the source of the ACCEPT statement is a file and identifier-1 is filled without using the full record delimited by the record terminator, the remainder of the input record is used in the next ACCEPT statement for the file. The record delimiter characters are removed from the input data before the input records are moved into the ACCEPT receiving area.

If the source of the ACCEPT statement is a terminal, the data entered at the terminal, followed by the enter key, is treated as the input data. If the input data is shorter than identifier-1, the area is padded with spaces.

MOVE Verb

MOVE verb is used to copy the contents of an identifier into another identifier.

```
MOVE  <identifier-1>
             Or              TO  <identifier-2>[<identifier-3>,……….].
         <literal-1>
```

E.g.:

```
MOVE A   TO   B,C,D
MOVE dataname-1 to dataname-2
MOVE  345 to num-1
MOVE  ‘345’ TO K
MOVE ‘XYZ’ TO data-name-1
```

If the length of the receiving field is less than the length of sending field then truncation occurs.

Figure 3-8 MOVE Statement

Notes:

The MOVE statement transfers data from one area of storage to one or more other areas.

An index data item cannot be specified in a MOVE statement.

If the sending field(identifier-1) is reference-modified, subscripted, or is an alphanumeric or alphabetic function-identifier, the reference-modifier, subscript, or function is evaluated only once, immediately before data is moved to the first of the receiving operands.

Elementary & Group Moves

The receiving or sending field of a MOVE statement can be either an elementary item or a group item.When both the fields are elementary items , the data movement is known as an **elementary move**. When atleast one of the fields is a group item, it is called **group move**.

```
01  MSG-FLD          PIC  X(10).
01  DATA-FLD         PIC  X(10).
01  OLD-ADDR.
  05   NO           PIC  X(5).
  05   NAME         PIC  X(15).
    ------------------------------
     ------------------------------
01  NEW-ADDR.
  05   N-NO           PIC  X(5).
  05   N-NAME         PIC  X(15).
    ------------------------------
     ------------------------------

MOVE ‘OUT OF SEQUENCE’ TO MSG-FIELD
MOVE SPACES TO OLD-ADDR, NEW-ADDR

MOVE DATA-FLD TO MSG-FIELD.
MOVE NEW-ADDR TO OLD-ADDR.
```

Figure   3-9 Elementary &Group Moves

Notes:

Elementary move

- Both sending and receiving data items are elementary items
- Data conversion may take place, as well as editing or de-editing
- On alphabetic moves, all necessary space-fill or truncation will occur

Group Move

- Both sending and receiving data items are group items
- No data conversion takes place

CORRESPONDING Phrase

```
01 STRUCT-1.
 03   FIELD-A         PIC     9(9)     VALUE  123456789.
         03   FIELD-B          PIC      X(5)      VALUE  “abcde”.
         03   FIELD-C          PIC      9(4)V99  VALUE 1234.56.
         03   FIELD-D          PIC      9(4)V99   VALUE  123456789.
      01 STRUCT-2.
         10   FIELD-C          PIC      Z(4).99.
         10   FILLER           PIC      XXX.
         10   FIELD-B          PIC      X(5).
         10   FILLER           PIC      XXX.
         10   FIELD-A          PIC      Z(9)
         10   FILLER           PIC     XXX.

         MOVE CORRESPONDING STRUCT-1  TO STRUCT-2
```

Statement moves 3 fields but gives warning.

Figure 3-10 CORRESPONDING Phrase

Given the data definitions in the visual, the MOVE CORRESPONDING statement in the visual moves three fields (FIELD-A, FIELD-B and FIELD-C) but gives a warning message similar to the one below.

ILLUSTRATES MOVE CORRESPONDING

```
DATA DIVISION

WORKING – STORAGE SECTION.
01      DATA-1
        05      E-ID        PIC 9(5)          VALUE 2345.
        05      E-NAME      PIC X (25)        VALUE ALL “N”.
        05      E-DEPT      PIC X (20)        VALUE ALL “D”
        05      E-BASIC     PIC 9(4) V99      VALUE 1234.67.
01      DATA-2.
        05      FILLER              PIC X(5)
        05      E-ID                PIC 9(5)
        05      FILLER              PIC X(5)
        05      E-NAME              PIC X (25).
        05      FILLER              PIC X(5).
        05      E-DEPT              PIC X(20)
        05      FILLER              PIC X(5)
        05      E-BASIC             PIC 9(4). 99

PROCEDURE DIVISION.

PARA 1.

        MOVE E-ID OF DATA-1 TO E-ID OF DATA-2
        MOVE E-NAME OF DATA-1 TO E-NAME OF DATA-2.
        MOVE E-DEPT OF DATA-1 TO E-BASIC OF DATA-2.
        DISPLAY DATA-1
        DISPLAY DATA-2
        MOVE SPACES TO DATA-2.
        MOVE CORRESPONDING DATA-1 TO DATA-2.
        DISPLAY  DATA-1
        DISPLAY  DATA-2.
        STOP RUN.
```

Reference Modification

- **Reference Modification defines a data item by specifying its leftmost character and optionally, a length**

  MOVE data-name1(begin : [length]) TO data-name2

- **If ‘length’ is omitted, the data item continues to rightmost character of data-name1 (the colon is required).**

- **The data name must have usage DISPLAY. It may be qualified or subscripted. When qualified or subscripted, the reference modification is specified last.**

Figure 3-11 Reference Modification

Notes:

Eg:

```
WORKING-STORAGE  SECTION.
 01   CAT-TYPE    PIC X(15) VALUE 'CALICO'.
 01   DOG-TYPE    PIC X(15) VALUE 'SCHNAUZER'.
 01   CAT-ABBREV  PIC X(5).
 01   DOG-END     PIC X(10).
 PROCEDURE DIVISION.
*Reference Modification Example Number 1: (From position 1:For 5 positions.)

                 MOVE CAT-TYPE(1:5) TO CAT-ABBREV.
*This will move "CALIC" to CAT-ABBREV. (The letters from position 1 of CAT-TYPE for
5 positions.)
           DISPLAY CAT-ABBREV.

*Reference Modification Example Number 2: (From position 2:For 4 Bytes.)
```

```
            MOVE CAT-TYPE(2:4) TO CAT-ABBREV.
```

*This will move "ALIC" to CAT-ABBREV2. (The letters from position 2 of CAT-TYPE for 4 positions.)

```
            DISPLAY CAT-ABBREV.
```

*Reference Modification Example Number 3: (From position number 5 to the end of the field.)

```
            MOVE DOG-TYPE(5:) TO DOG-END.
```

*This will move "AUZER" to DOG-END. (The letters from position 5 of DOG-TYPE to the end of DOG-TYPE.)

```
          DISPLAY DOG-END.
```

ADD Verb

- **All identifiers (or literals) preceding the word TO are added together, and then this sum is added to, and replaces, each identifier-2. The action is repeated in order left-to-right for each identifier-2**
- **Identifiers must be elementary numeric items**

Format 1 :

```
>>___ADD_______ identifier-1_  _|__ To _____identifier-2__ ________  ____________  __|_________>
                |_literal-1___|                                      |_ROUNDED  _|

>___  ___________________________________________________  ___________________________>
     |_  _____  ___SIZE ERROR  imperative-statement-1_______|

>___ ____________________________________________________  ___________________________>
     |_ NOT___  ______  ___SIZE ERROR__imperative –statement_2_|

>____  __________  ____________________________________________________________________>
     |_ END-ADD_|
```

Figure 3-12 ADD Statement – Format 1

In Format 1, all identifiers or literals preceding the key word TO are added together, and this sum is stored in a temporary data item. This| temporary data item is then added to each successive occurrence of identifier-2, in the left-to-right order in which identifier-2 is specified.
Identifier must name an elementary numeric item.

Literal must be a numeric.

The ADD statement sums two or more numeric operands and stores the result.
Example :

ADD A TO B.
ADD 112 TO B.
ADD A TO B ON SIZE ERROR GO TO ERR-PARA.

ADD Verb(Continue….)

- **The operands preceding the GIVING are added together and the sum replaces the value of each identifier-3**
- **Identifiers must be elementary numeric items, except when following GIVING then they may also be numeric –edited.**

Format 2 :

```
>>___ADD_______ identifier-1_  _|__   ___   _  ____ _ __  _ identifier-2__ __________________________>
                |_literal-1___|             |_TO_|        |_literal-2______|

>___  GIVING ___________identifier-3__ ______________  _| ___________________________________ >
                                       |_ ROUNDED__|

>___ _____________________________________________________  ____________________________  >
    |_        _________    ___SIZE ERROR__imperative –statement_1_|

>___ _____________________________________________________    ___________________________ >
     |_  NOT___  ______     _SIZE ERROR__imperative –statement_2_|
                 |_  ON_|
>____  __________  _____________________________________________________________________ >
     |_ END-ADD_|
```

Figure 3-13 ADD Statement – Format 2

In Format 2, the values of the operands preceding the word GIVING are added together, and the sum is stored as the new value of each data item referenced by identifier-3.

Identifier must name an elementary numeric item, except when following the word GIVING. Each identifier following the word GIVING must name an elementary numeric or numeric-edited item
Literal must be a numeric.

Example :

ADD A TO B GIVING C

ADD CORRESPONDING Statement

- **Elementary data items within identifer-1 are added to, and stored in the corresponding elementary data items with identifer-2.**
- **ADD CORRESPONDING identifiers must be group items**

Format :

```
>>___ADD_______ CORRESPONDING_  ___identifier-1___   TO___ identifier-2____________________>
                |_CORR__________|

 >___  ______________   __ _______________________________________________  ________________ >
      |_ ROUNDED__|    | _ ____    ___SIZE ERROR____ imperative-statement-1_|
                           |_ ON_ |

 >___ ________________________________________________________   ____________________________ >
     |_NOT___  ______     __SIZE ERROR__imperative –statement_1_|
                |_ON___|
>___ ________________________________________________________   ____________________________ >
     |_  NOT___  ______     _SIZE ERROR__imperative –statement_2_|
                  |_  ON_|
 >____  _________  __________________________________________________________________________ >
      |_ END-ADD_|
```

Figure 3-14 ADD CORRESPONDING Statement

In Format 3, elementary data items within identifier-1 are added to and stored in the corresponding elementary items within identifier-2.

Identifier must name a group item.

Literal must be a numeric.

Notes:

ON SIZE ERROR Phrase

- **If the value of an arithmetic evaluation exceeds the largest value that can be contained in a result, then a size error condition exists.**
- **The SIZE ERROR condition applies to final results, not intermediate calculations**
- **If ON SIZE ERROR phrase is not specified, then truncation of the results will occur.**
- **If ON SIZE ERROR phrase is specified, the imperative statement (in ON SIZE ERROR) is taken, following which control is transferred to the end of the arithmetic statement.**
- **For ADD CORRESPONDING or SUBTRACT CORRESPONDING, the ON SIZE ERROR imperative is not taken until all individual additions or subtractions have been completed**.

Figure 3-15 ON SIZE ERROR Phrase

A size error condition can occur in three different ways:

- ° When the absolute value of the result of an arithmetic evaluation, after decimal point alignment, exceeds the largest value that can be contained in the result field
- ° When division by zero occurs
- ° In an exponential expression, as indicated in the following table:

Size error	Action taken when a SIZE ERROR clause is present	Action taken when a SIZE ERROR clause is not present
Zero raised to zero power	The SIZE ERROR imperative is executed.	The value returned is 1, and a message is issued.
Zero raised to a negative number	The SIZE ERROR imperative is executed.	Program is terminated abnormally.
A negative number raised to a fractional power	The SIZE ERROR imperative is executed	The absolute value of the base is used, and a message is issued

The size error condition applies only to final results, not to any intermediate results.

NUMERIC Data

Types of numeric items are:

- **Binary**
- **Packed decimal. (internal decimal)**
- **Floating point representation.**
- **The PICTURE character-string can contain only the symbols 9, P, S, and V**
- **The number of digit positions must range from 1 through 18, inclusive**
- **If unsigned, the contents of the item in standard data format must contain a combination of the Arabic numerals 0-9. If signed, it may also contain a +, -, or other representation of the operation sign**

Figure 3-16 NUMERIC Data

Notes:

A VALUE clause can specify a figurative constant ZERO.

SUBTRACT Verb

Format 1 :

```
 >>___SUBTRACT_______ identifier-1_  _|__ FROM________________________________________>
                     |_literal-1___|

> ______identifier-2__  _______________  _|__________________________________________________>
                        | _ ROUNDED ____|

>___  ____________________________________________________  ___________________________>
    |_  _____  ___SIZE ERROR  imperative-statement-1_______|
        |_ON _|
>___ ______________________________________________________  ___________________________>
    |_ NOT___  ______  ___SIZE ERROR__imperative –statement_2_|


>____  __________  __________________________________________________________________>
     |_ END-SUBTRACT_|
```

All identifiers or literals preceding the key word FROM are added together and this sum is subtracted from and stored immediately in identifier-2. This process is repeated for each successive occurrence of identifier-2, in the left-to-right order in which identifier-2 is specified.

Figure 3-17 SUBTRACT Statement – Format 1

Notes:

SUBTRACT Verb(Continue………….)

Format 2 :

```
 >>___SUBTRACT_______ identifier-1_  _|__  FROM ___  _ identifier-2__ __________________________>
                     |_literal-1___|                   |_literal-2______|

>___  GIVING ___________identifier-3__ ______________  _| ____________________________________ >
                                      |_ ROUNDED___|

>___ ______________________________________________________   ____________________________ >
      |_        _______    ___SIZE ERROR__imperative –statement_1_|
             |_ ON     _|

>___ ______________________________________________________    ____________________________ >
      |_  NOT___  ______     _SIZE ERROR__imperative –statement_2_|
                  |_  ON_|
 >____  __________  _____________________________________________________________________ >
       |_ END-SUBTRACT_|
```

All identifier or literals preceding the key word FROM are added together and this sum is subtracted from identifier-2 or literals-2. The result of the subtraction is stored as the new value of each data item referenced by identifier-3.

Figure 3-18 SUBTRACT Statement – Format 2

Notes:

Example:

1. SUBTRACT A FROM B.
   The value of A subtttracted from the value of B and then the resultant value will be stored in B.

2. SUBTRACT 9 FROM C.
3. SUBTRACT C FROM 9. Is not valid because 9 is a Literal.

SUBTRACT CORRESPONDING Statement

Format :

```
>>___SUBTRACT_______ CORRESPONDING_  ___identifier-1___   FROM_______________________>
                     |_CORR__________|

 >___  identfier-2____   __ ___________   _____________________________________________________ >
                        |_ ROUNDED__|

 >___ _____________________________________________________   ____________________________  >
      |____  ______   __SIZE ERROR__imperative –statement_1_|
          |_ON___|
>___ _____________________________________________________    ____________________________  >
     |_  NOT___  ______     _SIZE ERROR__imperative –statement_2_|
                |_  ON__|
 >____   __________   ________________________________________________________________________ >
       |_ END-SUBTRACT_|
```

Elementary data items within identifier-1 are subtracted from, and the results are stored in, the corresponding elementary data items within identifier-2.

Figure 3-19   SUBTRACT CORRESPONDING Statement

Notes:

MULTIPLY Verb

Format 1 :

```
>>___MULTIPLY_______ identifier-1___  ___BY____identifier-2___  __________________| __________>
                      |_ literal-1________|

 >___ ____________________________________________________   ____________________________  >
     |____  ______    __SIZE ERROR__imperative –statement_1_|
         |_ON___|
>___ _____________________________________________________   ____________________________  >
     |_  NOT___  ______    _SIZE ERROR__imperative –statement_2_|
               |_  ON__|
 >____   __________  ____________________________________________________________________ ><
      |_ END-MULTIPLY_|
```

In Format 1, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2; the product is then placed in identifier-2. For each successive occurrence of identifier-2, the multiplication takes place in the left-to-right order in which identifier-2 is specified.

Figure 3-20   MULTIPLY Statement – Format 1

Notes:

The MULTIPLY statement multiplies numeric items and sets the values of data items equal to the results.

MULTIPLY Verb(Ccontinue…..)

Format 2 :

```
>>___MULTIPLY_______ identifier-1_  _|__  BY_______  _ identifier-2__ __________________________>
                     |_literal-1___|                  |_literal-2______|

 >___  GIVING ___________identifier-3__ ______________  _| ___________________________________ >
                                        |__ ROUNDED___|

 >___ _____________________________________________________   ____________________________ >
      |_       _______    ___SIZE ERROR__imperative –statement_1_|
               |_ ON     _|

>___ _____________________________________________________    ____________________________>
      |_  NOT___  ______     _SIZE ERROR__imperative –statement_2_|
                  |__  ON__|
 >____  _________   ___________________________________________________________________ >
        |_ END-MULTIPLY_|
```

In Format 2, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2 or literal-2. The product is then stored in the data item(s) referenced by identifier-3.

Figure 3-21 MULTIPLY Statement – Format 2

Notes:

DIVIDE Verb

Format 1 :

```
 >>___DIVIDE_____ _____ identifier-1_  _|__ INTO__________identifier-2____  _________     __ |____>
                      |_literal-1___|                                      |_ROUNDED_|

>___  ____________________________________________________   ____________________________>
     |_  ____  __SIZE ERROR  imperative-statement-1____________|
         |_ON _|
>___ _____________________________________________________   ____________________________>
    |_ NOT___  ______  ___SIZE ERROR__imperative –statement_2_|
                  |_ON __|
>____   __________   _____________________________________________________________________>
     |_ END-DIVIDE_|
```

In Format 1, the value of identifier-1 or literal is divided into the value of identifier-2, and the quotient is then stored in identifier-2. For each successive occurrence of identifier-2, the division takes place in the left-to-right order in which identifier-2 is specified.

Figure 3-22 DIVIDE Statement-Format 1

Notes:

The DIVIDE statement divides one numeric data item into or by other(s) and sets the values of data items equal to the quotient and remainder

DIVIDE Verb(Continue……)

Format 2 :

```
>>___DIVIDE_______ identifier-1_  _|__  INTO_______  _ identifier-2__ ___________________________>
                   |_literal-1___|                     |_literal-2______|

 >___  GIVING ___________identifier-3__ ______________  _| ___________________________________ >
                                        |_ ROUNDED___|

 >___ _____________________________________________________   ____________________________ >
     |_       _______    ___SIZE ERROR__imperative –statement_1_|
            |_ ON     _|

>___ _____________________________________________________    ____________________________>
    |_  NOT___  ______     _SIZE ERROR__imperative –statement_2_|
                    |_  ON_|
 >____  __________  ___________________________________________________________________ >
      |_ END-DIVIDE_|
```

In Format 2, the value of identifier-1 or literal-1 is divided into or by the value of identifier-2 or literal-2. The value of the result is stored in each data item referenced by identifier-3.

Figure 3-23 DIVIDE Statement – Format 2

Notes:

COMPUTE Verb

Format :

```
>>___COMPUTE_______ identifier-1_  ____________  _|____  _ =______   __________________________>
                                  |_ ROUNDED _|          |_ EQUAL_|

 >___  arithmetic –expression______________________________________________________________ >

 >___ _____________________________________________________   _____________________________ >
     |_        _______    ___SIZE ERROR__imperative –statement_1_|
              |_ ON    _|

>___ _____________________________________________________    _____________________________ >
    |_  NOT___  ______     _SIZE ERROR__imperative –statement_2_|
                |_  ON_|
 >____  __________   ______________________________________________________________________ >
      |_ END-COMPUTE_|
```

The arithmetic expression is calculated and replaces the value for each identifier-1 item. Valid operators allowed in the expression are:

+ addition	**- subtraction**
*** multiplication**	**/ division**
**** exponentiation**	

Figure 3-24 COMPUTE Statement

Notes:

The COMPUTE statement assigns the value of an arithmetic expression to one or more data items.

With the COMPUTE statement, arithmetic operations can be combined without the restrictions on receiving data items imposed by the rules for the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements.

Identifier-1

Must name elementary numeric item(s) or elementary numeric-edited item(s).

Can name an elementary floating-point data item.

The word EQUAL can be used in place of =.

An arithmetic expression ca consist of any of the following:

1. An identifier described as a numeric elementary item
2. A numeric literal
3. The figurative constant ZERO
4. Identifiers are literals, as defined in terms 1,2, and 3, separated by arithmetic operators
5. Two arithmetic expressions, as defined in items 1,2,3, and/or 4, separated by an arithmetic operator
6. An arithmetic expression, as defined in items 1,2,3,4 and/or 5, enclosed in parentheses.

When the COMPUTE statement is executed, the value of the arithmetic expression is calculated, and this value is stored as the new value of each data item referenced by identifier-1.

PERFORM Statement

PERFORM Paragraph-name/Section-header

Transfer the control to the specified paragraph or section and expects the control back after executing the paragraph.

PERFORM Para-name-1 [THROUGH (or) THRU Para-name-n]

Figure 3-25 PERFORM

Notes:

PERFORM types

- PERFORM para-name
- PERFORM para-name N TIMES
- PERFORM para-name VARYING K FROM M BY N
  UNTIL CONDITION K>20
- PERFORM para-name VARYING K FROM M BY N UNTIL CONDITION K>20 AFTER VARYING….

PERFORM THROUGH

```
  PROCEDURE DIVISION.
  100-MAIN-PARA.
              PERFORM 200-PARA  THRU 500-PARA.
              STOP RUN.

  200-PARA.
* Statements.

   400-PARA.
* Statements

   500-PARA.
* Statements

    300-PARA.
*   Statement   - Not executed
```

All the paragraphs between 200-PARA and 500-PARA are executed.

Figure 3-26 PERFORM THROUGH

Notes:

PERFORM…………N times

PERFORM PARA-NAME-1[THROUGH (or) THRU PARA-NAME-N]
N TIMES.

EX:

```
PERFORM  PARA-1000 15 TIMES.

PERFORM  PARA-1000 THRU PARA-4000   15  TIMES.

PARA-1000.
          ADD A TO B.
          ------------------------
          -------------------------
PARA-2000.
         SUBTRACT A FROM B.
          ------------------------------
         --------------------------------
PARA-4000.
         MULTIPLY  A BY B.
         ----------------------------
```

Figurre 3-27 PERFORM N times

Notes:

PERFORM…………VARYING

```
PERFORM  PARA-NAME-1 [THRU (or)  THROUGH  PARA-NAME-N]
         VARYING    { identifier- 1   }               {identifier-2   }
                    {Index-name-1}   FROM        {index-name-2}
                                                  { Literal-1      }

         BY            {identifier-3 }       UNTIL      Condition
                       {Literal-2     }
```

EX:

1. PERFORM PARA-2000 THRU PARA-5000 VARYING A FROM M BY N UNTIL A > Y

```
2. PERFORM para-1 Varying K FROM 10 BY 5
                  UNTIL K>100
```

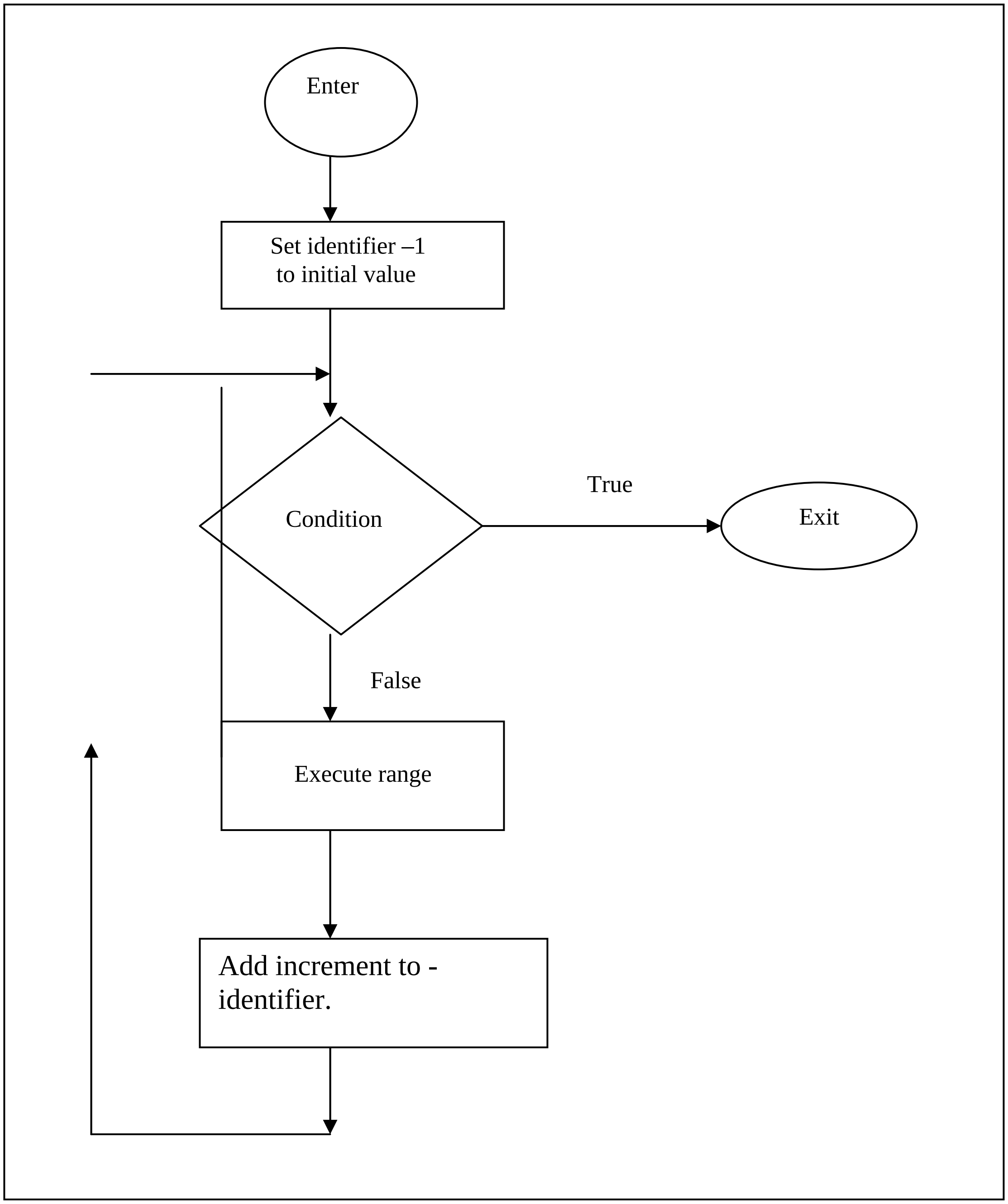
Figure 3-28 PERFORM……..VARYING

Notes:

Example 2 says :

Sets the value of K to 10 initially
Execute para-1
Check the condition K>100
If condition is true, transfer the control to next line
If condition is false, increment K by 5
Execute para-1 again
Check the condition K > 100
Repeat steps from 2 through 7 until Condition K > 100 becomes true

Flow Chart for PERFORM ….. VARYING



PERFORM with the VARYING-AFTER Option

```
PERFORM  PARA-NAME-1 [THRU (or)  THROUGH  PARA-NAME-N]
              VARYING    { identifier- 1    }                 {identifier-2    }
                         {Index-name-1}   FROM          {index-name-2}
                                                        { Literal-1      }

              BY             {identifier-3 }       UNTIL      Condition-1
                                {Literal-2    }


          [   AFTER          { identifier- 4    }                 {identifier-5    }
                         {Index-name-3}   FROM          {index-name-4}
                                                        { Literal-3      }

                 BY             {identifier-6 }       UNTIL      Condition-2
                                   {Literal-4    }

           [  AFTER          { identifier- 7    }                 {identifier-8    }
                         {Index-name-5}   FROM          {index-name-6}
                                                        { Literal-5      }

                 BY             {identifier-9 }       UNTIL      Condition-3  ]
                                   {Literal-6    }                           ]
```

This form is used when a nested repetition of the range is required while varying more than one identifier.
For example

```
   PERFORM  RANGE-TO-BE-EXECUTED
                 VARYING    I    FROM   1   BY  1  UNTIL  I > 50
                  AFTER      J   FROM   1    BY  1  UNTIL  J  > 10.
```

The range RANGE-TO-BE-EXECUTED will be performed 500 times,.

In-Line PERFORM

The in-line PERFORM will be coded using END-PERFORM.

Named Paragraph

```
     PERFORM MOVEIT
            VARYING X FROM 1 BY 1 UNTIL X = 5.
  . . .

    MOVEIT.
          MOVE DATA-FLD (X) TO PRINT (X).
```

In-line PERFORM

```
     PERFORM VARYING X FROM 1  BY 1 UNTIL X = 5.
            MOVE DATA-FLD (X) TO PRINT (X).
     END-PERFORM.

…
```

Figure 3-29 In-line PERFORM

Notes:

An In-line PERFORM requires the END-PERFORM terminator. Conversely the END-PERFORM phrase must not be specified when the statement is “PERFORM procedure name…”.

IN-LINE PERFORM Considerations

- DO not use for procedures executed from several places/
- Use for procedures referenced only once.
- Consider not using if readability is affected , such as multiple-page PERFORM,
- No periods may appear within the in-line PERFORM.
- Delimited by END-PERFORM.
- END-PERFORM cannot be used at end of an out-of-line PERFORM.
- The OPTIMIZE compile option may move the PERFORM in-line in the object code at the compile time.

IF .. ELSE Statement

The IF statement evaluates a condition and provides for alternative actions in the object program, depending on the evaluation.

Format :

```
>>_______IF_____Condition-1____   ___________   _____   ___statement-1___|__  ____________________>
                                |_THEN_____|          |_NEXT SENTENCE _|
>___  ____________________________  ____  ________________   ______________________________>
    |            <____________          |     |                 (1) |
    | _ ELSE__  ___statement-2_|_____  |     |___END-IF________|
```

Note :
(1)   END-IF can be specified with NEXT SENTENCE as an IBM extension.

Figure 3-31 IF -ELSE Statement

Notes:

The IF statement evaluates a condition and provides for different sets of statements to execute, depending on the evaluation of the IF.

Condition can be any simple or complex condition.

Statement-1, statement-2 Can be any one of the following:

- An imperative statement
- An conditional statement
- An imperative statement followed by a conditional statement

NEXT SENTENCE If the NEXT SENTENCE phrase is specified, and then the END-IF phrase must not be specified. NEXT SENTENCE passes control to the statement after the closest following period. However, if the NEXT SENTENCE phrase is executed, control will not pass to the statement after the closest following period.

Compound Conditionals

- **Conditional expressions can be "compound" using the AND and OR logical operators**
- **Conditional conditions can also use parentheses to group conditions.**

```
IF        ITEM-1      =     DOMESTIC-ITEM-NO
   AND   ITEM-2      =      OVERSEAS-ITEM-NO
    OR
          ITEM-1      =     OVERSEAS-ITEM-NO
     AND  ITEM-2      =     DOMESTIC-ITEM-NO
      SET MIXED-SHIPMENT-FLAG TO TRUE
END-IF


…………….

SEARCH TABLEPAIR VARYING NDX
 WHEN ITEM-1(NDX)   =  FROM-CITY AND ITEM-2(NDX)   =  TO-CITY
        MOVE  ………
   WHEN ITEM-2(NDX)  =  FROM-CITY AND ITEM-1(NDX)  =  TO-CITY
                        MOVE  ……..
        END-SEARCH
```

Figure 3-32 Compound Conditionals

Notes:

Relational Expressions

Relational tests (comparisons) can be express as:

IS LESS THAN	**IS <**
IS NOT LESS THAN	**IS NOT <**
GREATER THAN	**IS >**
IS NOT GREATER THAN	**IS NOT >**
IS EQUAL TO	**IS =**
IS NOT EQUAL TO	**IS NOT =**
IS GREATER THAN OR EQUAL TO	**IS >=**
IS LESS THAN OR EQUAL TO	**IS <=**

Figure 3-30 Relational Expressions

Notes:

CONTINUE & NEXT SENTENCE Statement

Example 1 - **NEXT SENTENCE**

```
IF  A = B
          IF  C = D
                    NEXT SENTENCE
          ELSE
                    MOVE MESSAGE-1  TO  RPT-MESSAGE-1
          END-IF
          ADD C TO TOTAL
          DISPLAY TOTAL
          IF E = F
                MOVE MESSAGE-4  TO  RPT-MESSAGE-2
          END-IF
 END-IF.
```

Example 2 - **CONTINUE**

```
IF  A = B
   IF  C = D
        CONTINUE
   ELSE
        MOVE  MESSAGE-1  TO  RPT-MESSAGE-1
   END-IF
   ADD C TO TOTAL
   DISPLAY  TOTAL
   IF E = F
         MOVE MESSAGE-4  TO RPT-MESSAGE-2
   END-IF
END-IF.
```

Figure 3-34 CONTINUE Statement

Notes:

EVALUATE Statement

- **EVALUATE is a great way to implement the "case" programming construct**

```
EVALUATE dataname
            WHEN  value-1 …….
            WHEN  value-2 {THROUGH | THRU}  value-3 ….
            WHEN NOT value-4
            ……
            WHEN  OTHER
END-EVALUATE
```

- **Basic EVALUATE Example:**

```
EVALUATE  dataname
            WHEN  'A'                    Perform  add-trans
            WHEN  'D'                    Perform  delete-trans
            WHEN  'U'
            WHEN  'W'                    Perform update-trans
            WHEN  OTHER                Perform bad-trans
END-EVALUATE
```

- **The scope of a WHEN clause is all statements UNTIL the next WHEN clause, the END-EVALUATE, or a period**

Figure 3-35 EVALUATE Statement

Notes:

The EVALUATE statement provides a shorthand notation for a series of nested IF statements. It can evaluate multiple conditions. That is, the IF Statements can be made up of compound conditions.

Examples:

Working-Storage for all Examples:

```
01  PLANET.
    05  PLANET-NUMBER PIC 9.
    05  PLANET-NAME   PIC X(7).
```

Evaluate Example Number 1: (Evaluate a PIC 9 field)

```
EVALUATE PLANET-NUMBER
   WHEN 1    MOVE "Mercury"    TO  PLANET-NAME
   WHEN 2    MOVE "Venus  "    TO  PLANET-NAME
   WHEN 3    MOVE "Earth  "    TO  PLANET-NAME
   WHEN 4    MOVE "Mars   "    TO  PLANET-NAME
   WHEN 5    MOVE "Jupiter"    TO  PLANET-NAME
   WHEN 6    MOVE "Saturn "    TO  PLANET-NAME
   WHEN 7    MOVE "Uranus "    TO  PLANET-NAME
   WHEN 8    MOVE "Neptune"    TO  PLANET-NAME
   WHEN 9    MOVE "Pluto  "    TO  PLANET-NAME
   WHEN OTHER MOVE "       "   TO  PLANET-NAME
END-EVALUATE.
```

Evaluate Example Number 2: (Evaluate a PIC X field)

```
EVALUATE PLANET-NAME
      WHEN   "Mercury"   MOVE 1    TO   PLANET-NUMBER
      WHEN   "Venus  "   MOVE 2    TO   PLANET-NUMBER
      WHEN   "Earth  "   MOVE 3    TO   PLANET-NUMBER
      WHEN   "Mars   "   MOVE 4    TO   PLANET-NUMBER
      WHEN   "Jupiter"   MOVE 5    TO   PLANET-NUMBER
      WHEN   "Saturn "   MOVE 6    TO   PLANET-NUMBER
      WHEN   "Uranus "   MOVE 7    TO   PLANET-NUMBER
      WHEN   "Neptune"   MOVE 8    TO   PLANET-NUMBER
      WHEN   "Pluto  "   MOVE 9    TO   PLANET-NUMBER
      WHEN   OTHER       MOVE 0    TO   PLANET-NUMBER
END-EVALUATE.
```

Evaluate Example Number 3:

Let each of MONTH and NO-OF-Days be two-digited numeric integer fields. The values 1,2,3, etc. for MONTH denote respectively, January, February, March etc. depending on the value of MONTH , we wish to ove 30,31 or 28 to NO-OF-DAYS. For example , if the value of MONTH is 1, we shall move 31; if it is 2, we shall move 28 and so on. The EVALUATE statement for the purpose is as follows:

```
EVALUATE TRUE
        WHEN MONTH =  4  OR  6  OR  9  OR  11
                MOVE    30   TO     NO-OF-DAYS
        WHEN  MONTH = 2
                MOVE    28    TO    NO-OF- DAYS
        WHEN OTHER MOVE   31  TO  NO-OF-DAYS
 END EVALUATE.
```

In this case, we have assumed that MONTH has a correct value.

Evaluate Example Number 4:

Suppose MARKS contains the marks obtained by a student. GRADE is an one-character alphanumeric field. We wish to calculate GRADE according to the following rules

MARKS	GRADE
80 – 100	A
60 - 79	B
45 - 59	C
30 - 44	D
0 - 29	E

The EVALUATE statement for the purpose is shown below.

```
EVALUATE MARKS
        WHEN   80     THRU      100   MOVE  “A”  TO GRADE
        WHEN   60     THRU       79   MOVE  “B”  TO GRADE
        WHEN   45     THRU       59   MOVE  “C”  TO GRADE
        WHEN   30    THRU        44   MOVE  “D”  TO GRADE
        WHEN   ZERO  THRU  29         MOVE  “E”  TO GRADE
        WHEN OTHER   MOVE  “W”  TO  GRADE
END-EVALUATE.
```

The literal “W” is moved to GRADE in the case of wrong marks.

ILLUSTRATES CONDITION NAMES

```
DATA DIVISION.

WORKING-STORAGE SECTION.

77    MARTIAL-STATUS     PIC 9.
88    SINGLE                          VALUE 0.
88    MARRIED                         VALUE 1.
88    WIDOWED                         VALUE 2.
88    DIVORCED                        VALUE 3.
88    ONCE-MARRIED                    VALUES ARE 1, 2, 3.
88    VALID-STATUS                    VALUES ARE 0 THRU 3.
77    AMOUNT             PIC 9 (4)    VALUE 1000.

PROCEDURE DIVISION.

MAIN-PARA.
      DISPLAY “Martial Status:”
      DISPLAY “0- Single / 1- Married / 2- Widowed / 3- Divorced”.
      ACCEPT MARTIAL-STATUS.
      IF NOT VALI-STATUS DISPLAY “Error in Entry”.
      IF SINGLE SUBTRACT 100 TO AMOUNT.
      IF MARRIED ADD 100 TO AMOUNT.
      IF WIDOWED ADD 200 TO AMOUNT.
      IF DIVORCED SUBTRACT 200 FROM AMOUNT.
      IF ONCE-MARRIED ADD 250 TO AMOUNT
      DISPLAY AMOUNT.
      STOP RUN.
```

Figure 3-36 EVALUATE Statement

INITIALIZE Statement

The INITIALIZE statement sets selected categories of data fields to predetermined values. It is functionally equivalent to one or more MOVE statements.

When the REPLACING phrase is not used:
SPACE is the implied sending field for alphabetic alphanumeric, alphanumeric-edited, and DBCS items.
ZERO is the implied sending field for numeric and numeric-edited items.

```
>>___INITIALIZE____identifier-
1_________________________________________________________________>
>___    ____________________________________________________________ _________________
_______><
       |               < _____________________________________________ __________________   |
       |_REPLACING____  _ALPHABETIC_______   __  _______  __BY____   identifier-2  _  _   | _ |
                         |_ALPHANUMER____|     |_DATA_|              |_LITERAL-1__|
                         |_NUMERIC __________|
                         |_ALPHANUMERIC-EDITED_|
                         |_NUMERIC-EDITED__|
                         |_ DBCS _____________|
                         |_ EGCS _____________|
```

Figure 3-38 INITIALIZE Statement

Notes:

The INITIALIZE statement sets selected categories of data fields to predetermined values. It is functionally equivalent to one or more MOVE statements.

A subscripted item can be specified for identifier-1. A complete table can be initialized only by specifying identifier-1 as a group that contains the complete table.

The data description entry for identifier-1 must not contain a RENAMES clause. An index data item cannot be an operand of INITIALIZE.

Special registers can be specified for identifier-1 and identifier-2 only if they are valid receiving fields or sending fields, respectively, for the implied MOVE statement(s).

When the REPLACING phrase is used:

- The category of identifier-2 or literal-1 must be compatible with the category indicated in the corresponding REPLACING phrase, according to the rules for the NUMERIC category.
- The same category cannot be repeated in a REPLACING phrase.
- The Key word following the word REPLACING corresponds to a category of data shown “Classes of Data” visual.

SET TO TRUE Statement

When this form of the SET statement is executed, the value associated with a condition-name is placed in its conditional variable according to the rules of the VALUE clause.

```
>>__SET____condition-name-1_|_ TO TRUE________________________><
```

condition-name-1: Must be associated with a conditional variable.

If more than one literal is specified in the VALUE clause of condition-name-1, its associated conditional variable is set equal to the first literal.

```
01  CUST-TYPE      PIC  99.
            88  INACTIVE       VALUE  9.
            88  SPEC-ACCTS VALUE   20, 11, 40, 44.
……
      SET INACTIVE  TO TRUE
      SET SPEC-ACCTS TO TRUE
```

Figure 3-37 SET TO TRUE Statement

Notes:

Class Condition

- **NUMERIC**
  - **The item entirely contains characters 0 through 9 (with or without a sign determined by its PICTURE clause). It may be USAGE DISPLAY or PACKED DECIMAL.**

- **ALPHABETIC**
  - **The entire item contains only A through Z, a through z, or spaces**

- **ALPHABETIC-UPPER**
  - **The entire item contains only A through Z (exclusively upper-case) or spaces.**
- **ALPHABETIC-LOWER**
  - **The entire item contains only a through z (exclusively lower-case) or spaces.**

Figure 3-33 Class Conditionals

Notes:

Ex:

```
 1. IF  A  IS   NUMERIC
        ----------------------
        -----------------------

2.   IF  C IS ALPHABETIC
            -----------------------
             ------------------------
```

Where A and C are Data items.