B. Heinz - Computer

SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS UPDATE

General

Fundamentals of COBOL Table Handling

Programmer Reference

UP-7503.2 Rev. 1–A

This Library Memo announces the release and availability of Updating Package A to "SPERRY UNIVAC Fundamentals of COBOL Table Handling Programmer Reference", UP-7503.2 Rev. 1.

This update includes a minor correction to the document.

Copies of Updating Package are now available for requisitioning. Either the updating package only or the complete manual with the updating package may be requisitioned by your local Sperry Univac representative. To receive only the updating package, order UP-7503.2 Rev. 1–A. To receive the complete manual, order UP-7503.2 Rev. 1.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists 1D, 3D, 4D, 5D, 8, 9, 10, 11, 18, 18U, 19, 19U, 20, 20U, 21, 21U, 37, 38, 43, 44, 51, 52, 53, 54, 55, 56, 57, 58, 60, 61, 62, 63, 64, 65, 66, 75, 75U, 76, 76U, 81, 83, 89 (Package A to UP-7503.2 Rev. 1, 5 pages plus Memo)	Library Memo RELEASE DATE: January, 1981

UNIVAC®

FUNDAMENTALS OF COBOL

TABLE HANDLING

PROGRAMMERS REFERENCE

UP-7503.2 Rev. 1

PAGE STATUS SUMMARY

ISSUE: Update A – UP-7503.2 Rev. 1

Part/Section	Page Number	Update Level
Cover/Disclaimer		Orig.
PSS	1	A
Preface	1	Orig.
Contents	1	Orig.
1	1, 2	Orig.
2	1 thru 6	Orig.
3	1 thru 8 9 10, 11	Orig. A Orig.
User Comment Sheet		

Part/Section	Page Number	Update Level

Part/Section	Page Number	Update Level

All the technical changes are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

PREFACE

This manual is another in the series of manuals entitled ''Fundamentals of COBOL''. It is an extension of the basic manual of this series, ''Fundamentals of COBOL – Language''. As with other volumes of this series, it does not represent the COBOL implementation for any particular computer system; rather, it is intended as a basic reference source to acquaint the reader with the basic attributes of the COBOL Table Handling Feature. The information in this manual is oriented toward level 3 of the USASI COBOL standard for Table Handling (see discussion of COBOL levels and modules in ''Fundamentals of COBOL – Language''.

The purposes of this manual are:

(a) to introduce the concept of the table by means of a definition and a general explanation,

(b) to discuss the techniques for COBOL table definition and table handling, and

(c) to describe those specific features in COBOL available to the user for the purpose of table definition and table handling.

CONTENTS

1. INTRODUCTION

1.1. GENERAL

Data processing problems frequently involve the handling of sets of values which can be most readily represented and interpreted when shown in table form.

A table is a named set of data items. These items are arranged in some logically meaningful and consecutive order, and are functionally homogeneous in some definite sense.

There are three basic reasons for arranging homogeneous data in table form:

- From a documentational standpoint, it is often desirable to arrange a body of data items such that the underlying homogeneity of the several items is readily apparent.
- Each item in a table occupies a definite physical position relative to the base or origin location of the table. Thus each item in the table is addressed relative to the table-name, rather than by a unique data-name.
- The number of occurrences of a table-element may be specified as fixed or variable.

1.2. DEFINING A TABLE

COBOL tables are defined structurally by including the OCCURS clause in the data description entry.

The OCCURS clause specifies the number of times an item is to be repeated. An item described by an OCCURS clause is called a table-element, and the name and description of the table-element applies to each repetition or occurrence.

To define a one-dimensional table, an OCCURS clause is written as a part of the data description of the item to be repeated. Example 1 shows a one-dimensional table defined by the item TABLE-ELEMENT.

Example 1:

```
01      TABLE-1.
        02 TABLE-ELEMENT   OCCURS 20 TIMES
           03 SAM . . .
           03 JOE . . .
```

Each of twenty TABLE-ELEMENTS contains one SAM and one JOE. To address any individual TABLE-ELEMENT, or any SAM or JOE contained within, it is necessary to specify the occurrence number of the TABLE-ELEMENT. The complete set of occurrences of TABLE-ELEMENT has been assigned the name TABLE-1. However, it is not necessary to give a group name to a table unless a means of addressing the entire table is desired.

Defining a one-dimensional table within each occurrence of an element of another one-dimensional table gives rise to a two-dimensional table, as shown in Example 2.

Example 2:

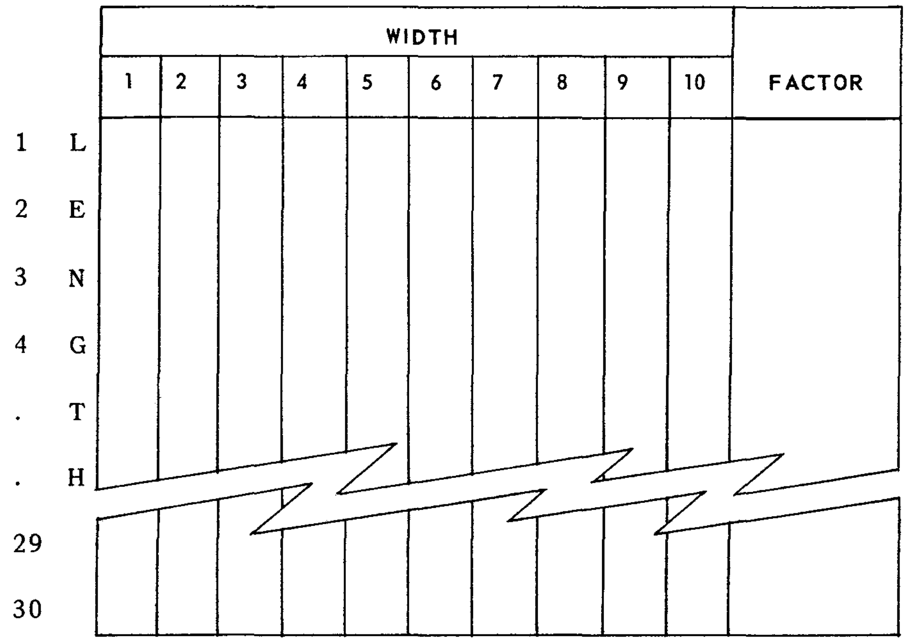
```
02      LENGTH   OCCURS 30 TIMES; . . .

        05 WIDTH   OCCURS 10 TIMES;

        05 FACTOR . . .
```

LENGTH is an element of a one-dimensional table. But within each LENGTH occurrence there are 10 occurrences of WIDTH. Thus WIDTH is an element of a two-dimensional table. However, FACTOR occurs only once within each LENGTH, so that it is an element of a one-dimensional table.

Pictorially, this table might be represented as follows:



In general, to define an n-dimensional table, the OCCURS clause must appear in the data description of the table-element and also appear in n-1 group items which contain that table-element as a subordinate item. In COBOL tables, up to three dimensions are permitted.

2. REFERENCE OF TABLE-ITEMS

2.1. GENERAL

When referencing a table-item it is necessary to indicate either explicitly (by subscripting) or implicitly (by indexing) which occurrence of the table-item is intended. The table-item can be one of the following:

- a table-element,
- an item that is within a group-item table-element,
- a condition-name associated with a table-element, or
- a condition-name associated with an item that is within a group-item table-element.

When subscripting or indexing is used to reference a table-item, the subscript or index (or set of subscripts or indices) is enclosed by parentheses. It is written immediately following the data-name, or table-item to which the subscripting or indexing is applied in that particular reference. The form is:

Data-name-1 (a_1[,a_2[,a_3]])

where each *a* represents an occurrence number of a dimension of the table, expressed either in the form of an index-name or a subscript. *Subscripts* are integer values, expressed either by numeric literals or data-names which contain integer values. An *index-name*, is a word, which contains at least one alphabetic character, that names an index (pointer) associated with a specific table.

One to three subscript or index levels may appear in the reference, depending upon how many dimensions are in that table of which the data-name has been defined to be an element. There must be one subscript or index level for each OCCURS clause in the defined hierarchy that contains data-name-1, including that for data-name-1. When more than one subscript or index is used in a reference, each must be separated within the parentheses by a comma and a space.

Multiple subscripts and indices are written left to right in descending order of inclusiveness, higher to lower. A multi-dimensional table may be thought of as a group of nested single-dimensional tables, with the outermost table the most inclusive and the innermost table the least inclusive.

The example below shows a typical multi-dimensional table and explains the manner in which its elements are addressed. Complete data description entries are not shown.

Example:

```
01  TABLE

02  CLASS OCCURS 12 TIMES

    03  RATE . . .

    03  LIMIT OCCURS 7  TIMES

        04  HIGH . . .

        04  LOW . . .

        04  FACTOR OCCURS 4 TIMES
```

In the above example reference to CLASS or RATE (which is within the table-element CLASS) requires only one subscript or index because CLASS is the most inclusive table-element, and RATE occurs only once for each CLASS. References to LIMIT, HIGH and LOW require two subscripts, since the proper occurrence number of CLASS must be specified and so must be the proper occurrence of LIMIT. Similarly, since there are several FACTORS within each LIMIT, reference to FACTOR requires three subscripts: one to specify the proper CLASS, one to show the proper LIMIT, and one to show the proper FACTOR. However, only one HIGH and one LOW occur within each LIMIT, and references to HIGH or LOW require the specification of only two subscripts: for CLASS and for LIMIT.

2.2. SUBSCRIPTING

One method by which occurrence numbers may be specified is to append one, two, or three subscripts to the data-name. A subscript is an integer value which specifies the occurrence number of the table-element to which the data-name refers, within the group item that has the next lower level number.

The subscript can be represented either as a numeric literal or as a data-name that is defined elsewhere as an elementary numeric data item.

The lowest valid subscript is 1. The highest valid subscript is the maximum occurrence number of the item, as specified in the OCCURS clause. A subscript is invalid if the item it references is neither an element of a defined table nor an item or condition-name within an element of a defined table.

Data-names may be used freely in place of numeric literal subscripts, and may be mixed with literal subscripts within an item reference; e.g., FACTOR (7, A-KEY, 3), where data-name A-KEY contains an integer indicating which occurrence of RANGE is intended. A single data-name may be used to refer to items in more than one table, and the tables need not have elements of the same size. Also, the same data-name may be the only subscript with one item reference, while being one of two or three subscripts with another item reference.

2.3. INDEXING

In addition to subscripting, individual items in a table may be referred to by indexing. With this technique, the programmer assigns one or more index-names, using the optional INDEXED BY clause, to an item whose data description includes an OCCURS clause.

No separate data description entry is needed to describe the index-name because the size and radix of each index-name is predetermined by the hardware.

At object time the value of an index-name corresponds to an occurrence number for the dimension of the table to which the index-name was assigned. The manner of correspondence is determined by the specific implementation.

The index-name must be initialized by a SET statement (see 3.3.3) before it can be used as a table reference.

References are made to individual items within a table of homogeneous elements by specifying the name of the item, followed by its related index-name(s) in parentheses. Each occurrence number required to complete the reference will be obtained from the respective index-name, the index-name acting as a subscript.

An index can be modified only by the PERFORM, SEARCH, and SET statements (described in 3.3.1, 3.3.2, and 3.3.3 respectively). Data items described by the USAGE IS INDEX clause permit storage of the values of index-names as data, without conversion. Such data items are called index data items.

Direct indexing is the use of an index-name in the form of a subscript. Relative indexing may also be specified by following the index-name with an operator (+ or –) and an integer. The occurrence number, then, is the same as if the integer were added to or subtracted from the occurrence number to which the setting of the index-name corresponds. Relative indexing does not cause any altering of the index-name value.

For example, in the reference:

```
LIMIT (XLZONE + 2)
```

The index-name XLZONE does not change in value, but if the value of XLZONE is set at, say, the third occurrence of LIMIT, the fifth occurrence will actually be addressed.

An index-name cannot be defined as part of a file, and therefore cannot be directly modified by use of input/output statements. However, a data item in a file can be described by a USAGE IS INDEX clause (see 3.2.2), and this data item value can be transferred, without conversion, to an index-name by means of the SET statement (see 3.3.3).

A reference to an item may be indexed only by an index-name which has been defined (through use of the INDEXED BY clause) as being associated with the table of which that item is an element.

2.4. SUMMARY OF GENERAL RULES FOR SUBSCRIPTING AND INDEXING

- The use of a data-name subscript in any reference to a table-element or to an item within a table-element will not cause any index-name to be altered by the object program.
- Index-names may not be combined with subscripts in a single data-name reference.
- Where subscripting is not permitted, indexing is also not permitted.
- Tables may have one, two, or three dimensions. Thus, a reference to a table-element may require up to three subscripts or indices.
- A data-name may be neither subscripted nor indexed when the data-name is itself used as a data-name subscript or index-name in that particular reference.
- An occurrence number specified by a subscript or implied by an index-name must not be less than 1 nor greater than the highest permissible occurrence number for the table-element.
- When indexing is used to reference a table, the INDEXED BY option must be employed in each OCCURS clause used to define the table. The SEARCH and SET statements appear in the Procedure Division only in connection with indexed table references.

2.5. SUBSCRIPTING AND INDEXING – A COMPARATIVE SAMPLE PROBLEM

To clarify the difference between the subscripting and indexing techniques, a simple table handling problem will be solved by each method. Consider the following two tables:

```
01  LIABILITY-RATES.
    02  TERRITORY-L OCCURS 9 TIMES.
        03  BASE-PREM PICTURE IS 9(3).
        03  CLASS - DIFFERENTIAL OCCURS 7  TIMES.
            04  LIMIT-FACTOR OCCURS 5 TIMES PICTURE IS 9(4).

01  PHYS-DAM-RATES.
    10  TERRITORY-P OCCURS 9  TIMES.
        15  COMP-ACV-BASE PICTURE IS 9(3).
        15  COMP-50D-BASE PICTURE IS 9(3).
        15  COLL-50D-BASE PICTURE IS 9(3).
        15  COLL-100D-BASE PICTURE IS 9(3).
        15  COMPOSITE-FACTOR OCCURS 196 TIMES PICTURE IS 9(4).
```

These two tables are simplified versions of what might be used in developing automobile insurance premiums. If it is known that the territorial definition is the same for both liability and automobile physical damage premiums, then the same data-name subscript can be used in determining the appropriate set of territorial values for each table.

Assume that an input file record of an individual policy is to be updated, and the input record includes the following elementary data:

TERR–IN

LIAB–CLASS–CODE–IN

LIAB–LIMIT–IN

CLASS–IN

SYMBOL–IN

AGE–IN

TERR–IN is the territory number. LIAB–CLASS–CODE–IN represents a class code in the range 1 to 7. LIAB–LIMIT–IN represents a coverage limit code number in the range 1 to 5. CLASS–IN, SYMBOL–IN, and AGE–IN represent three code numbers which, when multiplied, produce an occurrence number which points to 1 of 196 composite factors (effectively, a table within a table, although not formally treated as such).

Although the two tables have different numbers of dimensions and different element sizes, TERR–IN can be used to refer to the proper territorial occurrence in either table.

For example:

> LIMIT–FACTOR (TERR–IN, LIAB–CLASS–CODE–IN, LIAB–LIMIT–IN)
>
> or
>
> TERRITORY–P (TERR–IN)

are both valid uses of the TERR–IN contents as a subscript.

Also, the following procedure:

(1) MULTIPLY CLASS–IN BY SYMBOL–IN GIVING SCRATCH.

(2) MULTIPLY AGE–IN BY SCRATCH.

(3) MULTIPLY COMPOSITE–FACTOR (TERR–IN, SCRATCH) BY COLL–50D–BASE (TERR–IN) GIVING COLL–PREMIUM.

The first two multiplications put the appropriate occurrence number for COMPOSITE–FACTOR into SCRATCH. Then the proper COMPOSITE–FACTOR is multiplied by the COLL–50D–BASE ($50-deductible base premium) of the particular territory to get the final collision coverage premium into COLL–PREMIUM (a field in the output billing file for this hypothetical application).

This illustrates some of the organizational and technical aspects of handling tables by the subscripting technique.

Using the same table formats, if indexing were used rather than subscripting, the files might be defined as follows:

```
01 LIABILITY-RATES
   02 TERRITORY-L OCCURS 9 TIMES INDEXED BY XTL.
      03 BASE-PREM PICTURE IS 9(3).
      03 CLASS-DIFFERENTIAL OCCURS 7 TIMES INDEXED BY XCD.
         04 LIMIT FACTOR OCCURS 5 TIMES INDEXED
            BY XLF PICTURE IS 9(4).

01 PHYS-DAM-RATES
   10 TERRITORY-P OCCURS 9 TIMES INDEXED BY XTP.
      15 COMP-ACV-BASE PICTURE IS 9(3).
      15 COMP-50D-BASE PICTURE IS 9(3).
      15 COLL-50D-BASE PICTURE IS 9(3).
      15 COLL-100D-BASE PICTURE IS 9(3).
      15 COMPOSITE-FACTOR OCCURS 196 TIMES PICTURE IS 9(4)
         INDEXED BY XCF.
```

Indexing cannot be used to reference a table-element (i.e., an item described by an OCCURS clause) unless the INDEXED BY clause appears in the data description of the item and in any table group-items to which the table-element is subordinate. For example, COMPOSITE-FACTOR is INDEXED BY XCF. Thus TERRITORY-P (which has nine occurrences, each containing 196 occurrences of COMPOSITE – FACTOR) must also be indexed.

The handling of tables by indexing is similar to subscripting, except that indices must be modified and controlled by means of SET, SEARCH, and PERFORM statements. Data items that were used as subscripts in the subscripting example (such as TERR-IN, LIAB-LIMIT-IN, SCRATCH, etc.) would have to be converted by means of a SET statement.

For example:

```
SET XTL, XTP TO TERR-IN.
```

If the data-name TERR-IN is described by a USAGE IS INDEX clause it is moved without conversion to index-names XTL and XTP. If TERR-IN is a data-name not described by a USAGE IS INDEX clause, the compiler interprets the value in TERR-IN as a subscript value which must be converted to the appropriate index value.

3. COBOL TABLE HANDLING FEATURES

3.1. GENERAL

This section describes the specific formats and usage in COBOL for the definition and referencing of tables.

3.2. DATA DIVISION

The COBOL Table Handling feature provides two additional facilities for the Data Division of the COBOL source program: the OCCURS clause, and the USAGE IS INDEX format option of the USAGE clause.

3.2.1. **OCCURS**

Format:

Option 1:

```
OCCURS integer-2 TIMES [ {ASCENDING } KEY IS data-name-2
                         {DESCENDING}
[, data-name-3] . . . ] . . . [INDEXED BY index-name-1 [, index-name-2] . . . ]
```

Option 2:

```
OCCURS integer-1 TO integer-2 TIMES [DEPENDING ON data-name-1]
   [ { ASCENDING  } KEY IS data-name-2 [, data-name-3] . . . ] . . .
     { DESCENDING }
[ INDEXED BY index-name-1 [, index-name-2] . . . ]
```

Description:

The OCCURS clause is used for defining sets of repeated homogeneous data such as tables, thus eliminating the writing of separate entries for repeated data. It also supplies information required for the application of subscripts or indices in the handling of tables. All data description clauses associated with an item whose description includes an OCCURS clause apply to each repetition of the item so described.

The OCCURS clause must not be specified in a data description entry that:

(1) has an 01 or a 77-level number; or

(2) describes an item whose size is variable. The size of an item is defined as variable whenever the data description of an item subordinate to it contains an Option 2 OCCURS clause.

Integer-1 and integer-2 are positive integers. When both are used, integer-1 must be numerically less than integer-2, and may be zero. In Option 1 integer-2 represents the exact number of occurrences, and in Option 2 the maximum number of occurrences. Data-names described by an OCCURS clause must be subscripted or indexed whenever used as an operand, regardless of the number of occurrences.

Data-name-2 must be either the name of the entry that contains the OCCURS clause or the name of an entry subordinate to the entry containing the OCCURS clause. Data-name-3, etc., applies only in the latter case.

The KEY IS option is used to indicate whether repeated data is arranged in ASCENDING or DESCENDING order, according to values contained in data-name-2, data-name-3, etc. Data-names are listed in descending order of hierarchal significance (i.e., most inclusive through least inclusive). If data-name-2 is not the subject of this entry, then:

(1) All entries identified by data-name-2, data-name-3, etc. must be subordinate to the group item which is the subject of this entry.

(2) None of the data-names appearing in the KEY IS option can contain an entry that contains an OCCURS clause nor may they be subordinate to an entry containing an OCCURS clause.

The DEPENDING option is required only when the end of the occurrences cannot be otherwise determined. Data-name-1 can be qualified. Its data description must define it as a positive integer representing the occurrence count, and its value must not exceed that of integer-2. The implementor must specify whether the unused character positions that result from the use of the DEPENDING option are to appear on the external media.

The INDEXED BY clause is required if the subject of the entry to which this clause refers, or an item within it (if a group-item), is to be referred to by means of indexing. The index-name is not defined elsewhere, since it does not constitute a data-name.

Reference to data-names described by the OCCURS clause must always be either subscripted or indexed, except when the reference is done by means of a SEARCH statement (see 3.3.2). Also, any items subordinate to data-names so described must be subscripted or indexed.

If data-name-1 is an entry in the record described by this OCCURS clause, data-name-1 must not be the subject of, or subordinate to, an entry whose description contains an Option 2 OCCURS clause. An entry which contains or has a subordinate entry which contains Option 2 cannot be the object of a REDEFINES clause.

Except for condition-name entries, a VALUE clause must not be used in a Record Description entry which either contains an OCCURS clause or is subordinate to an entry that contains an OCCURS clause.

3.2.2. USAGE

Format:

USAGE IS <u>INDEX</u>

Description:

The USAGE clause specifies the format of a data item in computer storage. The USAGE IS INDEX option can be written at any level. When it is used to describe an elementary item, the item is called an index data item. This item contains a value which corresponds to the occurrence number of a table-element.

If the USAGE IS INDEX clause describes a group, the elementary items that comprise the group are all index data items. However, the group itself is not an index data item and it must not be specified in SEARCH or SET statements or in a relation condition.

The actual value assigned to an index data item may be dependent upon the description of the table-element. In any case, the method of representation of this value (with respect to its manner of correspondence with the occurrence number) is determined by the specific implementation. The index data item cannot be a conditional value.

An index data item can be referred to directly only by means of a SEARCH or SET statement (see 3.3.2 and 3.3.3) or in a relation condition with an index-name.

An index data item can be part of a group that is referred to as an operand in a MOVE or input/output statement, in which case no conversion takes place.

The SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE, or editing clauses must be used to describe group or elementary items described by a USAGE IS INDEX clause.

3.3. PROCEDURE DIVISION

COBOL Table Handling provides three features for the Procedure Division in addition to those of the basic language: the VARYING option of the PERFORM statement, the SEARCH statement, and the SET statement.

3.3.1. PERFORM

Format:

$$\underline{\text{PERFORM}}\ \textit{procedure-name-1} \left[\underline{\text{THRU}}\ \textit{procedure-name-2}\right]$$

$$\underline{\text{VARYING}} \left\{\begin{array}{l}\textit{index-name-1}\\ \textit{identifier-1}\end{array}\right\} \underline{\text{FROM}} \left\{\begin{array}{l}\textit{index-name-2}\\ \textit{literal-1}\\ \textit{identifier-2}\end{array}\right\}$$

$$\underline{\text{BY}} \left\{\begin{array}{l}\textit{literal-2}\\ \textit{identifier-3}\end{array}\right\} \underline{\text{UNTIL}}\ \textit{condition-1}$$

$$\Bigg[\ \underline{\text{AFTER}} \left\{\begin{array}{l}\textit{index-name-3}\\ \textit{identifier-4}\end{array}\right\} \underline{\text{FROM}} \left\{\begin{array}{l}\textit{index-name-4}\\ \textit{literal-3}\\ \textit{identifier-5}\end{array}\right\}$$

$$\underline{\text{BY}} \left\{\begin{array}{l}\textit{literal-4}\\ \textit{identifier-6}\end{array}\right\} \underline{\text{UNTIL}}\ \textit{condition-2}$$

$$\Bigg[\ \underline{\text{AFTER}} \left\{\begin{array}{l}\textit{index-name-5}\\ \textit{identifier-7}\end{array}\right\} \underline{\text{FROM}} \left\{\begin{array}{l}\textit{literal-5}\\ \textit{index-name-6}\\ \textit{identifier-8}\end{array}\right\}$$

$$\underline{\text{BY}} \left\{\begin{array}{l}\textit{literal-6}\\ \textit{identifier-9}\end{array}\right\} \underline{\text{UNTIL}}\ \textit{condition-3}\ \Bigg]\Bigg]$$

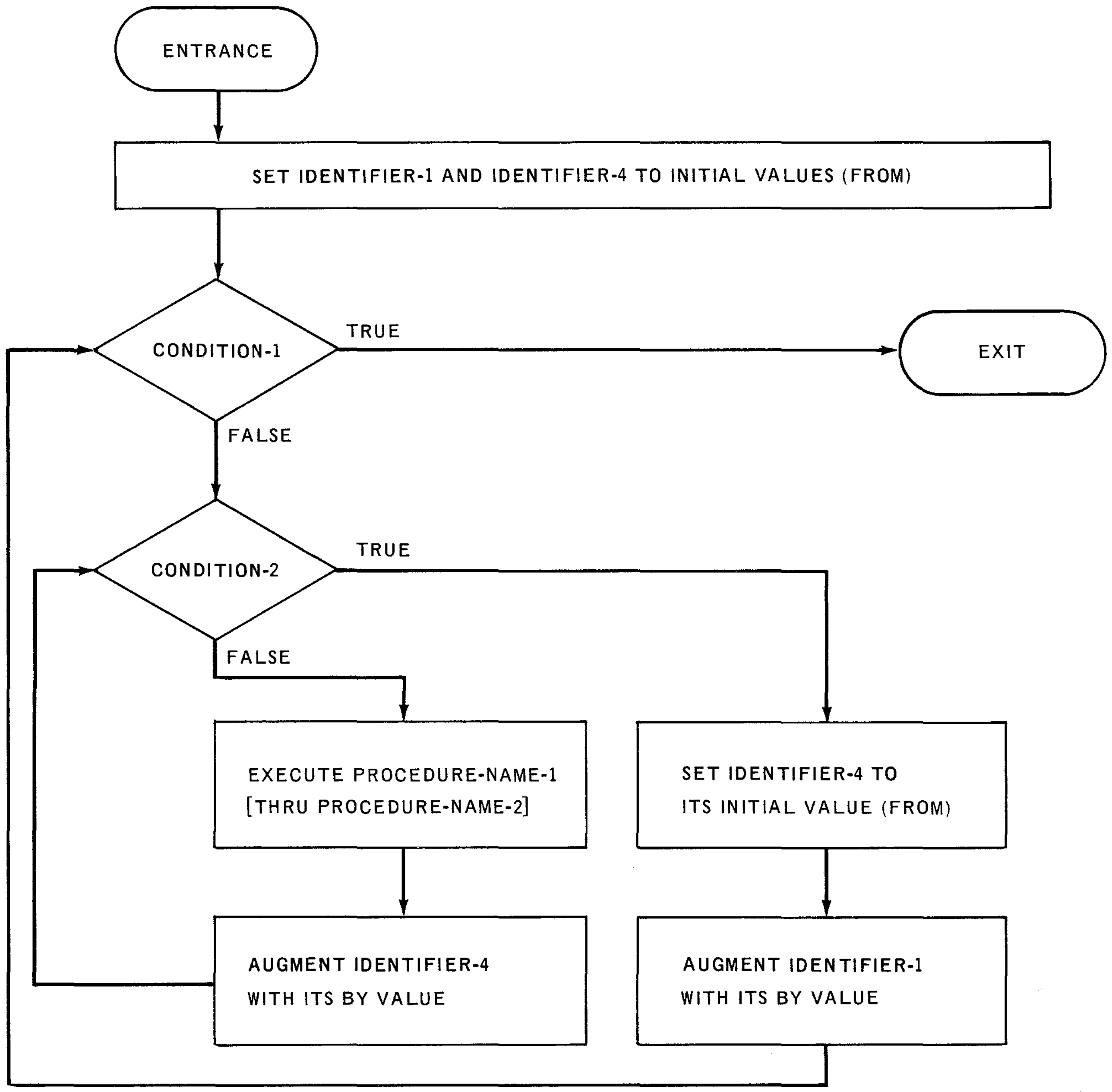
Description:

This format of the PERFORM statement is an extension of the VARYING data-name option (Option 4) discussed in the first manual of this series, "Fundamentals of COBOL–Language".

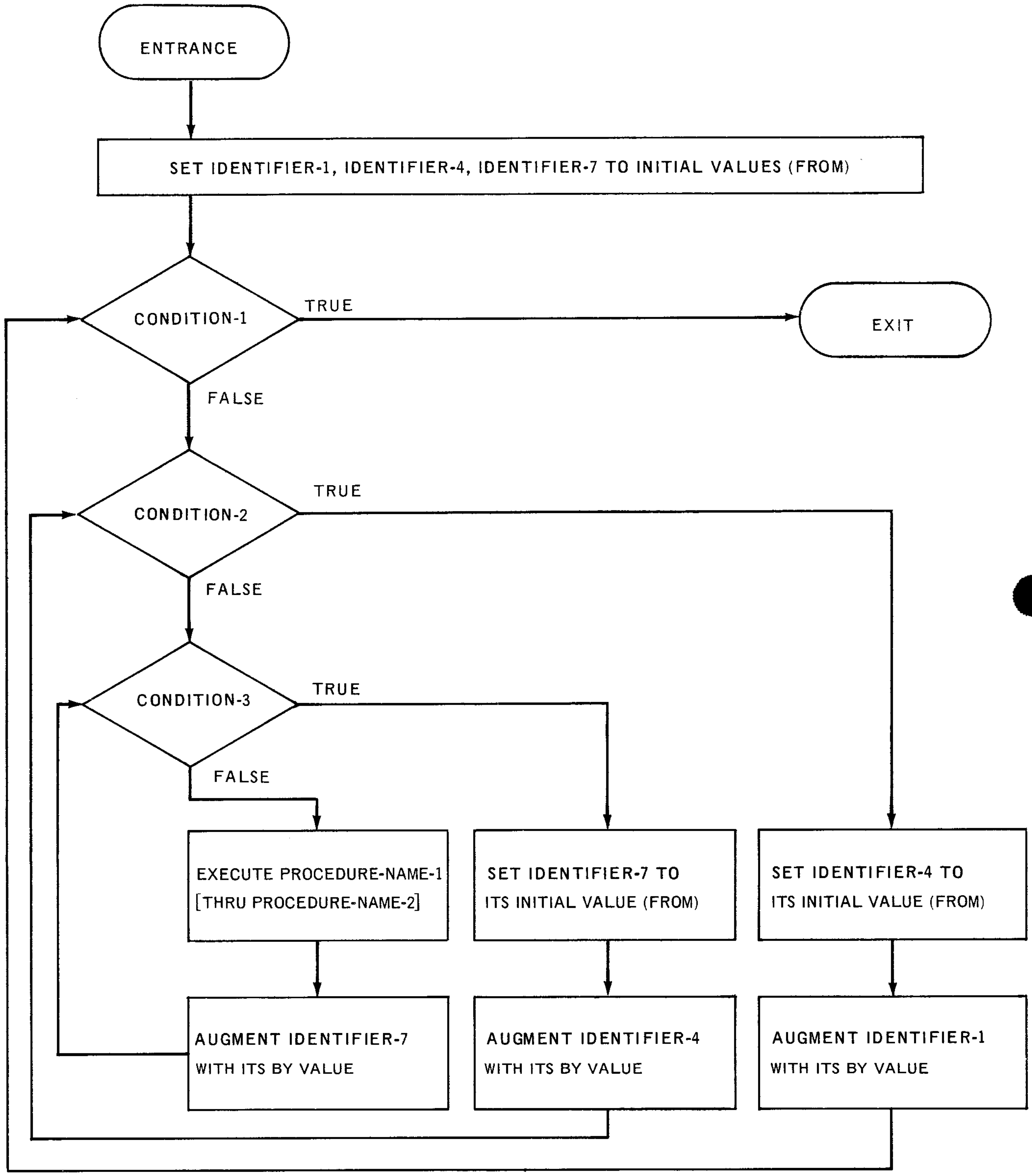
A maximum of three subscript-names or index-names can be varied in this option.

When only one subscript (or index) is being varied, the mechanism is exactly the same as that of the VARYING data-name option (Option 4 in "Fundamentals of COBOL–Language"). When two subscripts (or indices) are varied, the value of identifier-4 (or index-name-3) goes through a complete cycle (FROM, BY, UNTIL) each time that identifier-1 (or index-name-1) is augmented with its BY value. The PERFORM is completed as soon as condition-2 is found to be true. When three subscripts are varied, the value of identifier-7 (or index-name-5) goes through a complete cycle (FROM, BY, UNTIL) each time that identifier-4 (or index-name-3) is augmented with its BY value. Further, identifier-4 (or index-name-3) goes through a complete cycle (FROM, BY, UNTIL) each time that identifier-1 (or index-name-1) is augmented with its BY value. The PERFORM is completed as soon as condition-1 is found to be true. Regardless of the number of subscripts (or indices) being varied, as soon as condition-1 is found to be true, control is transferred to the next statement after the PERFORM statement. The FROM value must be a positive, nonzero integer. The BY value must be a nonzero integer. No two identifiers (or index-names) may reference the same item (i.e., they must not be alternative names for the same index data item). Diagrams for this mechanism follow:

TWO SUBSCRIPTS


ENTRANCE
SET IDENTIFIER-1 AND IDENTIFIER-4 TO INITIAL VALUES (FROM)
CONDITION-1
TRUE
EXIT
FALSE
CONDITION-2
TRUE
FALSE
EXECUTE PROCEDURE-NAME-1 [THRU PROCEDURE-NAME-2]
SET IDENTIFIER-4 TO ITS INITIAL VALUE (FROM)
AUGMENT IDENTIFIER-4 WITH ITS BY VALUE
AUGMENT IDENTIFIER-1 WITH ITS BY VALUE

THREE SUBSCRIPTS


ENTRANCE
SET IDENTIFIER-1, IDENTIFIER-4, IDENTIFIER-7 TO INITIAL VALUES (FROM)
CONDITION-1
TRUE
EXIT
FALSE
CONDITION-2
TRUE
FALSE
CONDITION-3
TRUE
FALSE
EXECUTE PROCEDURE-NAME-1 [THRU PROCEDURE-NAME-2]
SET IDENTIFIER-7 TO ITS INITIAL VALUE (FROM)
SET IDENTIFIER-4 TO ITS INITIAL VALUE (FROM)
AUGMENT IDENTIFIER-7 WITH ITS BY VALUE
AUGMENT IDENTIFIER-4 WITH ITS BY VALUE
AUGMENT IDENTIFIER-1 WITH ITS BY VALUE

After the completion of the PERFORM, identifier-4 (or index-name-4) and identifier-7 (or index-name-5) will each have a value equal to their respective initial settings (FROM); while identifier-1 (or index-name-1) will have a value which differs from its last used setting by one increment (or decrement).

3.3.2. SEARCH

Format:

Option 1:

```
SEARCH identifier-1 [VARYING { index-name   }]
                             { identifier-2 }

[; AT END imperative-statement-1]

 ; WHEN condition-1 { imperative-statement-2 }
                    { NEXT SENTENCE          }

[; WHEN condition-2 { imperative-statement-3 }] . . .
                    { NEXT SENTENCE          }
```

Option 2:

```
SEARCH ALL identifier-1 [;AT END imperative-statement-1]

 ; WHEN condition-1 { imperative-statement-2 }
                    { NEXT SENTENCE          }
```

Description:

The execution of a SEARCH statement causes a table to be searched for a table-element that satisfies a condition specified within the WHEN clause of the SEARCH statement. When the condition is satisfied, the index-name associated with the table is adjusted so as to reflect the location of the located element.

In either format, identifier may not be subscripted or indexed, but must contain an OCCURS clause and an INDEXED BY clause. In addition, the description of identifier in Option 2 must contain the KEY IS option in its OCCURS clause.

Option 1 causes a serial type of search, as follows:

(a) If, at the start of the SEARCH, the current index setting of the index-name associated with identifier-1 corresponds to an occurrence number that is greater than the highest permissible occurrence number of identifier-1, the SEARCH is terminated immediately. Then if the AT END clause is specified imperative-statement-1 is executed; if the AT END clause is not specified control passes to the next statement in sequence.

(b) If, at the start of the SEARCH, the current index setting of the index-name associated with identifier-1 corresponds to an occurrence number that is not greater than the highest permissible occurrence number for identifier-1 the SEARCH proceeds by evaluating the conditions in the order in which they are written. Index-name values, where specified, determine the occurrence of the items that are to be tested. If none of the stated conditions are satisfied, the index-name is incremented to obtain reference to the next occurrence. The process is repeated, using new index-name settings, unless the index-name setting for identifier-1 indicates an occurrence number which exceeds the last element limit of the table by one or more occurrences; in this case the SEARCH operation terminates as explained in (a) .

The SEARCH operation terminates when one of the conditions is satisfied. The index-name or identifier remains set at the occurrence on which the condition was satisfied, and the imperative statement associated with that condition is executed.

Identifier-2, when specified, must be described as USAGE IS INDEX or else be the name of a numeric elementary item of integer value.

Option 2, or the SEARCH ALL option, causes a nonserial type of search to be employed as follows:

The initial index setting is ignored; it is varied during the SEARCH operation as determined by the implementor. At no time is it set such that the occurrence number to which it corresponds exceeds the occurrence number of the last element of the table, nor is it less than the value corresponding to the first element of the table.

If condition-1 cannot be satisfied for any setting of the index within the permitted range, control passes to imperative-statement-1, if the AT END clause is present. If the AT END clause is not present, control passes to the next statement in sequence. In either case the final setting of the index is not predictable.

If condition-1 is satisfied at some point, the index setting corresponds to the occurrence number of the table-element that satisfies condition-1. Control then passes to imperative-statement-2 of the WHEN clause, if specified; otherwise, control passes to the next statement in sequence.

In the VARYING option of Option 1, identifier-2 when specified must be described as USAGE IS INDEX, or as the name of an elementary data item integer. Identifier-2 is incremented at the same time and by the same amount as the occurrence number represented by the index-name associated with identifier-1. Index-name, when specified, is used for the search if index-name appears in the INDEXED BY clause of identifier-1. Otherwise, the first (or only) index-name given in the INDEXED BY clause of identifier-1 is used.

If index-name also appears in the INDEXED BY clause of another table entry, the occurrence number represented by this index-name is incremented along with the occurrence number represented by the index-name associated with identifier-1.

If identifier-1 belongs to a hierarchy of groups, each described by an OCCURS clause, then each of those groups must also have an associated index-name. The settings of these index-names are used throughout the execution of the SEARCH statement to refer to identifier-1 or items contained within it. However, only index-name-1 is actually modified. Only the index-name associated with identifier-1 and the item identifier-2 or index-name-1 is incremented by the SEARCH.

In Option 1, condition-1, condition-2, etc., may be any condition described in 4.2.2 of "Fundamentals of COBOL – Language".

In Option 2, condition-1 may consist of a relation condition incorporating EQUALS or EQUAL TO, or a condition-name condition (VALUE clause that describes the condition-name must contain only a single literal). Condition-1 may also be a compound condition composed of the types just mentioned, with AND as the only connective. Any data-name that appears in the KEY option of identifier may be the subject or object of a test or it may be the name of a conditional variable with which the tested condition-name is associated. All preceding data-names in the KEY option must also be tested within condition-1. No other tests can appear within condition-1.

Example:

Assume that there exists a table of wholesale discount factors used in determining the total price of merchandise orders, as follows:

```
01 DISCOUNT–TABLE
02 RANGE–ENTRY, OCCURS 16 TIMES INDEXED BY XRANGE
    03 MAXRANGE PICTURE IS 9(8)
    03 CLASS–ITEM OCCURS 12 TIMES INDEXED BY XCLASS
```

Each wholesale order input record is computed against a basic price schedule catalogue. An item in the wholesale order record, TOTAL–PRICE, is then compared against the discount schedule to determine which set of discount rates is to be applied to the particular order. Each occurrence of RANGE–ENTRY contains a field called MAXRANGE and 12 occurrences of CLASS–ITEM. MAXRANGE specifies the maximum order amount for which the accompanying set of CLASS–ITEM discount factors can be applied. Each CLASS–ITEM contains a discount factor for a particular class of merchandise.

The statement:

```
SEARCH RANGE–ENTRY VARYING XRANGE AT END GO TO ERROR WHEN
MAXRANGE IS GREATER THAN TOTAL–PRICE GO TO RANGE–LOCATED
```

When the search is completed, XRANGE will be set at that occurrence of RANGE-ENTRY which contains the appropriate set of discount factors.

3.3.3. SET

Format:

Option 1:

```
                                                                 ⎧ index-name-3 ⎫
SET ⎧ index-name-1 ⎫ ⎡ , ⎧ index-name-2 ⎫ ⎤ . . . TO ⎨ identifier-3 ⎬
    ⎩ identifier-1 ⎭ ⎣   ⎩ identifier-2 ⎭ ⎦           ⎩ literal      ⎭
```

Option 2:

```
SET index-name-1 [, index-name-2] . . . ⎧ UP BY   ⎫ ⎧ identifier ⎫
                                        ⎩ DOWN BY ⎭ ⎩ literal    ⎭
```

Description:

The SET statement establishes reference points for table handling operations by setting index-names to values with respect to table-elements.

Each object of a SET statement must be one of the following:

(a) an index-name,

(b) an index data item described by a USAGE IS INDEX clause, or

(c) an elementary data item of integer value. However, in Option 2, identifier may not be an index data item.

In Option 1, index-name-1 (and index-name-2, etc.) is set to a value which corresponds to the same occurrence number as that to which either index-name-3, identifier-3, or literal corresponds. No conversion takes place if identifier-3 is an index data item, or if index-name-3 is related to the same table as index-name-1.

Identifier-1 (and identifier-2, etc.), if specified, and if an index data item, may be set equal to either the contents of index-name-3 or identifier (if identifier-3 is also an index data item). Literal cannot be applied in this case.

If identifier-1 is not an index data item, it may be set only to an occurrence number corresponding to the value of index-name-3. Neither identifier-3 nor literal can be applied in this case.

In Option 2, the contents of index-name-1 (and index-name-2, etc.) are incremented (UP BY) or decremented (DOWN BY) by a value corresponding to the number of occurrences represented by the value of literal or identifier.

Examples:

(a) SET IX-TABLE TO 3.

The index-name IX-TABLE is set to an index value which corresponds to occurrence number 3 for that table. If IX-TABLE is not defined via an INDEXED BY clause this statement is illegal.

(b) SET LOOKUP TO KEY–POINT

If LOOKUP is an index-name, it is set to the occurrence number which corresponds to KEY–POINT. If KEY–POINT is an index-name that is not related to the same table as LOOKUP, an appropriate conversion is performed; otherwise, no conversion takes place. If LOOKUP is an index data item, (i.e., defined by a USAGE IS INDEX clause) it is set to the actual contents of KEY–POINT. KEY–POINT must be either an index data item or an index-name, or this statement is illegal.

If LOOKUP is neither an index-name nor an index data item, KEY–POINT must be an index-name. LOOKUP is then set to the occurrence value to which index-name KEY–POINT corresponds.

(c) SET DEPT–INDEX UP BY KEY–JUMP

The index-name DEPT–INDEX is incremented by a value that corresponds to the number of occurrences indicated by KEY–JUMP; i.e., if the value of KEY–JUMP is 3, DEPT–INDEX is incremented by a value that is equivalent to three occurrences.

3.3.4. Comparisons Involving Index-Names and/or Index Data Items

The comparison of two index-names is the same as if the corresponding occurrence numbers were compared. Similarly, when an index-name is compared with a data item (other than an index data item) or literal, it is the same as if their corresponding occurrence numbers were compared.

When a comparison between an index-name and an index data item is made, the actual values are compared without conversion. Any other comparison involving an index data item is invalid and will cause unpredictable results.

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

____

(Document Title)

____________	____________	____________
(Document No.)	*(Revision No.)*	*(Update No.)*

Comments:

Cut along line.

From:

____

(Name of User)

____

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

CUT