IBM System/36

Programming with COBOL

When You Are:	You Can Find Information In:
Planning to Install Your Computer	What to Do Before Your Computer Arrives or Converting from System/34 to System/36
Getting Your Computer Ready to Use	Setting Up Your Computer Performing the First System Configuration For Your System System Security Guide
Operating Your Computer	Learning About Your Computer Operating Your Computer
Operating and Using the Utilities	Source Entry Utility Guide Data File Utility Guide Creating Displays Work Station Utility Guide Utilities Messages
Programming Your Computer	Concepts and Programmer's Guide System Reference Sort Guide Work Station Utility Guide **Programming with COBOL** **COBOL Summary** **COBOL Messages** Getting Started with the Interactive Definition Utility Guide Distributed Data Management Guide (communication manuals) (communication message manuals)
Communicating with Another Computer or Remote Device	System Messages (message manuals) System Problem Determination
Determining the Cause of a Problem	

Chapter 10. Data Division

Data Division Concepts

The Data Division of a COBOL source program describes all the data to be processed by the object program. Two types of data can be processed:

- External data
- Internal data.

External Data

External data is contained in files. A file is a collection of data records existing on an input/output device, such as a disk. A file can be a group of physical records or a group of logical records. The Data Division source statements describe the relationship between physical and logical records.

A physical record is a unit of data that is treated as a single object when it is moved into or out of auxiliary storage. The size of a physical record is determined by the particular input/output device on which you store it. The size does not necessarily have a direct relationship to the size or content of the logical information contained in the file.

A logical record is a unit of data with subdivisions that are logically related. A logical record can be a physical record (that is, contained completely in one physical unit of data), several logical records can be contained within one physical record, or one logical record can extend across several physical records.

Record description entries, which you place after the FD (file description) entry for a specific file, describe the logical records in the file. These entries also describe the category and the format of data within each field of the logical record and different values the data might be assigned.

The FD entry gives the physical aspects of the data such as the:

- Size relationship between physical and logical records
- Size and name(s) of the logical record(s)
- Labeling information.

Once the relationship between physical and logical records has been established, only logical records are made available to the COBOL program. Thus, in this manual, a reference to records means logical records unless the term physical records is used.

Internal Data

Program logic might develop additional data within storage. Such data is called internal data.

The concept of logical records applies to internal data as well as to external data. Internal data can thus be grouped into logical records that you can define with a series of record description entries. You can define items that need not be so grouped in independent data description entries.

Data Relationships

In the Data Division, you define the relationships of all data you want to use in a program through a system of level indicators and level numbers.

- A level indicator, together with its descriptive entry, identifies each file description in a program. Level indicators are the highest level of any data hierarchy with which they are associated.

- A level number, together with its descriptive entry, indicates the properties of specific data. You can use level numbers to describe a data hierarchy. These level numbers can:

  - Indicate that this data has a special purpose
  - Be associated with, and be subordinate to, level indicators
  - Be used independently to describe internal data or data common to two or more programs.

Data Division Organization

The Data Division is divided into three sections:

- The File Section

- The Working-Storage Section

- The Linkage Section.

Each section has a specific logical function within a COBOL source program. You can leave out a section from the source program when you do not need its logical function.

Format

```
DATA DIVISION.
[FILE SECTION.

     [file-description-entry or sort-merge-file-description-entry] . . .

     {record-description-entry} . . .]

[WORKING-STORAGE SECTION.

     [data-item-description-entry] . . .

     [record-description-entry] . . .]

[LINKAGE SECTION.

     [data-item-description-entry] . . .

     [record-description-entry] . . .]
```

In the source program, you must place the Data Division sections in the order shown.

File Section

The File Section contains a description of all externally stored data (FD entries) and a description of each sort-merge file (SD entries) used in the program.

You must begin the File Section with the header FILE SECTION followed by a period and a space. The File Section contains file description entries and sort-merge file description entries. Each entry is followed by its associated record description entry (or entries).

In a COBOL program, the file description entries (beginning with the level indicators FD and SD) represent the highest level of organization in the File Section. The file description entry provides information about the physical structure and identification of a file, and gives the record name(s) associated with that file. For a further description of the format and the clauses required in a file description entry, see *File Description Entry* later in this chapter. For a complete discussion of the sort-merge file description entry, see *Data Division Sort/Merge* in Chapter 13.

The record description entry consists of a set of data description entries that describe the records contained within a particular file. You can use more than one record description entry; each is an alternative description of the same storage area. For the format and the clauses required within the record description entry, see *Data Description* later in this chapter.

Data areas that you describe in the File Section are not available for processing unless you open the file containing the data area.

Working-Storage Section

In the Working-Storage Section, you can include description records that are not part of data files, but are developed and processed internally. These records are used for report description, counters, and other functions necessary in processing data.

You must begin the Working-Storage Section with the section header WORKING-STORAGE SECTION followed by a period and a space. The Working-Storage Section contains record description entries and data description entries for noncontiguous data items.

You must group data elements in the Working-Storage Section that bear a definite hierarchical relationship to one another into records structured by level number.

You need not group noncontiguous items in this section that bear no hierarchical relationship to one another into records if they do not need to be subdivided further. Instead, they are classified and defined as noncontiguous elementary items. Define each in a separate data description entry that begins with the special level number 77. The format of the data description entry is the same as the format for the record description entry.

Linkage Section

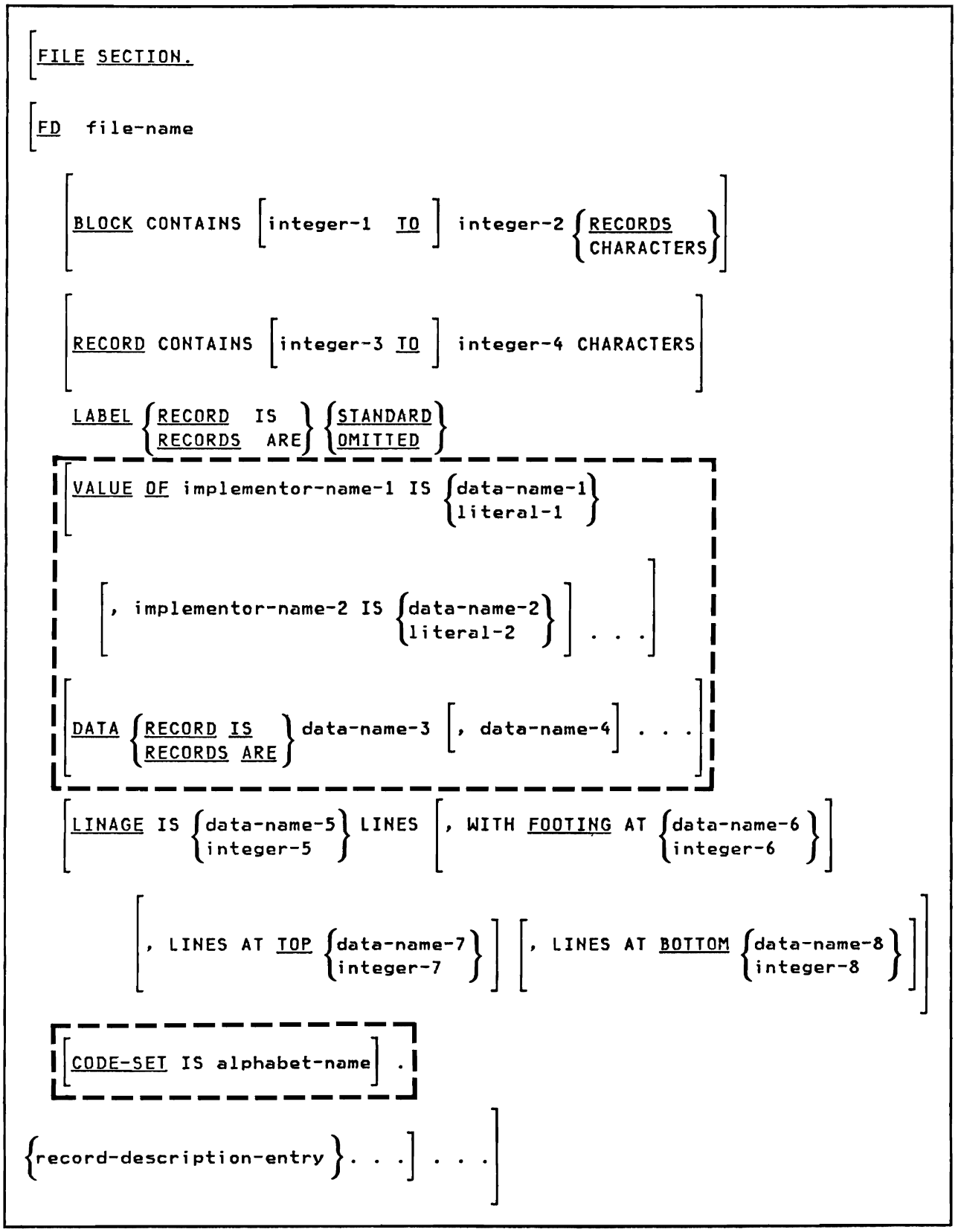
The Linkage Section describes data made available from another program.

Record description entries and data description entries in the Linkage Section provide names and descriptions, but storage within the program is not reserved because the data area exists elsewhere. You can use any data description clause to describe items in the Linkage Section, with one exception: You cannot use the VALUE clause for any items other than level-88 items. For additional information, see *Data Division Subprogram Linkage* in Chapter 13.

File Description Entry

In a COBOL program, the FD (file description) entry or the SD (sort-merge file description) entry is the highest level of organization in the File Section.

Format 1-Sequential, Indexed, Relative Files



Format 2-TRANSACTION File

```
FD  file-name

[ RECORD CONTAINS [integer-3  TO ] integer-4 CHARACTERS ]

 LABEL {RECORDS  ARE} {STANDARD}
       {RECORD   IS } {OMITTED }

[ DATA {RECORD  IS } data-name-3 [, data-name-4] . . . ]
       {RECORDS ARE}

record-description-entry  . . .
```

Coding Examples

The following coding example shows the clauses you will probably use most for a format 1 file description entry.

```
SEQUENCE  CONT  A   B
(PAGE)(SERIAL)
1  3  4  6  7  8  12  16  20  24  28
003 010  DATA DIVISION.
    020  FILE SECTION.
    030  FD   FILE-NAME
    040       RECORD
    050       LABEL RECORD
    060       DATA RECORD IS
    070  01   DESCRIPTION
    080
    090
    100
    110  WORKING-STORAGE SECTION.
    120  77   NAME-DESCRIPTION.
    130  01   RECORD-DESCRIPTION.
```

The following example shows the Data Division in a program:

```
DATA DIVISION.
FILE SECTION.
FD  INPUT-DATA
    BLOCK CONTAINS 1 RECORDS
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORDS ARE GEN-INFO SALES-DATA.
01  GEN-INFO.
    03 EMPLOYEE-NAME.
          05 FIRST-NAME                        PIC X(12).
          05 LAST-NAME                         PIC X(12).
    03 SOC-SEC-NUMBER                          PIC 9(9).
    03 CHECK-SSN  REDEFINES SOC-SEC-NUMBER     PIC X(9).
    03 AGE                                     PIC 99.
    03 BIRTH-DATE.
          05 B-MONTH                           PIC 99.
          05 B-DAY                             PIC 99.
          05 B-YEAR                            PIC 99.
    03 ANNUAL-SALARY                           PIC 9(5)V99.
    03 CHECK-SALARY REDEFINES ANNUAL-SALARY    PIC X(7).
*   THIS REDEFINES WILL BE USED TO SEE IF THE FIELD IS BLANK.
    03 RECORD-ID                               PIC X.
    03 FILLER                                  PIC X(31).
01  SALES-DATA.
    03 SALES-SSN                               PIC 9(9).
    03 SALES-LOCATION                          PIC XX.
       88 MICHIGAN VALUE IS 'MI'.
       88 EASTERN-REGION VALUES ARE 'PA' 'NY'.
       88 HEADQUARTERS VALUES ARE 'BA' THRU 'BZ'.
    03 TOTAL-COMMISSION                        PIC 9(5)V99.
    03 RECORD-CODE                             PIC X.
    03 FILLER                                  PIC X(61).
FD  REPORT-OUT
    LABEL RECORDS ARE OMITTED
    RECORD CONTAINS 132 CHARACTERS
    LINAGE IS 60 LINES
    WITH FOOTING 59
    LINES AT TOP 3
    LINES AT BOTTOM 3
    DATA RECORD IS PRINT-OUT.
01  PRINT-OUT                                  PIC X(132).

WORKING-STORAGE SECTION.
77  RECORDS-IN                                 PIC 9(6) VALUE ZEROS.
77  DECLARATIVE-ERRORS                         PIC 9(4) VALUE ZEROS.
77  EOF-SW                                     PIC X    VALUE ZERO.
77  BAD-DATA-COUNTER                           PIC 9(3) VALUE ZERO.
77  CHECK-IT                                   PIC XX.
01  PRINT-FIELDS-EDITED.
       03 FILLER                               PIC X(14) VALUE SPACES.
       03 TOTAL-SALARY                         PIC $$$,$$$.$9BB.
       03 COMMISSION-COSTS                     PIC $**,***,***.99B.
       03 FILLER                               PIC X(65) VALUE ALL '-'.
       03 FILLER                               PIC X(12)
                                               VALUE '...END...JOB'.
01  SALARY-COUNTER                             PIC 9(6)V99 VALUE ZEROS.
01  COMMISSION-COUNTER                         PIC 9(6)V99 VALUE ZEROS.
```

You must begin the file description entry with the level indicator FD followed by a space.

The clauses that follow the file name are optional in many cases and can be in any order; however, you must follow the FD entry with at least one record description entry. When you use more than one record description entry, each entry implies a redefinition of the same storage area. You must immediately follow the last clause in the FD entry with a period and a space.

IBM Extension

Format 2-TRANSACTION File Considerations

A file description entry consists of a:

- Level indicator (FD)
- File name
- Series of independent clauses.

For a TRANSACTION file, the independent clauses you can use are the:

- RECORD CONTAINS clause
- LABEL RECORDS clause
- DATA RECORDS clause.

Only the LABEL RECORDS clause is required.

The LABEL RECORDS clause specifies whether or not labels are present. This clause is treated as comments in a TRANSACTION file. You must include the LABEL RECORDS clause in every file description entry.

The RECORD CONTAINS clause and the DATA RECORDS clause are described under ***RECORD CONTAINS Clause*** and ***DATA RECORDS Clause*** later in this chapter. You must have a record definition large enough to hold the largest record defined by the display formats or SSP-ICF records processed by the program.

End of IBM Extension

File Name

You must place the file name after the level indicator, and you must use the same file name as the one you used in the SELECT clause of the associated file control entry. (See *FILE-CONTROL Paragraph* in Chapter 3.)

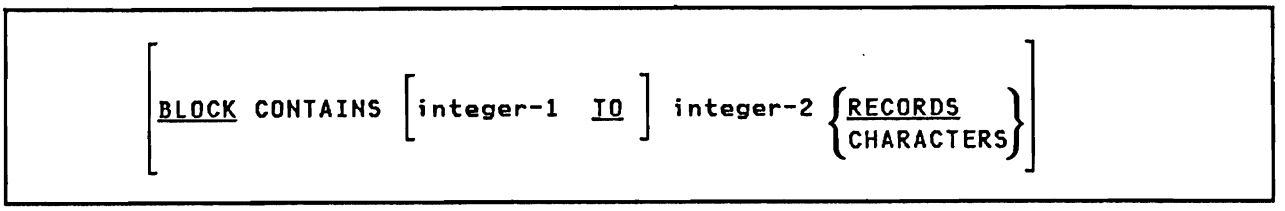
The file name must follow the rules of formation for a user-defined word; you must include at least one alphabetic character. You must make the file name unique within this program.

BLOCK CONTAINS Clause

The BLOCK CONTAINS clause gives the size of a physical record. The BLOCK CONTAINS clause is used by the compiler to determine the blocking factor for a disk file. The BLOCK CONTAINS clause has no effect on the physical formatting of the file as it resides on disk.

When the BLOCK CONTAINS clause is omitted, the compiler assumes that records are not blocked; thus, this clause can be omitted when each physical record contains only one complete logical record.

Format



You must make integer-1 and integer-2 unsigned, nonzero integers.

When you use neither the RECORDS phrase nor the CHARACTERS phrase, the CHARACTERS phrase is assumed.

RECORDS Phrase: When you use the RECORDS phrase, the physical record size is expressed as the number of logical records contained in each physical record.

The compiler assumes that the block size must provide for integer-2 records of maximum size, and provides any additional space needed for control bytes.

Note: Maximum record size is 4096; maximum block size is 9999.

CHARACTERS Phrase: When the CHARACTERS phrase is specified or implied, the physical record size is given as the number of character positions required to store the physical record no matter what USAGE the characters within the data record have.

If you use only integer-2, the compiler converts the value to a number of records that are to be blocked together. When you use both integer-1 and integer-2, they represent, respectively, the minimum and maximum character size of the physical record, rounded up to the nearest whole record.

The compiler assumes that the block size must provide for integer-2 characters, converted into a number of records, even when you use integer-1.

RECORD CONTAINS Clause

The RECORD CONTAINS clause specifies the size of a file's data records.

Format

```
[RECORD CONTAINS [integer-3 TO ] integer-4 CHARACTERS]
```

The RECORD CONTAINS clause is never required, because you completely define the size of each record in the record description entries. When you use this clause, the following rules apply:

- You must make integer-3 and integer-4 unsigned, nonzero integers.
- When you use both integer-3 and integer-4, integer-3 gives the size of the smallest data record, and integer-4 gives the size of the largest data record.
- You must not use integer-4 alone unless all the records are the same size. If all records are the same size, integer-4 gives the exact number of characters in the record.
- You must give the record size as the number of character positions needed to store the record internally; that is, size is given in terms of the bytes occupied internally by the record's characters, regardless of the number of characters used to represent the item within the record. The size of a record is determined according to the rules for obtaining the size of a group item. For a further description of record size, see the *USAGE Clause* later in this chapter.

  Note: When you leave out the RECORD CONTAINS clause, the record lengths are determined by the compiler from the record descriptions. When you have an entry within a record description that contains an OCCURS DEPENDING ON clause, the compiler uses the maximum value of the variable-length item to calculate the record length.

LABEL RECORDS Clause

The LABEL RECORDS clause specifies whether labels are present or left out. The LABEL RECORDS clause is required in every FD entry.

Format

```
LABEL {RECORD  IS } {STANDARD}
      {RECORDS ARE} {OMITTED  }
```

STANDARD Phrase: The STANDARD phrase specifies that this file has labels conforming to system specifications. You must use this phrase for disk files.

OMITTED Phrase: The OMITTED phrase specifies that no labels exist for this file. You must use this phrase for files assigned to unit record devices.

VALUE OF Clause

The VALUE OF clause serves only as documentation. It specifies the description of an item in the label records associated with this file.

Format

```
VALUE OF implementor-name-1 IS { data-name-1 }
                               { literal-1   }

    [, implementor-name-2 IS { data-name-2 }] . . .
                             { literal-2   }
```

DATA RECORDS Clause

The DATA RECORDS clause serves only as documentation for the names of data records associated with this file. The DATA RECORDS clause is never required.

Format

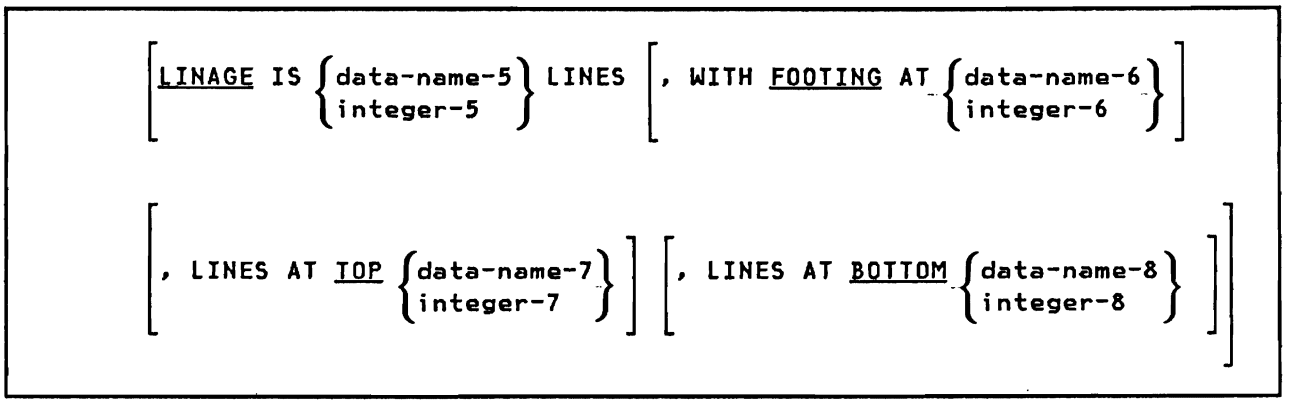
```
DATA { RECORD  IS } data-name-3 [, data-name-4] . . .
     { RECORDS ARE }
```

More than one data name indicates that this file contains more than one type of data record. Two or more record descriptions for this file occupy the same storage area. These records need not have the same description or length. The order in which you list the data names is not important.

LINAGE Clause

The LINAGE clause may be used only for printer files. It gives the depth of a logical page in terms of the number of lines. This clause also optionally gives the line number at which the footing area begins, as well as the top and bottom margins of the logical page. There is not necessarily a relationship between the logical page size and the physical page size.

Format



In the LINAGE clause, you must describe all data names and integers as unsigned, integer data items.

LINAGE Integer-5/Data-Name-5: Integer-5 or the value in data-name-5 gives the number of lines that can be written or spaced or both on this logical page. The area of the page that these lines represent is called the page body. You must use a value that is greater than 0.

This page is intentionally left blank.

WITH FOOTING Phrase: Integer-6 or the value in data-name-6 gives the first line number of the footing area within the page body. You must have a footing line number that is greater than 0, but it must not be greater than the number for the last line of the page body. The footing area extends between those two lines. If you do not use this phrase, the assumed value is equal to that of the page body (integer-5 or data-name-5).

LINES AT TOP Phrase: Integer-7 or the value in data-name-7 gives the number of lines in the top margin of the logical page. If you do not use this phrase, 0 is assumed.

LINES AT BOTTOM Phrase: Integer-8 or the value in data-name-8 gives the number of lines in the bottom margin of the logical page. If you do not use this phrase, 0 is assumed.

Figure 10-1 shows you how to use each phrase of the LINAGE clause.

LINAGE Clause Considerations: The logical page size that you give in the LINAGE clause is the sum of all values you gave in each phrase except the FOOTING phrase. If the LINES AT TOP and the LINES AT BOTTOM phrases are 0, each logical page immediately follows the preceding logical page with no additional spacing provided.

At the time an OPEN OUTPUT statement is performed, the values of integer-5, integer-6, integer-7, and integer-8 are used to determine the page body, first footing line, top margin, and bottom margin of the logical page for this file. These values are then used for all logical pages printed for this file during a given run of the program.

Data-name-5, data-name-6, data-name-7, and data-name-8 have the following effects on the logical page:

- Their values at the time an OPEN OUTPUT statement is performed determine the following for the first logical page only:
  - Page body
  - First footing line
  - Top margin
  - Bottom margin.
- Their values at the time a WRITE ADVANCING statement causes page ejection determine the following for the succeeding logical page only:
  - Page body
  - First footing line
  - Top margin
  - Bottom margin.

LINAGE-COUNTER Special Register: For each FD entry in which you use a LINAGE clause, a separate LINAGE-COUNTER special register is generated. LINAGE-COUNTER is initialized to 1 when an OPEN statement for this file is performed. LINAGE-COUNTER is automatically modified by any WRITE statement you use for this file.

When you refer to more than one LINAGE-COUNTER special register in the PROCEDURE DIVISION, you must qualify each LINAGE-COUNTER with its related file name. For example, LINAGE-COUNTER OF FILE-A.

The value in LINAGE-COUNTER at any given time is the line number at which the device is positioned within the current page. You can refer to LINAGE-COUNTER in Procedure Division statements; however, you must not change LINAGE-COUNTER with these statements.

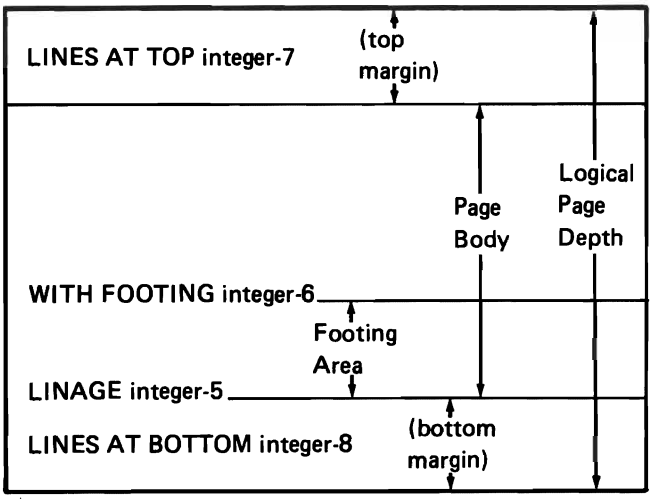
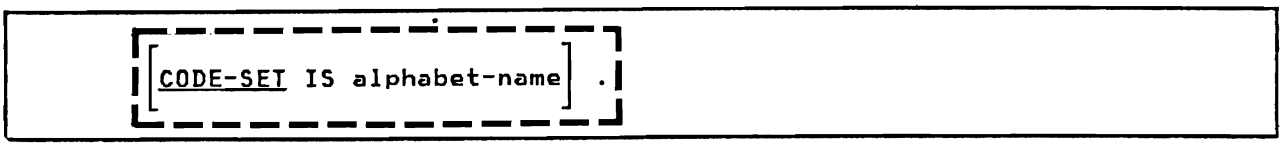


Figure 10-1. LINAGE Clause and Logical Page Depth

CODE-SET Clause

The CODE-SET clause is not required or used by the System/36 COBOL compiler. If you include it in the source program, the compiler treats this clause as a comment.

Format



DATA DESCRIPTION

All data you use in a COBOL program is described using a uniform system of representation. The basic concepts of data description are discussed in this chapter, as well as the actual COBOL clauses you use to describe data.

Data Description Concepts

You need to present most of the data processed by a COBOL program in hierarchically arranged records. This is necessary because you must subdivide most data for processing. To subdivide such records, COBOL uses a hierarchical concept of levels.

For example, in a department store's customer file, one complete record could contain all data about one customer. Subdivisions within that record could be customer name, customer address, account number, department number of sale, unit amount of sale, dollar amount of sale, previous balance, and other information.

This page left intentionally blank

Level Concepts

Because you must divide records into logical subdivisions, the concept of levels is part of the structure of a record. Once you have subdivided a record, you can further subdivide it to provide more detailed data references.

The basic subdivisions of a record (that is, those fields that you do not subdivide further) are called elementary items. A record can be made up of a series of elementary items or it may itself be an elementary item.

Because you might need to refer to a set of elementary items, you can combine elementary items into group items. You can also combine groups into a more inclusive group that contains two or more subgroups. Thus, within one hierarchy of data items, an elementary item can belong to more than one group item.

Level Numbers

You use a system of level numbers to organize elementary and group items into records. You also use special level numbers to identify data items you want to use for special purposes.

You need a separate entry for each group and elementary item in a record, and you must assign each a level number. Use the following level numbers to structure records:

01 This level number specifies the record itself and is the most-inclusive level number you can use. You can make a level-01 entry either a group item or an elementary item.

02-49 These level numbers specify group and elementary items within a record. Assign less-inclusive data items higher (not necessarily consecutive) level numbers.

A group item includes all group and elementary items following it until a level number less than or equal to the level number of this group is encountered.

You must give all elementary or group items immediately subordinate to one group item identical level numbers that are higher than the level number of this group item.

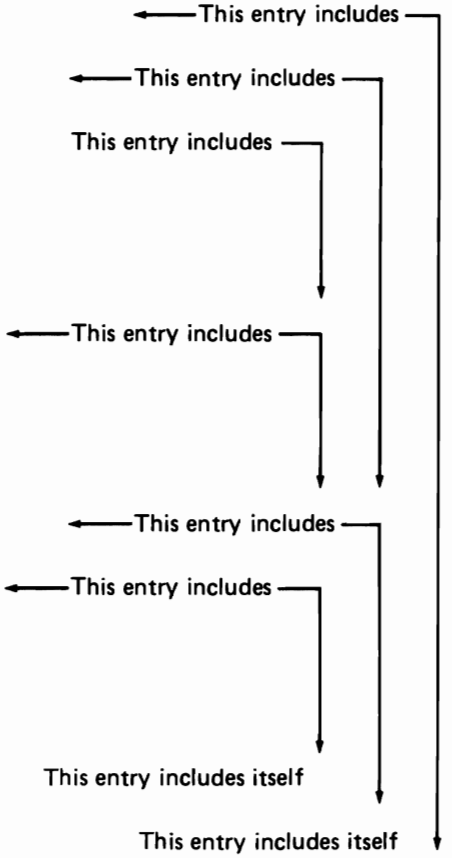
Figure 10-2 illustrates the concept of level numbers. Note that all groups immediately subordinate to the level-01 entry have the same level number. Note also that elementary items from different subgroups do not necessarily have the same level number and that elementary items can be used at any level within the hierarchy. Figure 10-2 shows the COBOL record description entry in the left portion of the figure: it shows the subdivision of the entry in the right portion of the figure.

Note: You can also write level numbers 01 through 09 as 1 through 9.

The COBOL record description entry is written as follows:

```
01   RECORD-ENTRY.
     05   GROUP-1.
          10   SUBGROUP-1.
               15   ELEM-1   PIC . . . .
               15   ELEM-2   PIC . . . .
          10   SUBGROUP-2.
               15   ELEM-3   PIC . . . .
               15   ELEM-4   PIC . . . .
     05   GROUP-2.
          15   SUBGROUP-3.
               25   ELEM-5   PIC . . . .
               25   ELEM-6   PIC . . . .
          15   SUBGROUP-4   PIC . . . .
     05   GROUP-3 PIC . . . .
```

The items included in the hierarchy of each level are indicated below:



The storage arrangement is illustrated below:

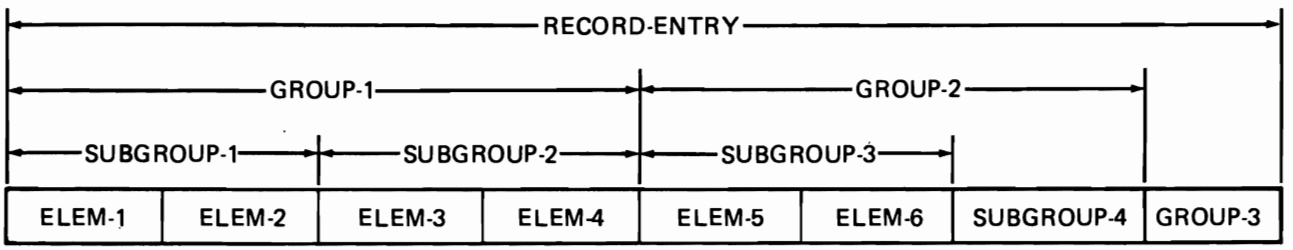


Figure 10-2. How the Record Description Entry Is Stored

Note: A PICTURE clause is required for every elementary item except an indexed data item. This clause is discussed under ***PICTURE Clause*** later in this chapter.

Special Level Numbers

Use special level numbers to identify items that do not structure a record. The following are special level numbers:

66 Use this level number to identify elementary or group items that you described with a RENAMES clause. Such items regroup previously defined data items.

77 Use this level number to identify independent data description entries in the Working-Storage or Linkage Section. These items are not subdivisions of other items and are not themselves subdivided.

88 Use this level number to identify any condition-name entry that is associated with a particular value of a conditional variable. An example is given under ***VALUE*** Clause later in this chapter.

Note: You must give unique data names to level-77 and level-01 entries in the Working-Storage Section and Linkage Section because you cannot qualify either entry. If you can qualify subordinate data names, you need not make them unique.

Indentation

You can begin successive data description entries in the same column as preceding entries, or you can indent them according to level number. Indentation is useful for documentation, but it does not affect the action of the compiler.

Classes of Data

You can divide all data used in a COBOL program into four classes and six categories. Every elementary item in a program belongs to one of the classes as well as to one of the categories. Every group item belongs to the alphanumeric class even if the subordinate elementary items belong to another class and category. Figure 10-3 shows the relationship of data classes and categories.

Level of Item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric edited Alphanumeric edited Alphanumeric
	Boolean	Boolean

Figure 10-3 (Part 1 of 2). Classes and Categories of Data

Level of Item	Class	Category
Group	Alphanumeric	Alphabetic Numeric Numeric edited Alphanumeric edited Alphanumeric Boolean

Figure 10-3 (Part 2 of 2). Classes and Categories of Data

IBM Extension

Boolean Data Facilities

Boolean data provides a means of modifying and passing the values of the indicators associated with the display formats. A Boolean value of 0 is the indicator's OFF status; a Boolean value of 1 is the indicator's ON status.

A Boolean literal contains a single 0 or 1 and is enclosed in quotes and immediately preceded by an identifying B. The Boolean literal is defined as either B'0' or B'1'. A Boolean character occupies 1 byte. You can use the figurative constant ZERO as a Boolean literal, and the reserved word ALL with a Boolean literal.

End of IBM Extension

Standard Alignment Rules

The standard alignment rules for positioning data in an elementary item depend on the data category of the receiving item (that is, the item into which you place the data).

Numeric Items: When a numeric item is the receiving item, the following rules apply:

- The data is aligned on the assumed decimal point and, if necessary, truncated or padded with 0's. (An assumed decimal point is one that has logical meaning but does not exist as a character in the data.)
- If a decimal point is not explicitly specified, the receiving item is treated as though an assumed decimal point is specified immediately to the right of the field. The data is then treated as in the preceding rule.

Numeric Edited Items: The data is aligned on the decimal point and, if necessary, truncated or padded with 0's at either end, except when editing causes replacement of leading 0's.

Alphanumeric, Alphanumeric Edited, Alphabetic: For these data categories, the following rules apply:

- The data is aligned at the leftmost character position and, if necessary, truncated or padded with spaces at the right.
- If you use the JUSTIFIED clause for alphanumeric or alphabetic receiving items, the above rule is modified as described in the JUSTIFIED clause. (See ***JUSTIFIED Clause*** later in this chapter.)

  Note: The JUSTIFIED clause must not be specified for any item for which editing is specified.

Standard Data Format

COBOL makes data description as machine independent as possible. For this reason, you describe the properties of the data in a standard data format rather than a machine-oriented format.

The standard data format uses the decimal system to represent numbers no matter what base is used by the system. You can include any characters in the nonnumeric data that are in the native character set. That is, nonnumeric data is not limited to just the COBOL character set or the nonnumeric COBOL characters.

Character String and Item Size

In COBOL, the size of an elementary item is determined through the number of character positions you used in its PICTURE character string. In storage, however, the size is determined by the actual number of bytes the item occupies as determined by the combination of its PICTURE character string and its USAGE clause.

Normally, when an arithmetic item is moved from a longer field to a shorter one, the compiler truncates the data to the number of characters represented in the shorter item's PICTURE character string.

For example, if you move a sending field with PICTURE S99999 and the value +12345 to a COMPUTATIONAL receiving field with PICTURE S99, the data is truncated to +45. (See *PICTURE Clause* later in this chapter.)

Signed Data

There are two categories of algebraic signs used in COBOL:

- Operational signs
- Editing signs.

Operational Signs

Operational signs (+ -) are associated with signed numeric items and indicate their algebraic properties. The internal representation of an algebraic sign depends on the item's USAGE clause and optionally upon its SIGN clause. Zero is considered a unique value regardless of the operational sign. An unsigned field is always assumed to be positive or 0.

Editing Signs

Editing signs are associated with numeric edited items. Editing signs are PICTURE symbols (+ - CR DB) that identify the sign of the item in edited output.

DATA DESCRIPTION ENTRY

A record description entry or a data description entry gives the characteristics of a particular data item. The maximum length for any item that is not otherwise restricted is 32,767 bytes. The four general formats are:

Format 1

```
level-number {data-name} clause
             {FILLER   }

      [REDEFINES clause]

      [USAGE clause]

      [SIGN clause]

      [OCCURS clause]

      [SYNCHRONIZED clause]

      [JUSTIFIED clause]

      [BLANK WHEN ZERO clause]

      [VALUE clause]

      [PICTURE clause].
```

Format 2-RENAMES Clause

```
66 data-name-1 RENAMES data-name-2 [{THROUGH} data-name-3].
                                   {THRU   }
```

Format 3

```
88 condition-name { VALUE IS   } literal-1 [ { THROUGH } literal-2 ]
                  { VALUES ARE }           [ { THRU    }           ]

[ literal-3 [ { THROUGH } literal-4 ] ]... .
            [ { THRU    }           ]
```

Format 4-Boolean Data

```
level-number { data-name } clause
             { FILLER    }
[REDEFINES clause]
[PICTURE clause]
[USAGE clause]
[OCCURS clause]
[SYNCHRONIZED clause]
[JUSTIFIED clause]
[VALUE clause]
[INDICATOR clause].
```

Format 1

Use this format for record description entries (except for Boolean data) in all sections and for level-77 entries in the Working-Storage and Linkage Sections. The following rules apply:

- You can make the level number any number from 01 through 49, or 77.
- You can write the clauses in any order, with two exceptions:
  - You must immediately follow the level number with the data-name/FILLER clause.
  - When you use the REDEFINES clause, you must place it immediately after the data-name/FILLER clause.
- You must use the PICTURE clause for every elementary item except index data items.
- You can use the BLANK WHEN ZERO, JUSTIFIED, PICTURE, and SYNCHRONIZED clauses only for elementary items.
- You must separate clauses either with a space or with a comma or a semicolon followed by a space.
- You must end each record description entry with a period followed by a space.

Format 2-RENAMES Clause

The RENAMES clause gives alternative, possibly overlapping, groupings of elementary data items. This clause lets a single data name rename a group of data items within a record.

You can write one or more RENAMES entries for a logical record. You must place all RENAMES entries associated with one logical record immediately after that record's last data description entry. You cannot use a level-66 entry to rename a level-01, a level-77, a level-88, or another level-66 entry, or another data name that contains an INDICATOR clause.

Note: You can use the RENAMES clause to rename an INDICATOR data item; however, the new data name does not have an INDICATOR value associated with it, and you cannot use it as an indicator.

Data-name-1 identifies an alternative grouping of data items. You cannot use data-name-1 as a qualifier. You can qualify data-name-1 only with the names of level indicator entries or level-01 entries.

Note: Level number 66 and data-name-1 are not part of the RENAMES clause itself and are included in the format only for clarity.

This page is intentionally left blank.

Data-name-2 or data-name-3 identifies the original grouping of elementary data items; that is, you must have them name elementary or group items within the associated level-01 entry, and you must not give them the same data name. You can qualify both data names.

You must not use the OCCURS clause in the data entries for data-name-2 and data-name-3, or for any group entry to which these data entries are subordinate. In addition, you must not use the OCCURS DEPENDING ON clause for any item occupying storage between data-name-2 and data-name-3.

Data-Name-2 Phrase: When you do not use data-name-3, you can make data-name-2 either a group item or an elementary item. When you make data-name-2 a group item, data-name-1 is treated as a group item. When you make data-name-2 an elementary item, data-name-1 is treated as an elementary item.

Data-Name-2 THRU Data-Name-3 Phrase: When you use data-name-3, data-name-1 is a group item that includes all elementary items:

- Starting with data-name-2 (if it is an elementary item) or the first elementary item within data-name-2 (if it is a group item)
- Ending with data-name-3 (if it is an elementary item) or the last elementary item within data-name-3 (if it is a group item).

The key words THRU and THROUGH are equivalent.

You must not have the leftmost character in data-name-3 precede that in data-name-2; you must have the rightmost character in data-name-3 follow that in data-name-2. This means that you make data-name-3 subordinate to data-name-2.

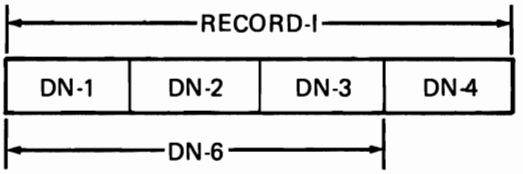
Valid and invalid uses of the RENAMES clause are given in Figure 10-4.

Note: The THRU option may not be used if the elementary items being renamed include an item that has a packed decimal representation (USAGE IS COMP-3). The RENAMES clause and the THRU option are accepted by the compiler, but unexpected results may occur.

COBOL Specifications | **Storage Layouts**

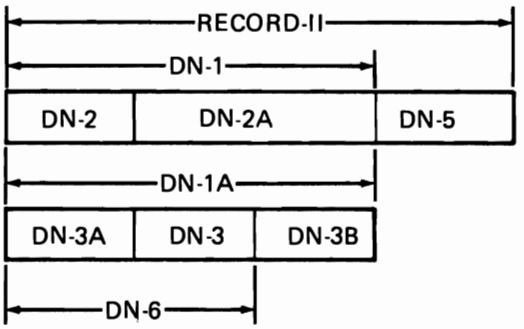
Example 1 (Valid)

```
01   RECORD-I.
     05  DN-1 . . .  .
     05  DN-2 . . .  .
     05  DN-3 . . .  .
     05  DN-4 . . .  .
66   DN-6 RENAMES DN-1 THROUGH DN-3.
```



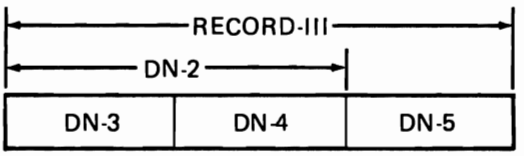
Example 2 (Valid)

```
01   RECORD-II.
     05  DN-1.
         10  DN-2 . . .  .
         10  DN-2A . . .  .
     05  DN-1A REDEFINES DN-1.
         10  DN-3A . . .  .
         10  DN-3 . . .  .
         10  DN-3B . . .  .
     05  DN-5 . . .  .
66   DN-6 RENAMES DN-2 THROUGH DN-3.
```



Example 3 (Invalid)

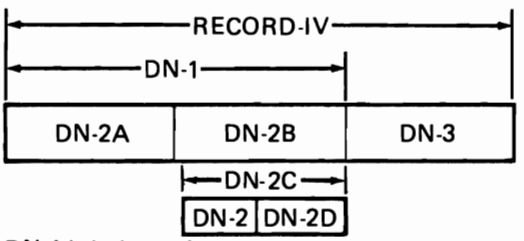
```
01   RECORD-III.
     05  DN-2.
         10  DN-3 . . .  .
         10  DN-4 . . .  .
     05  DN-5 . . .  .
66   DN-6 RENAMES DN-2 THROUGH DN-3.
```



DN-6 is indeterminate

Example 4 (Invalid)

```
01   RECORD-IV.
     05  DN-1.
         10  DN-2A . . .  .
         10  DN-2B . . .  .
         10  DN-2C REDEFINES DN-2B.
             15  DN-2 . . .  .
             15  DN-2D . . .  .
     05  Dn-3 . . .  .
66   DN-4 RENAMES DN-1 THROUGH DN-2.
```



DN-4 is indeterminate

Figure 10-4. Valid and Invalid Uses of the RENAMES Clause

Format 3

This format describes condition names. A condition name is a name you give that associates a value(s) or a range(s) of values (or both) with a conditional variable.

A conditional variable is a data item that you can, in turn, associate with a condition name. The following rules for condition-name entries apply:

- Any entry beginning with level number 88 is a condition-name entry.
- You must place the condition-name entries associated with a particular conditional variable immediately after the conditional-variable entry. You can make the conditional variable any elementary data description entry except another condition name, an index data item, or a level-66 entry.
- You can associate a condition name with a group item data description entry. The following rules apply:
  - You must make the condition-name value a nonnumeric literal or figurative constant.
  - You must not use a condition-name value that is larger in size than the sum of the sizes of all the elementary items within the group.
  - You cannot include a JUSTIFIED or SYNCHRONIZED clause in any element within the group.
  - You can use no USAGE other than USAGE IS DISPLAY within the group.
- You can use condition names both at the group level and at subordinate levels within the group.
- The relation test implied by the definition of a condition name at the group level is performed according to the rules for comparing nonnumeric operands regardless of the nature of elementary items within the group.
- You must separate successive operands either with a space or with a comma or a semicolon followed by a space.
- You must end each entry with a period followed by a space.
- You must not qualify the condition name when you use it in a REDEFINES clause.

Examples of both elementary and group condition-name entries are given under *VALUE Clause* later in this chapter.

IBM Extension

Format 4-Boolean Data

Use this format for Boolean data items in all sections. The following rules apply:

- You must implicitly or explicitly define USAGE as DISPLAY.
- In the OCCURS clause, you cannot use the ASCENDING/DESCENDING KEY phrase for Boolean data items.
- You must use the INDICATOR clause at an elementary level only.
- You can compare a Boolean data item only with another Boolean data item.
- You can use only EQUAL or NOT EQUAL comparisons for Boolean data items.

End of IBM Extension

Level Numbers

The level number gives the hierarchy of data within a record and also identifies special-purpose data entries.

Format

```
level-number
```

The following rules for level numbers apply:

- A level number begins a:
  - Data description entry
  - Regrouped item
  - Condition-name entry.
- You must begin level numbers 01 and 77 in area A.
- You can begin level numbers 02 through 49, 66, and 88 in either area A or area B, and you must follow them with a space.
- You can substitute single-digit level numbers 1 through 9 for level numbers 01 through 09.

Data Name or FILLER Clause

A data name explicitly identifies the data being described; the key word FILLER identifies an item that is never explicitly referenced in the program.

Format

```
data-name
FILLER
```

In a data description entry, you must make the first word after the level number either the data name or the key word FILLER. The data name identifies a data item by referring to the field, not to a particular value. This data item can assume a number of different values during the course of a program.

You can begin a data name anywhere in area B. You must place a period at the end of a data name, and you must include at least one alphabetic character.

You cannot qualify entries at level numbers 01 and 77 in the Working-Storage and Linkage sections, so you must use unique data names. You do not need unique data names for subordinate data names that can be qualified.

The key word FILLER specifies an elementary item to which you never explicitly refer to in a record. You can write the word FILLER anywhere in area B. You must place a period at the end of the entry.

FILLER items are ignored in the following statements:

- MOVE CORRESPONDING
- ADD CORRESPONDING
- SUBTRACT CORRESPONDING.

IBM Extension

You can use a FILLER item as a group item definition. You can then use the appropriate data name to reference subordinate data items.

End of IBM Extension

REDEFINES Clause

The REDEFINES clause indicates that the same storage area can contain different data items. Redefinition can save storage by letting you use the same area for different purposes.

Format

```
level-number data-name-1 REDEFINES data-name-2
```

The level number and data-name-1 are not part of the REDEFINES clause itself and are included in the format only for clarity.

If you use the REDEFINES clause, it must be the first entry following data-name-1.

You must make the level numbers of data-name-1 and data-name-2 identical and you must not make them level-66 or level-88 entries.

Data-name-2 is the redefined item.

Data-name-1 is the redefining item and is an alternative description for the data-name-2 area.

Implicit redefinition is assumed when you make more than one level-01 entry subordinate to an FD entry. In such level-01 entries, you must not use the REDEFINES clause.

Redefinition begins at data-name-1 and ends when a level number less than or equal to that of data-name-2 is encountered. You cannot have an entry with a level number numerically lower than those of data-name-1 and data-name-2 between these entries.

In the following example, A is the redefined item, and B is the redefining item. Redefinition begins with B and includes the two subordinate items B-1 and B-2. Redefinition ends when the level-05 item C is encountered.

```
05  A         PICTURE X(6).
05  B REDEFINES A.
      10 B-1 PICTURE X(2).
      10 B-2 PICTURE 9(4).
05  C         PICTURE 99V99.
```

You cannot have a REDEFINES clause or an OCCURS clause in the data description entry for data-name-2, the redefined item. You can make the redefined item subordinate to an item that contains either clause. If you make the redefined item subordinate to an item that contains an OCCURS clause, you must not subscript or index data-name-2 in the REDEFINES clause (the redefined item).

You cannot have an OCCURS DEPENDING ON clause in the redefined item, the redefining item, or any items subordinate to them.

When you use data-name-1, the redefining item, with a level number other than 01, it must give a storage area of the same size as the redefined item data-name-2.

You can have more than one redefinition of the same storage area. You must place the entries giving the new descriptions of the storage area immediately after the description of the redefined area without having intervening entries that define new character positions. Multiple redefinitions must all use the data name of the original entry that defined this storage area. For example:

```
05  A               PICTURE 9999.
05  B REDEFINES A   PICTURE 9V999.
05  C REDEFINES A   PICTURE 99V99.
```

You must not have any VALUE clauses in the redefining entry (identified by data-name-1) and any subordinate entries. This rule does not apply to condition names.

You can redefine data items within an area without their lengths being changed. For example:

```
05  NAME-2.
    10 SALARY    PICTURE XXX.
    10 SO-SEC-NO PICTURE X(9).
    10 MONTH     PICTURE XX.
05  NAME-1 REDEFINES NAME-2.
    10 WAGE      PICTURE XXX.
    10 EMP-NO    PICTURE X(9).
    10 YEAR      PICTURE XX.
```

You can also rearrange data items within an area. For example:

```
05  NAME-2.
    10 SALARY    PICTURE XXX.
    10 SO-SEC-NO PICTURE X(9).
    10 MONTH     PICTURE XX.
05  NAME-1 REDEFINES NAME-2.
    10 EMP-NO    PICTURE X(6).
    10 WAGE      PICTURE 999V999.
    10 YEAR      PICTURE XX.
```

When you redefine an area, all descriptions of the area are always in effect; that is, redefinition does not cause any data to be erased and does not supersede the previous description. Thus, if you have used B REDEFINES A, either of the two procedural statements MOVE X TO B and MOVE Y TO A could be performed at any point in the program.

In the first case, the area described as B would assume the value of X. In the second case, the same physical area (described now as A) would assume the value of Y. If the second statement is performed immediately after the first, the value of Y replaces the value of X in the one storage area.

You need not make the USAGE of a redefining data item the same as that of a redefined item. This does not, however, cause any change in existing data. For example:

```
05  B PICTURE 99 USAGE DISPLAY VALUE 8.
05  C REDEFINES B PICTURE S99 USAGE
      COMPUTATIONAL-4.
05  A PICTURE S99 USAGE COMPUTATIONAL-4.
```

The bit configuration of the DISPLAY value 8 is 1111 0000 1111 1000. Redefining B does not change the bit configuration of the data in the storage area; therefore, the two statements, ADD B TO A and ADD C TO A give different results. In the first case, the value 8 is added to A (because B has USAGE DISPLAY specified). In the second statement, the value -48 is added to A (because C has specified USAGE COMPUTATIONAL-4 specified), and the bit configuration (truncated to 2 decimal digits) in the storage area has the binary value -48.

Unexpected results might occur if you move a redefining item to a redefined item (that is, if B REDEFINES C and the statement MOVE B TO C is performed).

Unexpected results might also occur if you move a redefined item to a redefining item (from the previous example, unexpected results occur if the statement MOVE C TO B is performed).

You can use the REDEFINES clause for an item within any area being redefined (that is, an item subordinate to a redefined item). For example:

```
05  REGULAR-EMPLOYEE.
    10  LOCATION           PICTURE A(8).
    10  GRADE              PICTURE X(4).
    10  SEMI-MONTHLY-PAY   PICTURE 9999V99.
    10  WEEKLY-PAY REDEFINES
        SEMI-MONTHLY-PAY   PICTURE 999V999.

05  TEMPORARY-EMPLOYEE REDEFINES
    REGULAR-EMPLOYEE.
    10  LOCATION     PICTURE A(8).
    10  FILLER       PICTURE X(6).
    10  HOURLY-PAY   PICTURE 99V99.
```

The REDEFINES clause may also be specified for an item subordinate to a redefining item. For example:

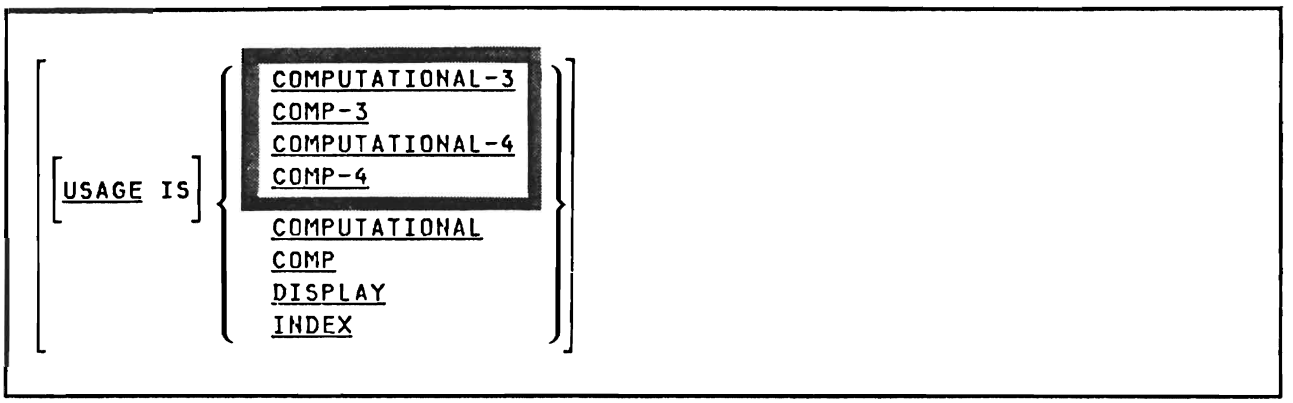
```
05  REGULAR-EMPLOYEE.
    10  LOCATION  PICTURE A(8).
    10  GRADE     PICTURE X(4).
    10  SEMI-MONTHLY-PAY
                  PICTURE 999V999.

05  TEMPORARY-EMPLOYEE REDEFINES
    REGULAR-EMPLOYEE.
    10  LOCATION     PICTURE A(8).
    10  FILLER       PICTURE X(6).
    10  HOURLY-PAY   PICTURE 99V99.
    10  CODE-H REDEFINES HOURLY-PAY
                  PICTURE 9999.
```

USAGE Clause

The USAGE clause specifies the format of a data item in storage. The USAGE clause can be specified for an entry at any level; if it is specified at the group level, it applies to each elementary item in the group. The usage of an elementary item cannot contradict the usage of a group to which the elementary item belongs.

The format of the data specified by the USAGE clause may be restricted if certain Procedure Division statements are used.

Format



When you do not use the USAGE clause at either the group or elementary level, USAGE IS DISPLAY is assumed.

INDEX Phrase: The USAGE IS INDEX clause specifies that the data item named has an indexed format and, therefore, is an index data item. The index data item is an elementary item that you can use to save index-name values for future reference.

The USAGE IS INDEX clause is described in detail under *Using the Table Handling Facilities* in Chapter 13.

DISPLAY Phrase: The DISPLAY option can be explicit or implicit. It specifies that the data item is stored in character form, 1 character per byte. This corresponds to the form in which information is represented for keyboard input or for printed output. You can use USAGE IS DISPLAY for the following types of items:

- Alphabetic
- Alphanumeric
- Alphanumeric edited
- Numeric edited
- Zoned decimal (numeric)
- Boolean.

Alphabetic, alphanumeric, alphanumeric edited, Boolean, and numeric edited items are discussed in the description of the PICTURE clause later in this chapter.

Zoned Decimal Items: These items are sometimes referred to as external decimal items. Each digit of a number is represented by a single byte. The 4 high-order bits of each byte are zone bits; the 4 high-order bits of the low-order byte represent the sign of the item. If the number is positive, these 4 bits contain hex F. If the number is negative, these 4 bits contain hex D. The 4 low-order bits of each byte contain the value of the digit. When you use zoned decimal items for computations, the compiler performs the necessary conversions. The maximum length of a zoned decimal item is 18 digits.

The only characters you can place in the PICTURE character string of a zoned item are:

- 9 (one or more numeric character positions)
- S (one operational sign)
- V (one implied decimal point)
- P (one or more decimal scaling positions).

Examples of zoned decimal items are shown in Figure 10-5.

Computational Phrases: The term computational refers to the following phrases of the USAGE clause:

COMPUTATIONAL or COMP (zoned decimal).

IBM Extension

COMPUTATIONAL-3 or COMP-3 (packed decimal)

COMPUTATIONAL-4 or COMP-4 (binary).

End of IBM Extension

A computational item represents a value to be used in arithmetic operations and you must make it numeric. If you describe the USAGE of a group item with any of these options, it is the elementary items within the group that have this usage. The group itself is considered nonnumeric and you cannot use it in numeric operations except with the CORRESPONDING phrase. The maximum length of a computational item is 18 decimal digits.

The only characters you can place in the PICTURE character string of a computational item are:

- 9 (one or more numeric character positions)
- S (one operational sign)
- V (one implied decimal point)
- P (one or more decimal scaling positions).

This page is intentionally left blank.

The COMPUTATIONAL phrase is in zoned decimal format. Each digit of the number is represented by a single byte. The 4 leftmost bits of each byte are zone bits; the 4 leftmost bits of the rightmost byte represent the sign of the item. The 4 rightmost bits of each byte contain the value of the digit. You can place any of the digits 0 through 9, plus a sign, in a zoned decimal item.

IBM Extension

Use the COMPUTATIONAL-3 phrase for packed decimal items. Such an item appears in storage as 2 digits per byte, with the sign contained in the 4 rightmost bits of the rightmost byte. If the number is positive, these 4 bits contain hexadecimal F. If the number is negative, these 4 bits contain hexadecimal D.

You can place any of the digits 0 through 9, plus a sign, in a packed decimal item. If you do not place an S in the PICTURE character string of a packed decimal item, the sign position is occupied by a bit configuration that is interpreted as positive, but does not represent an overpunch.

Use the COMPUTATIONAL-4 option for binary data items. Such items have decimal equivalents consisting of the decimal digits 0 through 9, plus a sign.

The amount of storage occupied by a binary data item depends on the number of decimal digits you define in its PICTURE clause:

Digits in PICTURE Clauses	**Storage Occupied**
1 through 4	2 bytes
5 through 9	4 bytes
10 through 18	8 bytes

The leftmost bit of the storage area is the operational sign.

Examples of packed decimal and binary items are shown in Figure 10-5.

End of IBM Extension

Item	Description	Value	Internal Representation*
Zoned Decimal	PIC S9999 DISPLAY	+1234	F1 F2 F3 F4
		-1234	F1 F2 F3 D4
		1234	F1 F2 F3 F4
	PIC 9999 DISPLAY	+1234	F1 F2 F3 F4
		-1234	F1 F2 F3 F4
		1234	F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN LEADING	+1234	F1 F2 F3 F4
		-1234	D1 F2 F3 F4
		1234	F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN TRAILING SEPARATE	+1234	F1 F2 F3 F4 4E
		-1234	F1 F2 F3 F4 60
		1234	F1 F2 F3 F4 4E
	PIC S9999 DISPLAY SIGN LEADING SEPARATE	+1234	4E F1 F2 F3 F4
		-1234	60 F1 F2 F3 F4
	(COMP applies to all zoned decimal data formats)	1234	4E F1 F2 F3 F4
Packed Decimal	PIC S9999 COMP-3	+1234	01 23 4F
		-1234	01 23 4D
	PIC 9999 COMP-3	+1234	01 23 4F
		-1234	01 23 4F
Binary	PIC S9999 COMP-4	+1234	04 D2
		-1234	FB 2E
	PIC 9999 COMP-4	+1234	04 D2
		-1234	04 D2

*The internal representation of each byte is shown as two hex digits. The bit configuration for each digit is as follows:

Hex Digit	Bit Configuration	Hex Digit	Bit Configuration
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Notes:

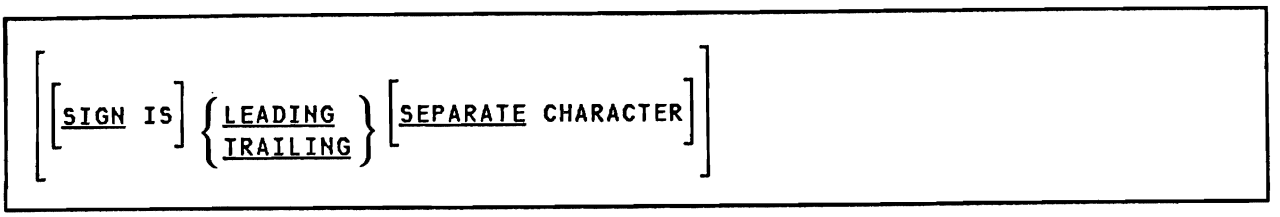
1. The leftmost bit of a binary number represents the sign: 0 is positive, 1 is negative.
2. Negative binary numbers are represented in twos complement form.
3. Hexadecimal 4E represents the EBCDIC character +. Hexadecimal 60 represents the EBCDIC character -.
4. Specification of SIGN TRAILING (without the SEPARATE CHARACTER option) is the equivalent of the standard action of the compiler.
5. Hexadecimal 1C, which is the DUP KEY, requires definition using a specific method. This method is found in Appendix A.

Figure 10-5. Internal Representation of Numeric Items

SIGN Clause

The SIGN clause gives the position and mode of representation of the operational sign for a numeric entry.

Format



You can use the SIGN clause only for a signed numeric data description entry (that is, one with a PICTURE character string that contains an S), or for a group item that contains at least one such elementary entry. USAGE IS DISPLAY must be specified either explicitly or implicitly.

You can have only one SIGN clause for each data description entry. The SIGN clause is required only when an explicit description of the properties or position of the operational sign is necessary.

The SIGN clause defines the position and mode of representation of the operational sign for the numeric data description entry to which it applies, or for each signed numeric data description entry subordinate to the group to which it applies.

If you do not use the SEPARATE CHARACTER phrase, then:

- The operational sign is presumed to be associated with the LEADING or TRAILING digit position (whichever you used) of the elementary numeric data item.
- The character S in the PICTURE character string is not counted in determining the size of the item (in terms of standard data format characters).

If you use the SEPARATE CHARACTER phrase, then:

- The operational sign is presumed to be the LEADING or TRAILING character position (whichever you used) of the elementary numeric data item. This character position is not a digit position.
- The character S in the PICTURE character string is counted in determining the size of the data item (in terms of standard data format characters).
- Use the + character for the positive operational sign.
- Use the - character for the negative operational sign.
- If you do not use a + or a - in the data at object time, an error occurs and the program ends abnormally.

Every numeric data description entry with a PICTURE character string that contains the symbol S is a signed numeric data description entry. If you also use the SIGN clause for such an entry and conversion is necessary for computations or comparisons, the conversion takes place automatically.

If you do not use a SIGN clause for a signed numeric data description entry, the position and method of representation for the operational sign is determined as explained in the USAGE clause description.

OCCURS Clause

The OCCURS clause specifies tables with elements that you can refer to by indexing or subscripting. The OCCURS clause is described under *Data Division Table Handling* in Chapter 13.

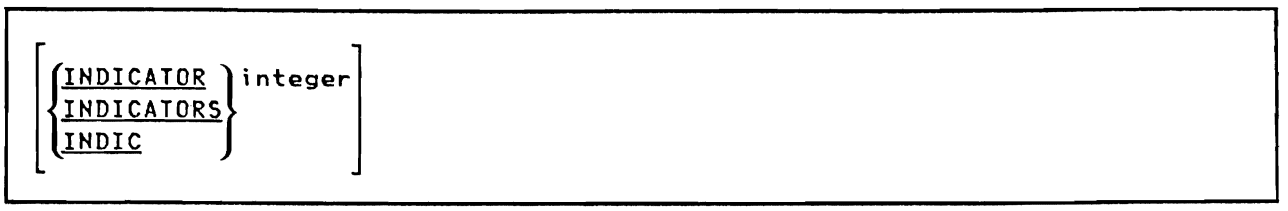
IBM Extension

OCCURS Clause with Boolean Data Items

If you use both the OCCURS clause and the INDICATOR clause at an elementary level, a table of Boolean data items is defined with each element in the table corresponding to an external indicator.

INDICATOR Clause

The INDICATOR clause associates a $SFGR or IDDU indicator number with a Boolean data item. The format is:



You must make the integer greater than or equal to 1 and less than or equal to 99.

You must use the INDICATOR clause only at an elementary level.

Since you can only have a value of 0 or 1 in an indicator, you can associate the indicator only with a Boolean data item.

OCCURS Clause with the INDICATOR Clause

If you use both the OCCURS clause and the INDICATOR clause at an elementary level, a table of Boolean data items is defined with each element in the table corresponding to an external indicator. The first element in the table corresponds to the indicator number you used in the INDICATOR clause, the second element corresponds to the indicator that sequentially follows the indicator you used in the INDICATOR clause.

For example, if you coded the following:

```
7 SWITCHES PIC 1 OCCURS 10 TIMES
   INDICATOR 16.
```

then:

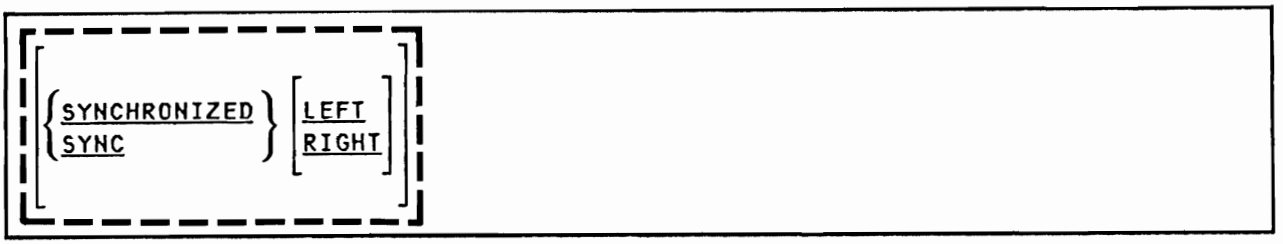
SWITCHES (1) corresponds to $SFGR or IDDU indicator 16,
SWITCHES (2) corresponds to $SFGR or IDDU indicator 17, . . .
SWITCHES (10) corresponds to $SFGR or IDDU indicator 25.

End of IBM Extension

SYNCHRONIZED Clause

The SYNCHRONIZED clause gives the alignment of an elementary item on a proper boundary in storage.

Format



The SYNCHRONIZED clause is treated as documentation only. The SYNCHRONIZED clause is never required. You can use it only at the elementary level. SYNC is an abbreviation for SYNCHRONIZED and has the same meaning.

JUSTIFIED Clause

The JUSTIFIED clause overrides standard positioning rules for a receiving item of the alphabetic or alphanumeric categories.

Format

```
[ { JUSTIFIED } RIGHT ]
  { JUST      }
```

You can use the JUSTIFIED clause only at the elementary level. JUST is an abbreviation for JUSTIFIED and has the same meaning.

You must not use the JUSTIFIED clause:

- For a numeric item
- For any item for which you use editing
- With level-66 (RENAMES) entries
- With level-88 (condition-name) entries.

When you use the JUSTIFIED clause for a receiving item, the data is aligned at the rightmost character position in the receiving item. Also:

- If the sending item is larger than the receiving item, the leftmost characters are truncated.
- If the sending item is smaller than the receiving item, the unused character positions at the left are filled with spaces.
- If the sending and receiving items are the same size, the JUSTIFIED clause does not affect the result.

When you leave out the JUSTIFIED clause, the rules for standard alignment are followed.

The following shows the difference between standard and justified alignment:

Alignment	Sending Field Value	Receiving Field Value
Standard	THE	THEbb
Right justified	THE	bbTHE
Standard	THEbb	THEbb
Right justified	THEbb	THEbb

BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause specifies that an item is to be filled entirely with spaces when its value is 0.

Format

You can use the BLANK WHEN ZERO clause only for elementary numeric or numeric edited items. When you use it for a numeric item, the item is considered to be a numeric edited item.

If you use the BLANK WHEN ZERO clause, the item contains nothing but spaces when its value is 0.

You must not use the BLANK WHEN ZERO clause for level-66 or level-88 items.

IBM Extension

When you use both the BLANK WHEN ZERO clause and the asterisk (*) as a suppression symbol for the same data description entry, zero suppression editing overrides the function of the BLANK WHEN ZERO clause.

End of IBM Extension

VALUE Clause

The VALUE clause gives the initial contents of a data item, or the value(s) associated with a condition name. The two formats for the VALUE clause are as follows:

Format 1

Format 2

```
88 condition-name {VALUE IS  } literal-1 [ THROUGH  literal-2 ]
                  {VALUES ARE}           [ THRU                ]

[ literal-3 [ {THROUGH} literal-4 ] ... . ]
            [ {THRU   }           ]
```

Level number 88 and the condition name are not part of the format 2 VALUE clause itself and are included in the format only for clarity. The use of the VALUE clause differs with the Data Division section in which it is used.

File and Linkage Sections: You only use the VALUE clause in condition-name entries.

Working-Storage Section: You use the VALUE clause in condition-name entries and also to give the initial value of any data item; the data item assumes the given value when the program begins to run. If the initial value is not explicitly specified, it is unpredictable.

General Considerations

The key words THRU and THROUGH are equivalent.

You must not use the VALUE clause for any item with variable length.

For group entries, you must not use the VALUE clause if the entry or an entry subordinate to it contains any of the following clauses:

- JUSTIFIED
- SYNCHRONIZED
- USAGE (other than USAGE DISPLAY).

You must not use a VALUE clause that conflicts with other clauses in the data description entry or in the data description of this entry's hierarchy. The following rules apply:

- Wherever you use a literal, you can substitute a figurative constant.
- If the item is numeric, you must make all VALUE clause literals numeric. If the literal defines the value of a Working-Storage item, the literal is aligned according to the rules for numeric moves with one additional restriction: you must not give the literal a value that requires truncation of nonzero digits. If the literal is signed, you must place a sign symbol (S) in the associated PICTURE character string.

- You must give all numeric literals in a VALUE clause of an item a value that is within the range of values indicated by the PICTURE clause for that item. For example, for PICTURE 99PPP, the literal must be within the range 1000 through 99,000 or be 0. For PICTURE PPP99, the literal must be within the range .00000 through .00099.

- If the item is an elementary or group alphabetic, alphanumeric, alphanumeric edited, or numeric edited item, you must make all VALUE clause literals nonnumeric. The number of characters in the literal must not be larger than the size of the item.

- The functions of the editing characters or attributes in a PICTURE clause are ignored in determining the initial appearance of the item described. Editing characters are included in determining the size of the item, however, so you must include any editing character in the literal. For example, if you define the item as PICTURE +999.99 and the value is +12.34, then you should write the VALUE clause as VALUE '+012.34'.

- You can initialize no more than 32,767 bytes with a single VALUE clause.

Format 1 Considerations

This format gives the initial value of a data item in storage. Initialization is independent of any BLANK WHEN ZERO or JUSTIFIED clause you used.

You must not use a format 1 VALUE clause for an entry that contains or is subordinate to an entry in which you used a REDEFINES or OCCURS clause.

If you use the VALUE clause at the group level, you must make the literal nonnumeric or a figurative constant. The group area is initialized without consideration for the subordinate entries within this group. In addition, you must not use the VALUE clause for subordinate entries within this group.

IBM Extension

Boolean Considerations: The values you can use for a Boolean literal are B'0', B'1', and ZERO(S).

End of IBM Extension

Format 2 Considerations

This format associates a value, values, or range(s) of values with a condition name. You need a separate level-88 entry for each such condition name.

You must use the VALUE clause in a condition-name entry and you must make it the only clause in the entry. Each condition-name entry is associated with a preceding conditional variable. Thus, you must always precede every level-88 entry with either the entry for the conditional variable or with another level-88 entry when several condition names apply to one conditional variable. Such level-88 entries implicitly have the PICTURE characteristics of the conditional variable.

You can qualify every condition name with the name of its associated conditional variable and with the qualifier(s) of the conditional variable. If the associated conditional variable requires subscripts or indexes, you must subscript or index each procedural reference to the condition name as required for the conditional variable.

When you use only literal-1, the condition name is associated with a single value.

When you use literal-1, literal-3, and so on, the condition name is associated with several single values.

When you use literal-1 THRU literal-2, the condition name is associated with one range of values.

When you use literal-1 THRU literal-2, literal-3 THRU literal-4, and so on, the condition name is associated with more than one range of values. You must make literal-1 less than literal-2, literal-3 less than literal-4, and so on.

You can use one or more single values and one or more ranges of values in a single Format 2 VALUE clause.

You must make the type of literal in a condition-name entry consistent with the data type of the conditional variable. In the following example, CITY-COUNTY-INFO, COUNTY-NO, and CITY are conditional variables; the associated condition names immediately follow the level-number 88. The PICTURE clause associated with COUNTY-NO limits the condition-name value to a 2-digit numeric literal. The PICTURE clause associated with CITY limits the condition-name value to a 3-character nonnumeric literal. Any values for the condition names associated with CITY-COUNTY-INFO cannot exceed 5 characters, and the literal (because this is a group item) must be nonnumeric:

```
05  CITY-COUNTY-INFO.
      88  BRONX           VALUE  '03NYC'.
      88  BROOKLYN        VALUE  '24NYC'.
      88  MANHATTAN       VALUE  '31NYC'.
      88  QUEENS          VALUE  '41NYC'.
      88  STATEN-ISLAND   VALUE  '43NYC'.
    10 COUNTY-NO PICTURE 99.
      88  DUTCHESS        VALUE  14.
      88  KINGS           VALUE  24.
      88  NEW YORK        VALUE  31.
      88  RICHMOND        VALUE  43.
    10 CITY      PICTURE X(3).
      88  BUFFALO         VALUE  'BUF'.
      88  NEW-YORK-CITY   VALUE  'NYC'.
      88  POUGHKEEPSIE    VALUE  'POK'.
05  POPULATION. . .
```

The following example shows how to use the THRU option. In this example, the number of miles a person drives to work each day is categorized.

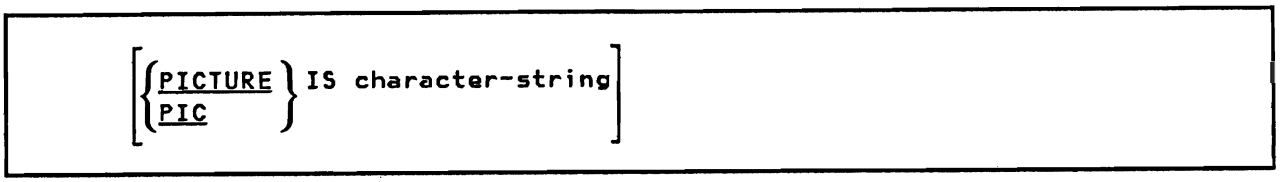
```
05  MILEAGE PIC 9(2)V9.
    88  LOW VALUE 0 THRU 09.9.
    88  MED VALUE 10.0 THRU 19.9.
    88  HIGH VALUE 20.0 THRU 99.9.
```

Condition names are used procedurally in condition-name conditions, and are described under *Conditional Expressions* in Chapter 11.

PICTURE Clause

The PICTURE clause gives the general characteristics and editing requirements of an elementary item.

Format



You must use the PICTURE clause for every elementary item except an indexed data item. You can use the PICTURE clause only at the elementary level. PIC is an abbreviation for PICTURE and has the same meaning.

The character string is made up of certain COBOL characters used as symbols. The allowable combinations determine the category of the data item. You can include no more than 30 characters in the character string.

Symbols Used in the PICTURE Clause

The functions of each PICTURE clause symbol are defined in the following list. Any punctuation character you include in the PICTURE character string is not considered a punctuation character, but rather a PICTURE character string symbol.

A — Each A in the character string represents a character position that can contain only a letter of the alphabet or a space.

B — Each B in the character string represents a character position into which the space character will be inserted.

P — The P indicates an assumed decimal scaling position and gives the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character P is not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items or in items that appear as arithmetic operands. In any operation converting data from one form of internal representation to another, if you describe the item being converted with the PICTURE symbol P, each digit position you describe with a P is considered to contain the value 0, and the size of the item is considered to include these 0 digit positions.

For example, PICTURE PPP99 DISPLAY defines a 2-character item with a value that is 0 or that ranges from .00001 through .00099. PICTURE 99PPP DISPLAY defines a 2-character item with a value that is 0 or that ranges from 1000 through 99,000.

You can place the scaling position character P only to the left or right of the other characters in the string as a continuous string of Ps within a PICTURE description. The sign character S and the assumed decimal point V are the only characters that you can place to the left of a leftmost string of Ps. Because the scaling position character P implies an assumed decimal point (to the left of the Ps if the Ps are leftmost PICTURE characters; to the right of the Ps if the Ps are rightmost PICTURE characters), the assumed decimal point symbol V is redundant as either the leftmost or rightmost character within such a PICTURE description.

S — The symbol S is used in a PICTURE character string to indicate the presence (but not the representation or, necessarily, the position) of an operational sign. You must write the sign as the leftmost character in the PICTURE string. An operational sign indicates whether the value of an item involved in an operation is positive or negative. The symbol S is not counted in determining the size of the elementary item, unless you use the SEPARATE CHARACTER option in an associated SIGN clause.

V — The V is used in a character string to indicate the location of the assumed decimal point. You can use the V only once in a character string. The V does not represent a character position and, therefore, is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string, the V is redundant.

X — Each X in the character string represents a character position that can contain any allowed character from the EBCDIC set.

Z — Each Z in the character string represents a leading numeric character position. When that position contains a 0, the 0 is replaced by a space character. Each Z is counted in the size of the item.

IBM Extension

1 A single 1 indicates a Boolean data item. If you place a 1 in the PICTURE character string, it must be the only character.

End of IBM Extension

9 Each 9 in the character string represents a character position that contains a numeral and is counted in the size of the item.

0 Each 0 in the character string represents a character position into which the numeral 0 will be inserted. Each 0 is counted in the size of the item.

/ Each slash in the character string represents a character position into which the slash character will be inserted. Each slash is counted in the size of the item.

, Each comma in the character string represents a character position into which a comma will be inserted. This character is counted in the size of the item. You cannot make the comma insertion character the last character in the PICTURE character string.

. When a period appears in the character string, it is an editing symbol that represents the decimal point for alignment purposes. In addition, it represents a character position into which a period will be inserted. This character is counted in the size of the item. You cannot make the period insertion character the last character in the PICTURE character string.

Note: For a given program, the functions of the period and comma are exchanged if you use the clause DECIMAL-POINT IS COMMA in the SPECIAL-NAMES paragraph. In this exchange, the rules for the period apply to the comma, and the rules for the comma apply to the period wherever they appear in a PICTURE clause.

+(CR) -(DB) These symbols are editing sign control symbols. Each symbol represents the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in one character string. Each character used in the symbol is counted in determining the size of the data item.

* Each asterisk (check protect symbol) in the character string represents a leading numeric character position into which an asterisk will be placed when that position contains a 0. Each asterisk is counted in the size of the item.

IBM Extension

Within a given data description entry, the use of the check protect symbol overrides the BLANK WHEN ZERO clause.

End of IBM Extension

'CS' The currency symbol in the character string represents a character position into which a currency symbol is to be placed. The currency symbol in a character string is represented either by the symbol $ or by the single character you used in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division. The currency symbol is counted in the size of the item.

Note: Because you can replace the currency symbol in the CURRENCY SIGN clause, the term 'CS' is used throughout this book rather than the actual currency symbol ($).

Figure 10-6 gives the order in which you must use PICTURE clause symbols.

First Symbol		Second Symbol																					
		Nonfloating Insertion Symbols									Floating Insertion Symbols						Other Symbols						
		B	0	/	,	.	{+ −}[1]	{+ −}[1]	{CR DB}	$	{Z *}[1]	{Z *}[1]	{+ −}[1]	{+ −}[1]	$[2]	$[2]	9	A X	S	V	P[1]	P[1]	1[3]
Nonfloating Insertion Symbols	B	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x	
	0	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x	
	/	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x	
	,	x	x	x	x	x	x			x	x	x	x	x	x	x	x			x		x	
	.	x	x	x	x		x			x	x		x		x		x						
	{+ −}																						
	{+ −}	x	x	x	x	x				x	x	x			x	x	x			x	x	x	
	{CR DB}	x	x	x	x	x				x	x	x			x	x	x			x	x	x	
	$						x																
Floating Insertion Symbols	{Z *}	x	x	x	x		x			x	x												
	{Z *}	x	x	x	x	x	x			x	x	x								x		x	
	{+ −}	x	x	x	x					x			x										
	{+ −}	x	x	x	x	x				x			x	x						x		x	
	$	x	x	x	x		x								x								
	P	x	x	x	x	x	x								x	x				x		x	
Other Symbols	9	x	x	x	x	x	x			x	x		x		x		x	x	x	x		x	
	A X	x	x	x													x	x					
	S																						
	V	x	x	x	x		x			x	x		x		x		x		x		x		
	P	x	x	x	x		x			x	x		x		x		x		x		x		
	P						x			x									x	x		x	

[1] Nonfloating insertion symbols + and −, floating insertion symbols Z, *, +, −, and $, and other symbol P appear twice in the above table. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of the symbol in the table represents its use to the right of the decimal point position.

[2] $ is the default value for the currency symbol. This value may be replaced by a character specified in the currency SIGN clause. At least one of the symbols A, X, Z, 9, or *, or at least two of the symbols +, −, or $ must be present in a PICTURE character string.

[3] The character 1 must appear alone in the character string.

An X at an intersection indicates that the symbol(s) at the top of the column may, in a given character string, appear anywhere to the left of the symbol(s) at the left of the row.

Braces ({ }) indicate items that are mutually exclusive.

Figure 10-6. PICTURE Clause Symbol Order

Character String Representation: You can use the following symbols more than once in one PICTURE character string:

A B P X Z 9 0 / , + - * 'CS'.

Each time you use one of these symbols in the character string, it represents an occurrence of that character or set of allowable characters in the data item.

An integer enclosed in parentheses immediately following any of these symbols gives the number of times that symbol occurs consecutively. You cannot have more than 32,767 consecutive occurrences.

For example, the following two uses of the PICTURE clause are equivalent:

PICTURE IS $99999.99CR

PICTURE IS $9(5).99CR.

You can use the following five symbols only once in one PICTURE character string:

S V . CR DB.

Data Categories and PICTURE Considerations: The combinations of PICTURE symbols that you can use determine the data category of the item. Rules for each category follow.

- Alphabetic items:
  - You can use only the symbols A and B in the PICTURE character string.
  - You must use only the 26 letters of the alphabet and the space character as the contents of the item in standard data format.
  - You must either specify or imply USAGE DISPLAY.
  - You must use a nonnumeric literal in any associated VALUE clause.
- Numeric items:
  - You can use only the symbols 9, P, S, and V in the PICTURE character string.
  - You must have from 1 through 18 digit positions.
  - You must make the contents of a numeric item a combination of the digits 0 through 9. You can use an operational sign in the numeric item. If you place an S in the PICTURE, the contents of the item are treated as a positive or negative value, depending on the operational sign present in the data. If you do not place an S in the PICTURE, the contents of the item are treated as an absolute value.
  - If you use a VALUE clause for an elementary numeric item, you must make the literal numeric. If you use a VALUE clause for a group item consisting of elementary numeric items, the group is considered alphanumeric, and you must therefore make the literal nonnumeric.
  - You can make the USAGE of the item DISPLAY or COMPUTATIONAL.

IBM Extension

You can make the USAGE be COMPUTATIONAL-3 or COMPUTATIONAL-4.

End of IBM Extension

Examples of numeric items are shown in Figure 10-7.

PICTURE	Valid Range of Values
9999	0 through 9999
S99	-99 through +99
S999V9	-999.9 through +999.9
PPP999	0 through .000999
S999PPP	-1000 through -999000 and +1000 through +999000 or zero

Figure 10-7. Examples of Numeric Items

- Alphanumeric items:
  - You must use either of the following for the PICTURE character string:
    a. An entire string of the symbol X.
    b. Combinations of the symbols A, X, and 9. The item is treated as if the character string contained only the symbol X. A PICTURE character string containing all A's or all 9s does not define an alphanumeric item.
  - You can make the contents of the item in standard data format any allowable characters from the EBCDIC character set.
  - You must either specify or imply USAGE DISPLAY.
  - You must use a nonnumeric literal in any associated VALUE clause.

- Alphanumeric edited items:
  - You can use the following symbols in the PICTURE character string:

    A X 9 B 0 /

  - You must include at least one of the following combinations in the string:
    a. At least one B and at least one X
    b. At least one 0 and at least one X
    c. At least one X and at least one /
    d. At least one A and at least one 0
    e. At least one A and at least one /
  - You can use any allowed character from the EBCDIC character set as the contents of the item in standard data format.
  - You must either specify or imply USAGE DISPLAY.
  - You must use a nonnumeric literal in any associated VALUE clause. The literal is treated exactly as given; no editing is performed.

- Alphanumeric edited items are subject to simple-insertion editing only, using the symbols 0, B, and /.

- Numeric edited items:

  - You can use the following symbols in the PICTURE character string:

    B P V Z 9 0 / , . + - CR DB * 'CS'.

    The combinations of symbols you can use are determined from the PICTURE clause symbol order allowed (Figure 10-6), and the editing rules (see the following section). The following additional rules also apply:

    a. You must include at least one of the following symbols in the string:

       B / Z 0 , . * + - CR DB.

    b. You must represent the number of digit positions represented in the character string in the range of 1 through 18 inclusive.

    c. You must not have more than 30 total character positions in the string (including editing characters).

  - You must use the digits 0 through 9 as the contents of those character positions representing digits in standard data format.
  - You must either specify or imply USAGE DISPLAY.
  - You must use a nonnumeric literal in any associated VALUE clause. The literal is treated exactly as specified; no editing is performed.

IBM Extension

Boolean items: You must have a single character 1 for the PICTURE character string.

End of IBM Extension

PICTURE Clause Editing

There are two general methods of performing editing in a PICTURE clause:

- Insertion
- Suppression and replacement.

There are four types of insertion editing:

- Simple insertion
- Special insertion
- Fixed insertion
- Floating insertion.

There are two types of suppression and replacement editing:

- Zero suppression and replacement with asterisks
- Zero suppression and replacement with spaces.

The type of editing you can use for an item depends on its data category. The type of editing and the insertion symbols that you can use for each category are shown in Figure 10-8.

Category	Type of Editing	Valid Insertion Symbols
Alphabetic	Simple insertion	B
Numeric	None	None
Alphanumeric	None	None
Alphanumeric edited	Simple insertion	B 0 /
Numeric edited	All	B 0 / ,
Boolean	None	None

Figure 10-8. Valid Editing for Each Data Category

Simple Insertion Editing: You can use this type of editing for alphabetic, alphanumeric edited, and numeric edited items. The insertion symbols you can use for each category are shown in Figure 10-8.

Each insertion symbol is counted in the size of the item and represents the position within the item where the equivalent characters will be inserted. Examples of simple insertion editing are shown in Figure 10-9.

PICTURE Character String	Value of Data	Edited Result
X(10)/XX	Alphanumer01	Alphanumer/01
X(5)BX(7)	Alphanumeric	Alpha numeric
A(5)BA(5)	Alphabetic	Alpha betic
99,B999,B000	1234	01, 234, 000
99,999	12345	12,345

Figure 10-9. Examples of Simple Insertion Editing

Special Insertion Editing: You can use this type of editing only for numeric edited items.

The period is the special insertion symbol; it also represents the actual decimal point for alignment purposes.

The period insertion symbol is counted in the size of the item, and represents the position within the item where the actual decimal point will be inserted.

You must not use both the actual decimal point and the assumed decimal point (the symbol V) in one PICTURE character string.

Fixed Insertion Editing: You can use this type of editing only for numeric edited items. Use the following insertion symbols:

'CS' (currency symbol)

+ - CR DB (editing sign control symbols)

- In fixed insertion editing, you can use only one currency symbol and one editing sign control symbol in one PICTURE character string.
- Unless it is preceded by a + or - symbol, you cannot make the currency symbol the leftmost character position in the character string.
- When you use either + or - as a symbol, it must represent either the leftmost or rightmost character position in the character string.
- When you use CR or DB as a symbol, it must represent the rightmost two character positions in the character string.
- Editing sign control symbols produce results depending on the value of the data item, as shown in Figure 10-10.

Examples of fixed insertion editing are shown in Figure 10-11.

Editing Symbol in PICTURE Character String	Resulting Data Item Positive or Zero	Resulting Data Item Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

Figure 10-10. Editing Sign Control Results

PICTURE Character String	Value of Data	Edited Result
999.99+	+6555.556	555.55+
+9999.99	-6555.555	-6555.55
9999.99	+1234.56	1234.56
$999.99	-123.45	$123.45
-$999.99	-123.456	-$123.45
$9999.99CR	+123.45	$0123.45
$9999.99DB	-123.45	$0123.45DB

Figure 10-11. Examples of Fixed Insertion Editing

Floating Insertion Editing: You can use this type of editing only for numeric edited items. The following symbols are used:

'CS' + - .

Within one PICTURE character string, these symbols are mutually exclusive as floating insertion characters.

Specify floating insertion editing with a character string of at least two of the allowable floating insertion symbols to represent leftmost character positions in which these characters can be inserted.

The leftmost floating insertion symbol in the character string represents the leftmost limit at which this character can appear in the data item. The rightmost floating insertion symbol represents the rightmost limit at which this character can appear.

The second leftmost floating insertion symbol in the character string represents the leftmost limit at which numeric data can appear within the data item. Nonzero numeric data can replace all characters at or to the right of this limit.

Any simple insertion symbols (B 0 / ,) within or to the immediate right of the string of floating insertion symbols are considered part of the floating character string. If the period special insertion symbol is included within the floating string, it is considered to be part of the character string.

In a PICTURE character string, there are two methods by which you can represent floating insertion editing and perform editing:

- Any or all leading numeric character positions to the left of the decimal point are represented by the floating insertion symbol. When editing is performed, a single floating insertion character is placed to the immediate left of the first nonzero digit in the data or of the decimal point, whichever is the leftmost. The character positions to the left of the inserted character are filled with spaces.

- All the numeric character positions are represented by the floating insertion symbol. When editing is performed:

  - If the value of the data is 0, the entire data item will contain spaces.
  - If the value of the data is not 0, the result is the same as in method 1.

To avoid truncation, you must have a PICTURE character string at least the sum of:

- The total number of character positions in the sending item

- The total number of nonfloating insertion symbols in the receiving item

- One character for the floating insertion symbol.

Examples of floating insertion editing are shown in Figure 10-12.

PICTURE Character String	Value of Data	Edited Result
$$$$.99	.123	$.12
$$$9.99	.12	$0.12
$$,$$$,999.99	-1234.56	$1,234.56
+ +,+ + +,999.99	-123456.789	-123,456.78
$$,$$$,$$$.99CR	-1234567	$1,234,567.00CR
++,+++,+++,+++	0000.00	

Figure 10-12. Examples of Floating Insertion Editing

Zero Suppression and Replacement Editing: You can use this type of editing only for numeric edited items.

Use the symbols Z and * for zero suppression. These symbols are mutually exclusive in the PICTURE clause.

The following symbols are mutually exclusive as floating replacement symbols in one PICTURE character string:

Z * + - 'CS' .

Specify zero suppression editing with a string of one or more of the allowable symbols to represent leftmost character positions in which zero suppression and replacement editing can be performed.

Any simple insertion symbols (B 0 / ,) within or to the immediate right of the string of floating editing symbols are considered part of the string. If the period special insertion symbol is included within the floating editing string, it is considered to be part of the character string.

In a PICTURE character string, there are two methods by which you can represent zero suppression and perform editing:

- Any or all of the leading numeric character positions to the left of the decimal point are represented by suppression symbols. When editing is performed, any leading 0 in the data that appears in the same character position as a suppression symbol is replaced by the replacement character. Suppression stops at the leftmost character that:

  - Does not correspond to a suppression symbol
  - Contains nonzero data
  - Is the decimal point.

- All the numeric character positions in the PICTURE character string are represented by the suppression symbols. When editing is performed and the value of the data is nonzero, the result is the same as in the preceding rule. The following rules apply if the value of the data is 0:

  - If you used Z, the entire data item contains spaces.

- If you used *, the entire data item, except the actual decimal point, contains asterisks.

IBM Extension

You can use the asterisk (*) as a suppression symbol and the BLANK WHEN ZERO clause for the same entry. The asterisk overrides the BLANK WHEN ZERO clause if you specify both.

End of IBM Extension

Examples of zero suppression and replacement editing are shown in Figure 10-13.

PICTURE	Value of Data	Edited Result
****.**	0000.00	****.**
ZZZZ.ZZ		0000.00
ZZZZ.99	0000.00	.00
****.99	0000.00	****.00
ZZ99.99	0000.00	00.00
Z,ZZZ.ZZ+	+123.456	123.45+
*,***.**+	-123.45	**123.45
,*,***.**+	+12345678.9	12,345,678.90+
$Z,ZZZ,ZZZ.ZZCR	+12345.67	$ 12,345.67
$B*,***,***.**BBDB	-12345.67	$ ***12,345.67 DB

Figure 10-13. Examples of Zero Suppression and Replacement Editing

Procedure Division

Chapter 11. Procedure Division

This chapter contains a discussion of the Procedure Division concepts and organization.

Also, this chapter discusses in alphabetic order the Procedure Division statements, except for the input/output statements. The input/output statements are discussed in alphabetic order in Chapter 12.

Procedure Division Concepts

You must include a Procedure Division in every COBOL source program. The Procedure Division consists of optional Declaratives and procedures that contain the sections or paragraphs, sentences, and statements that solve a data processing problem.

The program begins running with the first statements in the Procedure Division, not including Declarative sections. Unless the logic flow gives some other order, statements are performed in the order in which they are given for compilation. The end of the Procedure Division and the physical end of the program is that physical position in a source program after which you place no more Procedure Division statements.

Declaratives

A Declarative section provides a way to begin procedures that are performed when an exceptional condition occurs that you want to test.

When you use Declarative sections, you must:

- Group them at the beginning of the Procedure Division
- Place the key word DECLARATIVES before the Declarative sections
- Place the key words END DECLARATIVES after the Declarative sections
- Divide the entire Procedure Division into sections.

Procedures

A *procedure* is a paragraph, a group of paragraphs, a section, or a group of sections within the Procedure Division. A *procedure name* is a user-defined name with which you identify a paragraph or a section.

A *section* consists of a section header and zero, one, or more than one successive paragraphs. A *section header* is a section name followed by the key word SECTION, an optional priority number, and a period and a space. Priority numbers are explained under *Procedure Division Segmentation* in Chapter 13. A *section name* is a user-defined word with which you identify a section. Because you cannot qualify a section name, you must make it unique. A section ends at one of the following:

- Immediately before the next section header
- At the end of the Procedure Division
- At the key words END DECLARATIVES in the Declaratives portion.

A *paragraph* consists of a paragraph name and zero, one, or more than one successive sentences. A *paragraph name* is a user-defined word followed by a period and a space and identifies a paragraph. Because you can qualify a paragraph name, it need not be unique. A paragraph ends at one of the following:

- Immediately before the next paragraph name or section header
- At the end of the Procedure Division
- At the key words END DECLARATIVES in the Declaratives portion.

If you place one paragraph within a section in a program, you must place all paragraphs in sections.

A *sentence* consists of one or more statements and is ended by a period and a space.

A *statement* is a syntactically valid combination of words (identifiers, figurative constants, and so on) and symbols (literals, relational operators, and so on) beginning with a COBOL verb.

An *identifier* consists of the word or words with which you can make a unique reference to a data item through:

- Qualification
- Subscripting
- Indexing.

In any Procedure Division reference except the class test (see *Class Condition* later in this chapter), if you do not make the contents of an identifier compatible with the class you used in its PICTURE clause, results are unpredictable.

Note: You cannot use a level-88 (condition-name) entry as an identifier because it is not a data item. You can use the associated conditional variable as an identifier.

Procedure Division Organization

The structure of the Procedure Division is shown in the following formats:

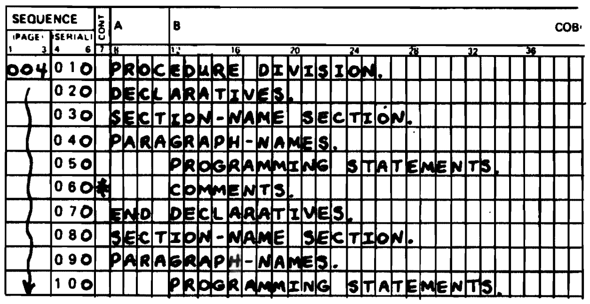
Format 1

```
PROCEDURE DIVISION [USING data-name-1 [, data-name-2] . . .] .

[DECLARATIVES .

{section-name SECTION [segment-number] . declarative-sentence
[paragraph-name. [sentence] . . .] . . .} . . .

END DECLARATIVES .]

{section-name SECTION [segment-number] .
[paragraph-name. [sentence] . . .] . . .} . . .
```

Format 2

```
PROCEDURE DIVISION [USING data-name-1[, data-name-2] . . .] .

{paragraph-name. [sentence] . . .} . . .
```

Example of Statement Sequence in Procedure Division



Sample Procedure Division Statements

```
PROCEDURE DIVISION.
DECLARATIVES.
ERROR-IT SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON INPUT-DATA.
ERROR-ROUTINE.
    IF CHECK-IT = '30' ADD 1 TO DECLARATIVE-ERRORS.
END DECLARATIVES.
BEGIN-NON-DECLARATIVES SECTION.
100-BEGIN-IT.
    OPEN INPUT INPUT-DATA OUTPUT REPORT-OUT.
110-READ-IT.
    READ INPUT-DATA RECORD AT END MOVE 'Y' TO EOF-SW.
    ADD 1 TO RECORDS-IN.
200-MAIN-ROUTINE.
    PERFORM PROCESS-DATA UNTIL EOF-SW = 'Y'.
    PERFORM FINAL-REPORT THRU FINAL-REPORT-EXIT.
    DISPLAY 'TOTAL RECORDS IN = ' RECORDS-IN.
    DISPLAY 'DECLARATIVE ERRORS = >>> ' DECLARATIVE-ERRORS.
    STOP RUN.
PROCESS-DATA.
    IF RECORD-ID = 'G'
        PERFORM PROCESS-GEN-INFO
    ELSE
        IF RECORD-CODE = 'C'
            PERFORM PROCESS-SALES-DATA
        ELSE
            PERFORM UNKNOWN-RECORD-TYPE.
```

Categories of Sentences

There are three categories of sentences: conditional sentences, imperative sentences, and compiler-directing sentences.

A *conditional sentence* is a conditional statement, optionally preceded by an imperative statement, ended by a period and a space.

An *imperative sentence* is an imperative statement, which may consist of a series of imperative statements, ended by a period and a space.

A *compiler-directing* sentence is a single compiler-directing statement, ended by a period and a space.

Categories of Statements

Three categories of statements are used in COBOL: conditional statements, imperative statements, and compiler-directing statements.

A *conditional statement* specifies that the truth value of a condition is to be determined, and that the subsequent action of the object program is dependent on this truth value. Figure 11-1 lists COBOL conditional statements.

An *imperative statement* specifies that an unconditional action is to be taken by the object program. An imperative statement may also consist of a series of imperative statements. Figure 11-2 lists COBOL imperative statements.

A *compiler-directing statement* causes the compiler to take a specific action during compilation. Figure 11-3 lists the COBOL compiler-directing statements.

Categories of Expressions

Two categories of expressions are used in COBOL: arithmetic expressions and conditional expressions.

Decision	IF
Input/Output	DELETE....INVALID KEY READ....AT END READ...INVALID KEY REWRITE...INVALID KEY START...INVALID KEY WRITE...AT END-OF-PAGE WRITE...INVALID KEY
Arithmetic	ADD...ON SIZE ERROR COMPUTE...ON SIZE ERROR DIVIDE...ON SIZE ERROR MULTIPLY...ON SIZE ERROR SUBTRACT...ON SIZE ERROR
Procedure Branching	PERFORM...UNTIL
Data Movement	STRING...ON OVERFLOW UNSTRING...ON OVERFLOW
Table Handling	SEARCH
Ordering	RETURN...AT END
Debug	EXHIBIT...CHANGED

Figure 11-1. Conditional Statements and Their Categories

Arithmetic	ADD[1] COMPUTE[1] DIVIDE[1] INSPECT(TALLYING) MULTIPLY[1] SUBTRACT[1]
Data Movement	ACCEPT (DATE, DAY, TIME) INSPECT (REPLACING) MOVE STRING[3] UNSTRING[3]
Ending	EXIT PROGRAM STOP RUN
Input/Output	ACCEPT (mnemonic) ACQUIRE CLOSE DELETE[2] DISPLAY DROP OPEN READ[4] REWRITE[2] SET[6] START[2] STOP literal WRITE[5]
Ordering	MERGE RELEASE RETURN SORT
Procedure Branching	ALTER CALL EXIT GO PERFORM
Table Handling	SET

Figure 11-2 (Part 1 of 2). Categories of Imperative Statements

---

1 Without the SIZE ERROR option

2 Without the INVALID KEY option

3 Without the ON OVERFLOW option

4 Without the AT END or INVALID KEY options

5 Without the INVALID KEY or END-OF-PAGE options

6 When used to modify external switch values

Subprogram Linkage	CALL
Debug	EXHIBIT READY TRACE RESET TRACE

Figure 11-2 (Part 2 of 2). Categories of Imperative Statements

Library	COPY
Declarative	USE
Documentation	ENTER

Figure 11-3. Categories of Compiler-Directing Statements

Arithmetic Expressions

Arithmetic expressions are used as operands of certain conditional and arithmetic statements. An arithmetic expression can consist of any of the following:

1. An identifier described as a numeric elementary item
2. A numeric literal
3. Identifiers and literals (as defined in items 1 and 2) separated by arithmetic operators
4. Two arithmetic expressions (as defined in item 1, 2, or 3) separated by an arithmetic operator
5. An arithmetic expression (as defined in item 1, 2, 3, or 4) enclosed in parentheses.

You can precede any arithmetic expression by a unary operator.

Identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic can be performed.

Arithmetic Operators

You can use the five binary arithmetic operators and two unary arithmetic operators shown in Figure 11-4 in arithmetic expressions. The arithmetic operators are represented by specific characters that must be preceded and followed by a space.

Binary Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation[7]

[7] Fractional exponents are not allowed

Unary Operator	Meaning
+	Multiplication by +1; retains original sign
-	Multiplication by -1; changes sign

Figure 11-4. Binary and Unary Operators

You can use parentheses in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first. When expressions are contained within a nest of parentheses, evaluation proceeds from the least-inclusive to the most-inclusive set.

When you do not use parentheses, or when parenthesized expressions are at the same level of inclusiveness, the following hierarchical order is implied:

1. Unary operator
2. Exponentiation
3. Multiplication and division
4. Addition and subtraction.

When exponentiation is used as an arithmetic operator, the exponential identifier or literal must be a positive integer value.

Parentheses either eliminate ambiguities in logic in which consecutive operations appear at the same hierarchical level or modify the normal hierarchical sequence of performance when the sequence needs to be modified. When the order of consecutive operations at the same hierarchical level is not completely specified by parentheses, the order is from left to right.

Figure 11-5 shows allowed arithmetic symbol pairs. An arithmetic symbol pair is the appearance of two such symbols in sequence.

An arithmetic expression can begin only with a left parenthesis, a unary operator, or a variable (that is, an identifier or a literal). An arithmetic expression can end only with a right parenthesis or a variable. An arithmetic expression must contain at least one reference to an identifier or a literal. There must be a one-to-one correspondence between left and right parentheses in an arithmetic expression.

First Symbol	Second Symbol				
	Variable (identifier or literal)	*** / ** + -**	**unary + unary -**	**(**	**)**
Variable (identifier or literal)	-	p	-	-	p
* / ** + -	p	-	p	p	-
unary + or unary -	p	-	-	p	-
(	p	-	p	p	-
)	-	p	-	-	p

Figure 11-5. Valid Arithmetic Symbol Pairs

Note: p indicates a permissible pairing

- indicates that the pairing is not permitted

Arithmetic Statements

Arithmetic statements are used for computations. You specify individual operations by using the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements. You can use the COMPUTE statement to symbolically combine these operations in a formula.

Arithmetic Statement Operands

The data description of operands in an arithmetic statement need not be the same. Throughout the calculation, the compiler supplies any necessary data conversion and decimal point alignment.

Size of Operands

The maximum size of each operand is 18 decimal digits. The composite of operands (a hypothetical data item resulting from the superposition of the operands aligned by decimal point) must not contain more than 18 decimal digits.

- For the ADD and SUBTRACT statements, the composite of operands is determined by superimposing all operands in a given statement except those following the word GIVING.

  For example, the items A, B, and C are defined in the Data Division as follows:

  ```
  01  A  PICTURE S9(7)V9(5).

  01  B  PICTURE S9(11)V99.

  01  C  PICTURE S9(12)V9(3).
  ```

  If the statement ADD A, B TO C is run, the composite of operands for this statement consists of 17 decimal digits. It has the following implicit description:

  ```
  Composite-of-Operands PICTURE S9(12)V9(5).
  ```

- For the MULTIPLY statement, the composite of operands is determined by superimposing all receiving data items.

- For the DIVIDE statement, the composite of operands is determined by superimposing all receiving data items except the REMAINDER data item.

- For the COMPUTE statement, the restriction on composite of operands does not apply.

Overlapping Operands

When operands in an arithmetic statement share part of their storage (that is, when the operands overlap), the result when the statement is run is unpredictable.

Multiple Results

When an arithmetic statement has multiple results, the sequence of events is as follows:

1. The statement performs all arithmetic operations to find the result to be placed in the receiving items and stores that result in a temporary location.

2. A sequence of statements transfers or combines the value of this temporary result with each single receiving field. The statements are considered to be written in the same left-to-right order that the multiple results are listed.

For example, running the following statement:

```
ADD A, B, C TO C, D(C), E.
```

is the same as running the following series of statements:

```
ADD A, B, C GIVING TEMP

ADD TEMP TO C

ADD TEMP TO D(C)

ADD TEMP TO E.
```

TEMP is a compiler-supplied temporary result field. When the addition operation for D(C) is performed, the subscript C contains the new value of C.

Note: It is your responsibility, in all arithmetic statements, to define data with enough digits and decimal places to ensure accuracy in the final result.

Intermediate Result Fields

This section discusses the conceptual compiler algorithms for determining the number of integer and decimal places reserved for intermediate results (ir) of arithmetic statements. The following abbreviations are used:

i	Number of integer places carried for an intermediate result
d	Number of decimal places carried for an intermediate result
dmax	In a particular statement, the larger of either: • The number of decimal places needed for the final result field • The maximum number of decimal places defined for any operand except exponents and divisors.
op1	First operand in a generated arithmetic statement
op2	Second operand in a generated arithmetic statement
d1,d2	Number of decimal places defined for op1 or op2, respectively
ir	Intermediate result field obtained from running a generated arithmetic statement or operation. Ir1, ir2, and so on represent successive intermediate results. These intermediate results are generated either in registers or in storage locations. Successive intermediate results may have the same location.

When an arithmetic statement contains only a single pair of operands, no intermediate results are generated. Intermediate results are possible in the following cases:

- In an ADD or a SUBTRACT statement containing multiple operands immediately following the verb
- In a COMPUTE statement specifying a series of arithmetic operations
- In arithmetic expressions contained in an IF or a PERFORM statement
- In the GIVING phrase with multiple result fields for ADD, SUBTRACT, MULTIPLY, DIVIDE
- In a COMPUTE statement specifying multiple result fields.

In such cases, the compiler treats the statement as a succession of operations. For example, the following statement:

```
COMPUTE Y = A + B * C - D / E + F ** G
```

is replaced by

F**G		yielding ir1
MULTIPLY B	BY C	yielding ir2
DIVIDE E	INTO D	yielding ir3
ADD A	TO ir2	yielding ir4
SUBTRACT ir3	FROM ir4	yielding ir5
ADD ir5	TO ir1	yielding Y

Compiler Calculation of Intermediate Results

The number of integer places in an ir is calculated as follows:

- The compiler first determines the maximum value that the ir can contain by performing the statement in which the ir occurs.
  - If an operand in this statement is a data name, the value used for the data name is equal to the numeric value of the PICTURE character string for the data name (for example, PICTURE 9V99 has the value 9.99).
  - If an operand is a literal, the actual value of the literal is used.
  - If an operand is an intermediate result, the value determined for the intermediate result in a previous arithmetic operation is used.
  - If the operation is division:
    - a) If op2 is a data name, the value used for op2 is the minimum nonzero value of the digit in the PICTURE character string for the data name (for example, PICTURE 9V99 has the value 0.01).
    - b) If op2 is an intermediate result, the intermediate result is treated as though it had a PICTURE character string, and the minimum nonzero value of the digits in this PICTURE character string is used.

- When the maximum value of the ir is determined by the above procedures, i is set equal to the number of integers in the maximum value.

- The number of decimal places contained in an ir is calculated as:

Operation	Decimal Places
+ or -	d1 or d2, whichever is greater
*	d1 + d2
/	d1 - d2 or dmax, whichever is greater[8]
**	dmax if op2 is a data name; d1 * op2 if op2 is an integral literal

Note: You must define the operands of any arithmetic statement with enough decimal places to give the desired accuracy in the final result.

The following chart indicates the action of the compiler when handling intermediate results:

Value of i + d	Value of d	Value of i + dmax	Action Taken
< 19 = 19	Any value	Any value	i integer and d decimal places are carried for ir
> 19	< dmax	Any value	19 - d integer and d decimal places are carried for ir
	= dmax		
	> dmax	< 19	i integer and 19 - i decimal places are carried for ir
		= 19	
		> 19	19 - dmax integer and dmax decimal places are carried for ir

[8] After a division operation in a COMPUTE statement with a ROUNDED option, the number of decimal places carried in the intermediate result field is increased by 1.

Data Manipulation Statements

Movement and inspection of data are the functions of the following COBOL statements: INSPECT, MOVE, STRING, and UNSTRING.

When the sending and receiving fields of a data manipulation statement share a part of their storage (that is, when the operands overlap), the results when the statement is run are unpredictable.

Procedure Branching Statements

Statements, sentences, and paragraphs in the Procedure Division are usually run sequentially. The procedure branching statements allow alterations in the sequence. The procedure branching statements are ALTER, EXIT, GO TO, PERFORM, and STOP.

Compiler-Directing Statements

Compiler-directing statements provide instructions to the COBOL compiler. The compiler-directing statements are COPY, ENTER, and USE.

Only the ENTER statement and the USE AFTER EXCEPTION/ERROR statement are discussed in this chapter. The COPY statement is discussed under *Using the Library Copy Facility* in Chapter 4. The USE FOR DEBUGGING statement is discussed under *Debugging Features* in Chapter 6.

Conditional Expressions

A conditional expression causes the object program to select alternative paths of control, depending on the truth value of a test. Conditional expressions can be specified in IF, PERFORM, and SEARCH statements. The IF and PERFORM statements are discussed in this chapter. The SEARCH statement is discussed in Chapter 13.

A conditional expression can be specified in simple conditions and in complex conditions. You can enclose both simple and complex conditions within any number of paired parentheses; parentheses do not change the category of the condition.

Simple Conditions

There are five simple conditions:

- Class condition
- Condition-name condition
- Relation condition
- Sign condition
- Switch-status condition.

A simple condition has a truth value of true or false. When a simple condition is enclosed in paired parentheses, its truth value is not changed.

Class Condition

The class condition determines whether a data item is numeric or alphabetic.

Format

The identifier you are testing must be described implicitly or explicitly as USAGE DISPLAY. The identifier is determined to be numeric only if the contents consist of any combination of the digits 0 through 9, with or without an operational sign.

If the PICTURE character string of the identifier you are testing does not contain an operational sign, the identifier is determined to be numeric only if the contents are numeric and an operational sign is not present.

If the PICTURE character string of the identifier you are testing does contain an operational sign, the identifier is determined to be numeric only if the item is an elementary item, the contents are numeric, and a valid operational sign is present.

In the EBCDIC collating sequence, valid embedded operational signs are hexadecimal F and hexadecimal D. For items described with the SIGN IS SEPARATE clause, valid operational signs are + (hexadecimal 4E) and - (hexadecimal 60).

You cannot use the NUMERIC test with an identifier described either as alphabetic or as a group item that contains one or more signed elementary items. The identifier being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters A through Z and the space.

You cannot use the ALPHABETIC test with an identifier described as numeric.

Figure 11-6 shows valid forms of the class test.

Type of Identifier	Valid Forms of the Class Test
Alphabetic	ALPHABETIC NOT ALPHABETIC
Alphanumeric	ALPHABETIC NOT ALPHABETIC NUMERIC NOT NUMERIC
Zoned Decimal	NUMERIC NOT NUMERIC

Figure 11-6. Valid Forms of the Class Test

Condition-Name Condition

A condition-name condition causes a conditional variable to be tested to determine whether its value is equal to any of the values associated with the condition name (level-88 item).

Format

```
condition-name
```

In conditions, you can use a condition name as an abbreviation for the relation condition, because the specified condition name is equal to only one of the values or ranges of values assigned to the specified conditional variable. The result of the test is true if one of the values corresponding to the condition name equals the current value of the associated conditional variable.

If you associate the condition name with a range of values or with several ranges of values, the conditional variable is tested to determine whether or not its value falls within the range(s), including the end values. The result of the test is true if

one of the values corresponding to the condition name equals the value of its associated conditional variable.

The following example shows the usage of condition names and conditional variables:

```
01  GRADE-ID PIC 99.
    88 PRIMARY-OTHER   VALUE 1 THRU 3, 5, 6.
    88 PRIMARY-FOUR    VALUE 4.
    88 JUNIOR-HI       VALUE 7 THROUGH 9.
    88 SENIOR-HI       VALUE 10 THROUGH 12.
```

GRADE-ID is the conditional variable; PRIMARY-OTHER, PRIMARY-FOUR, JUNIOR-HI, and SENIOR-HI are condition names. For individual records in the file, only one of the values specified in the condition name entries can be present. To determine the grade level of a specific record, you can code any of the following:

```
IF PRIMARY-OTHER...
```

(which tests for values 1, 2, 3, 5, 6)

```
IF PRIMARY-FOUR...
```

(which tests for value 4)

```
IF JUNIOR-HI...
```

(which tests for values 7, 8, 9)

```
IF SENIOR-HI...
```

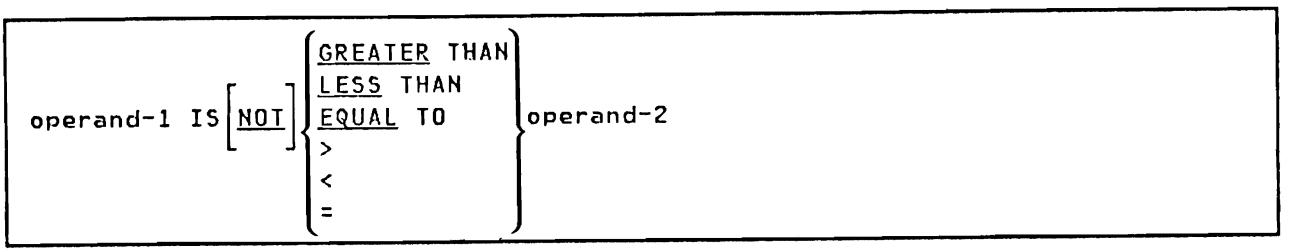
(which tests for values 10, 11, 12)

Depending on the evaluation of the condition-name condition, alternative paths are taken by the object program.

Relation Condition

A relation condition causes a comparison between two operands, either of which may be an identifier, a literal, or an arithmetic expression.

Format



Operand-1 is the subject of the relation condition; operand-2 is the object of the relation condition. Operand-1 and operand-2 may each be an identifier, a literal, or an arithmetic expression. The relation condition must contain at least one reference to an identifier. Except when two numeric operands are compared, operand-1 and operand-2 must have the same USAGE clause specified.

The relational operator specifies the type of comparison to be made. Figure 11-7 shows relational operators and their meanings. Each relational operator must be preceded and followed by a space.

Relational Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] >	Greater than or not greater than
IS [NOT] LESS THAN IS [NOT] <	Less than or not less than
IS [NOT] EQUAL TO IS [NOT] =	Equal to or not equal to

Figure 11-7. Relational Operators and Their Meanings

IBM Extension

Boolean Considerations: The valid types of relation conditions that can be used with Boolean data items are EQUAL TO and NOT EQUAL TO.

End of IBM Extension

Rules for numeric and nonnumeric comparisons are given in the paragraph following Figure 11-8. If either of the operands is a group item, nonnumeric comparison rules apply. Figure 11-8 summarizes the permissible comparisons.

First Operand	Second Operand													
	GR	AL	AN	ANE	NE	FC[1] NNL	ZR NL	ZD	BI	PD	AE	BO	IN	IDI
Group (GR)	NN	NN	NN	NN	NN	NN	NN	NN						
Alphabetic (AL)	NN	NN	NN	NN	NN	NN	NN	NN						
Alphanumeric (AN)	NN	NN	NN	NN	NN	NN	NN	NN						
Alphanumeric edited (ANE)	NN	NN	NN	NN	NN	NN	NN	NN						
Numeric edited (NE)	NN	NN	NN	NN	NN	NN	NN	NN						
Figurative constant (FC)[1] and nonnumeric literal (NNL)	NN	NN	NN	NN	NN			NN						
Figurative constant ZERO (ZR) and numeric literal (NL)	NN	NN	NN	NN	NN			NU	NU	NU	NU		IO[2]	
Zoned decimal (ZD)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU		IO[2]	
Binary (BI)							NU	NU	NU	NU	NU		IO[2]	
Packed decimal (PD)							NU	NU	NU	NU	NU		IO[2]	
Arithmetic expression (AE)							NU	NU	NU	NU	NU			
Boolean data item (BO) or Boolean literal												BO		
Index name (IN)							IO[2]	IO[2]	IO[2]	IO[2]			IO	IV
Index data item (IDI)													IV	IV

BO = Comparison as described for Boolean operands.

NN = Comparison as described for nonnumeric operands.

NU = Comparison as described for numeric operands.

IO = Comparison as described for two index names or index data items.

IV = Comparison as described for index data items.

[1]FC includes all figurative constants except ZERO.
[2]Valid only if the numeric item is an integer.

Figure 11-8. Permissible Comparisons of Operands

Comparison of Numeric Operands: For numeric class operands, algebraic values are compared. The length (number of digits) of the operands is not significant. Zero is considered a unique value, regardless of the sign; unsigned numeric operands are considered positive. Regardless of what you specified in their USAGE clause, comparison of numeric operands is permitted.

Comparison of Nonnumeric Operands: A comparison of two nonnumeric operands or of one numeric and one nonnumeric operand is made with respect to the binary collating sequence of the character set in use.

Comparison of Numeric and Nonnumeric Operands: When you compare a nonnumeric and a numeric operand, the following rules apply:

- If the nonnumeric operand is a literal or an elementary data item, the numeric operand is treated as though it were moved to an alphanumeric elementary data item of the same size. The contents of this alphanumeric data item is then compared with the nonnumeric operand.

- If the nonnumeric operand is a group item, the numeric operand is treated as though it were moved to a group item of the same size. The contents of this group item is then compared with the nonnumeric operand. For further discussion of the rules for alphanumeric and group move operations, see the *MOVE Statement* later in this chapter.

Numeric and nonnumeric operands can be compared only when their USAGE, explicitly or implicitly, is the same. In such comparisons, you should describe the numeric operand as an integer literal or data item; noninteger literals and data items should not be compared with nonnumeric operands.

The size of each operand is the total number of characters in that operand; the size affects the result of the comparison. There are two kinds of operands to consider: operands of equal size and operands of unequal size.

Operands of Equal Size: Characters in corresponding positions of the two operands are compared, beginning with the leftmost character and continuing through the rightmost character.

If all pairs of characters through the last pair test as equal, the operands are considered equal. If a pair of unequal characters is encountered, the characters are tested to determine their relative positions in the collating sequence. The operand containing the character higher in the sequence is considered the greater operand.

Operands of Unequal Size: If the operands are of unequal size, the comparison is made as though the shorter operand were extended to the right with enough spaces to make the operands equal in size.

Note: Valid comparisons for index names and index data items are discussed under *Using Table Handling Facilities* in Chapter 13.

Sign Condition

The sign condition determines whether or not the algebraic value of a numeric operand is greater than, less than, or equal to 0.

Format

```
operand IS [NOT] { POSITIVE }
                 { NEGATIVE }
                 { ZERO     }
```

You must define the operand being tested as a numeric identifier or as an arithmetic expression that contains at least one reference to an identifier.

The operand is POSITIVE if its value is greater than 0, NEGATIVE if its value is less than 0, and ZERO if its value is equal to 0. An unsigned operand is POSITIVE or ZERO.

When you specify NOT, one algebraic test is run for the truth value of the sign condition. For example, NOT ZERO is regarded as true when the operand tested is positive or negative in value.

Switch-Status Condition

The switch-status condition determines the on or off status of an UPSI switch.

Format

```
condition-name
```

The condition name must be defined to be associated with the ON or OFF value of a switch in the SPECIAL-NAMES paragraph.

The switch-status condition tests the value associated with the condition name. The result of the test is true if the UPSI switch is set to the position corresponding to the condition name.

Complex Conditions

A complex condition is a condition in which one or more logical operators act upon one or more conditions. Complex conditions include:

- Negated simple conditions
- Combined conditions
- Negated combined conditions.

Each logical operator must be preceded and followed by a space. The logical operators and their meanings are shown in Figure 11-9.

Logical Operator	Meaning
AND	Logical conjunction-the truth value is true when both conditions are true.
OR	Logical inclusive OR-the truth value is true when either or both conditions are true.
NOT	Logical negation-reversal of truth value (the truth value is true if the condition is false).

Figure 11-9. Logical Operators and Their Meanings

Negated Simple Conditions

A simple condition is negated through the use of the logical operator NOT.

Format

```
NOT simple-condition
```

The simple condition you are negating can be a:

- Class condition
- Condition-name condition
- Relation condition
- Sign condition
- Switch-status condition.

The simple condition cannot be negated if the condition itself contains a NOT.

The negated simple condition gives the opposite truth value as the simple condition. For example, if the truth value of the simple condition is true, then the truth value of that same negated simple condition is false.

Placing a negated simple condition within parentheses does not change its truth value. For example, the following two statements are equivalent:

NOT A IS EQUAL TO B.

NOT (A IS EQUAL TO B).

Combined Conditions

Two or more conditions can be logically connected to form a combined condition.

Format

The condition you are combining can be a:

- Simple condition
- Negated simple condition
- Combined condition
- Negated combined condition (the NOT logical operator followed by a combined condition enclosed in parentheses).

Combinations of these conditions are specified according to the rules given in Figure 11-10.

You never need parentheses when either AND or OR (but not both) are used exclusively in one combined condition; however, parentheses may be needed to find a final truth value when you use a combination of AND, OR, and NOT. There must be a one-to-one correspondence between left and right parentheses with each left parenthesis to the left of its corresponding right parenthesis.

Figure 11-10 summarizes the way in which conditions and logical operators can be combined and put in parentheses. Figure 11-11 illustrates the relationships between logical operators and conditions C1 and C2 in which C1 and C2 are any of the preceding conditions.

Condition Element	Permissible Position in Conditional Expressions			
	Leftmost	When Not Leftmost, May Be Immediately Preceded By:	When Not Rightmost, May Be Immediately Followed By:	Rightmost
Simple condition	Yes	OR NOT AND (	OR AND)	Yes
OR AND	No	Simple condition)	Simple-condition NOT (	No
NOT	Yes	OR AND (	Simple-condition (	No
(	Yes	OR NOT AND (	Simple-condition NOT (	No
)	No	Simple-condition)	OR AND)	Yes

Figure 11-10. Valid Combinations of Conditions, Logical Operators, and Parenthesis in a Conditional Expression

Value for C1	Value for C2	C1 AND C2	C1 OR C2	NOT (C1 AND C2)	NOT C1 AND C2	NOT (C1 OR C2)	NOT C1 OR C2
True	True	True	True	False	False	False	True
False	True	False	True	True	True	False	True
True	False	False	True	True	False	False	False
False	False	False	False	True	False	True	True

Figure 11-11. How Logical Operators Affect the Evaluation of Conditions

The truth value of a complex condition depends on the truth values of the simple conditions and negated simple conditions that make up the complex condition. The logical operators tell the compiler how to combine these individual truth values.

Evaluating Conditional Expressions: If you use parentheses, logical evaluation of combined conditions proceed in the following order:

1. Conditions within parentheses are evaluated first.

2. Within nested parentheses, evaluation proceeds from the least-inclusive condition to the most-inclusive condition.

If you do not use parentheses or they are not at the same level of inclusiveness, the combined condition is evaluated in the following order:

1. Arithmetic expressions.

2. Simple conditions, in the following order:

   a. Relation
   b. Class
   c. Condition name
   d. Switch status
   e. Sign

3. Negated simple conditions in the same order as item 2.

4. Combined conditions, in the following order:

   a. AND
   b. OR

5. Negated combined conditions, in the following order:

   a. AND
   b. OR

6. Consecutive operands at the same evaluation-order level. These are evaluated from left to right.

For example, the expression

```
A IS NOT GREATER THAN B
OR A + B IS EQUAL TO C AND
D IS POSITIVE
```

is evaluated as if it were enclosed in parentheses as follows:

(A IS NOT GREATER THAN B) OR (((A+B) IS EQUAL TO C) AND (D IS POSITIVE)).

The order of evaluation is as follows:

1. (A IS NOT GREATER THAN B) is evaluated, giving some intermediate truth value; for example, t1.

2. (A + B) is evaluated, giving some intermediate result; for example, x.

3. (x IS EQUAL TO C) is evaluated, giving some intermediate truth value; for example, t2.

4. (D IS POSITIVE) is evaluated, giving some intermediate truth value; for example, t3.

5. (t2 AND t3) is evaluated, giving some intermediate truth value; for example, t4.

6. (t1 OR t4) is evaluated, giving the final truth value, which is the result of the expression.

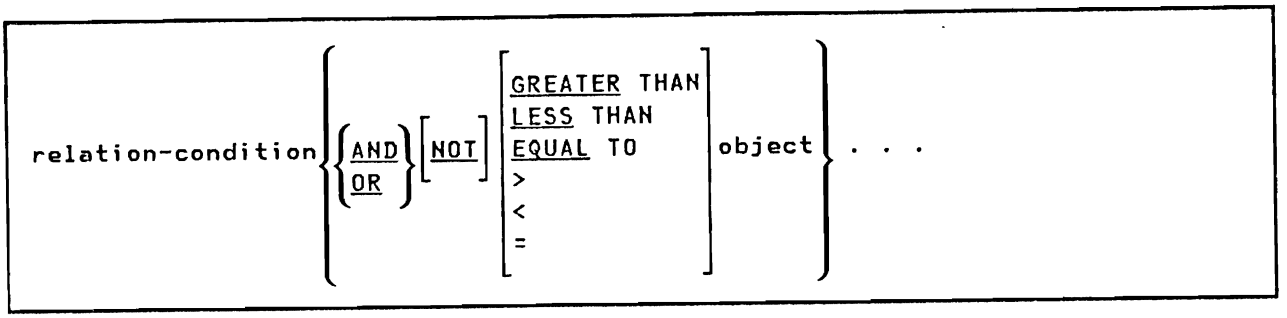
Note: Every condition in the expression is always evaluated before a final truth value is determined. You must ensure that any subscripted or indexed data items stay within the boundaries described in the table.

Abbreviated Combined Relation Conditions

When you write relation conditions consecutively, and no parentheses are used within the consecutive sequence, any relation condition after the first can be abbreviated by either:

- Omitting the subject
- Omitting the subject and the relational operator.

Format



The resulting combined condition must comply with the rules for element sequence in combined conditions, as shown in Figure 11-10.

In any consecutive sequence of relation conditions, you can specify both forms of abbreviation. The abbreviated condition is evaluated as if:

- The last stated subject is the missing subject.
- The last stated relational operator is the missing relational operator.
- The word NOT is part of the relational operator in the forms NOT GREATER THAN, NOT >, NOT LESS THAN, NOT <, NOT EQUAL TO, and NOT =.
- NOT in any other position is a logical operator and, thus, results in a negated relation condition.

Figure 11-12 shows examples of abbreviated combined relation conditions and their nonabbreviated equivalent meanings.

Abbreviated Combined Relation – Condition	Nonabbreviated Equivalent Meaning
A = B AND NOT LESS THAN C OR D	((A = B) AND (A NOT LESS THAN C)) OR (A NOT LESS THAN D)
A NOT GREATER THAN B OR C	(A NOT GREATER THAN B) OR (A NOT GREATER THAN C)
NOT A = B OR C	(NOT (A = B) OR (A = C))
NOT (A = B OR LESS THAN C)	NOT ((A = B) OR (A LESS THAN C))
NOT (A NOT = B AND C AND NOT D)	NOT (((A NOT = B) AND (A NOT = C)) AND (NOT (A NOT = D)))

Figure 11-12. Abbreviated Combined Relation-Condition and Equivalent Meanings

Declaratives

The Declaratives section provides a method of invoking procedures that are run when an exceptional condition occurs that you cannot normally test. Declarative procedures are provided for processing exceptional input/output conditions and for debugging procedures.

Format

```
PROCEDURE DIVISION [USING data-name-1 [, data-name-2] . . . ] .

[DECLARATIVES .

{section-name SECTION [segment-number] . declarative-sentence

[paragraph-name. [sentence] . . .] . . .}. . .

END DECLARATIVES .]
```

You write declarative procedures at the beginning of the Procedure Division in a series of Declarative sections. You precede each such section by a USE sentence that identifies the function of this section. The series of procedures that follow specify what actions are to be taken when the exceptional condition occurs. Each Declarative section ends with either another section name followed by a USE sentence or with the key words END DECLARATIVES.

You precede the entire group of Declarative procedures by the key word DECLARATIVES, written on the line after the Procedure Division header; the group is followed by the key words, END DECLARATIVES. The key words DECLARATIVES and END DECLARATIVES must each begin in area A and be followed by a period. You cannot have any other text on the same line.

In the Declaratives portion of the Procedure Division, you must follow each section header (with an optional segment number) with a period and a space; this is followed by a USE sentence with a period and a space. No other text can appear on the same line. There are two forms of the USE sentence:

- USE AFTER EXCEPTION/ERROR
- USE FOR DEBUGGING.

The USE sentence itself is never run; instead, the USE sentence defines the conditions that cause the immediately following procedural paragraphs to be run. These paragraphs specify the actions to be taken. After the procedure is run, control is returned to the routine that activated it.

Within a Declarative procedure, you must not reference any nondeclarative procedure, except for the USE statement.

Within a Declarative procedure, no statement can be run that would cause the running of a USE procedure that has been previously invoked and has not yet returned control to the invoking routine.

A Declarative procedure ends when the last statement in the procedure has run.

In this chapter, only the USE AFTER EXCEPTION/ERROR procedure is described. The USE FOR DEBUGGING procedure is described under *Debugging Features* in Chapter 13.

Procedure Division Statements (Except Input/Output Statements)

The remainder of this chapter discusses the various statements used in the Procedure Division. The input and output statements are discussed in Chapter 12.

ADD Statement

The ADD statement causes two or more numeric operands to be added and the result to be stored. The formats of the ADD statement are as follows:

Format 1

```
ADD {identifier-1} [, identifier-2] . . . TO identifier-m [ROUNDED]
    {literal-1   } [, literal-2   ]

[, identifier-n [ROUNDED]] . . .[ON SIZE ERROR imperative-statement]
```

Format 2

```
ADD {identifer-1}, {identifier-2} [, identifier-3] . . .
    {literal-1  }  {literal-2  }  [, literal-3  ]

     GIVING identifier-m [ROUNDED] [, identifier-n [ROUNDED]] . . .

    [ON SIZE ERROR imperative-statement]
```

Format 3

```
ADD {CORRESPONDING} identifier-1 TO identifier-2 [ROUNDED]
    {CORR         }

    [ON SIZE ERROR imperative-statement]
```

In formats 1 and 2, each identifier, except those following the key word GIVING, must name an elementary numeric item. In format 2, each identifier following the key word GIVING must name an elementary numeric or numeric edited item. In format 3, each identifier must name a group item. In all formats, each literal must be a numeric literal.

In format 1, all identifiers or literals preceding the key word TO are added together, and this sum is added to and stored immediately in identifier-m. If you specify as such, the sum is then added to and stored immediately in identifier-n, and so on.

In format 2, at least two operands must precede the key word GIVING. The values of these operands are added together, and the sum is stored as the new value of identifier-m and, if specified, identifier-n, and so on.

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

If the composite of the operands is 18 digits or less, the compiler ensures that enough places are carried so that no significant digits are lost while the statement is run.

ROUNDED Phrase

After decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is compared with the number of places provided for the fraction of the resultant identifier.

If the size of the fractional result exceeds the number of places provided for its storage, truncation occurs unless the ROUNDED phrase is specified. When the ROUNDED phrase is specified, the least-significant digit of the resultant identifier has its value increased by 1 whenever the most-significant digit of the excess is greater than or equal to 5.

When the resultant identifier is described by a PICTURE clause containing rightmost Ps, and when the number of places in the calculated result exceeds the number of integer positions specified, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

SIZE ERROR Phrase

A size error condition exists if, after decimal point alignment, the value of a result exceeds the largest value that can be contained in the resultant field. In the ADD statement, the size error condition applies only to final results.

If you specify the ROUNDED phrase, rounding takes place before size error checking.

When a size error occurs, the subsequent action of the program depends on whether or not the SIZE ERROR phrase is specified.

If you do not specify the SIZE ERROR phrase, and a size error condition occurs, the value of the affected resultant identifier is unpredictable. When you specify multiple receivers, those that do not have a size error are not affected by receivers that do have the error.

If you specify the SIZE ERROR phrase and a size error condition occurs, the error results are not placed in the receiving identifier. When the arithmetic operation is completed, the imperative statement in the SIZE ERROR phrase is run.

If an individual arithmetic operation causes a size error condition for an ADD CORRESPONDING statement, the SIZE ERROR imperative statement is not run until all individual additions or subtractions have been completed.

GIVING Phrase

If you specify the GIVING phrase, the value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation. Because this identifier is not involved in the computation, it can be a numeric edited item.

CORRESPONDING Phrase

The CORRESPONDING phrase lets operations be performed on elementary items of the same name. You simply specify the group items to which the elementary items belong. The results are the same as if each pair of CORRESPONDING identifiers had been referred to in a separate MOVE statement.

Both identifiers following the key word CORRESPONDING must name group items. In this discussion, these identifiers are referred to as d1 and d2.

A pair of subordinate data items, one from d1 and one from d2, correspond if the following conditions are true:

- At least one of the subordinate items is elementary.
- The two subordinate items have the same name and the same qualifiers up to but not including d1 and d2.
- The subordinate items are not identified by the key word FILLER.
- The subordinate items do not include a REDEFINES, a RENAMES, an OCCURS, or a USAGE IS INDEX clause in their descriptions; if such a subordinate item is a group, the items subordinate to it are also ignored. However, d1 and d2 themselves can contain or be subordinate to items containing a REDEFINES or OCCURS clause in their descriptions.

  For example, two data hierarchies are defined as follows:

```
05  ITEM-1 OCCURS 6 INDEXED BY X.
    10 ITEM-A ...
    10 ITEM-B ...
    10 ITEM-C REDEFINES ITEM-B ...
05  ITEM-2
    10 ITEM-A ...
    10 ITEM-B ...
    10 ITEM-C ...
```

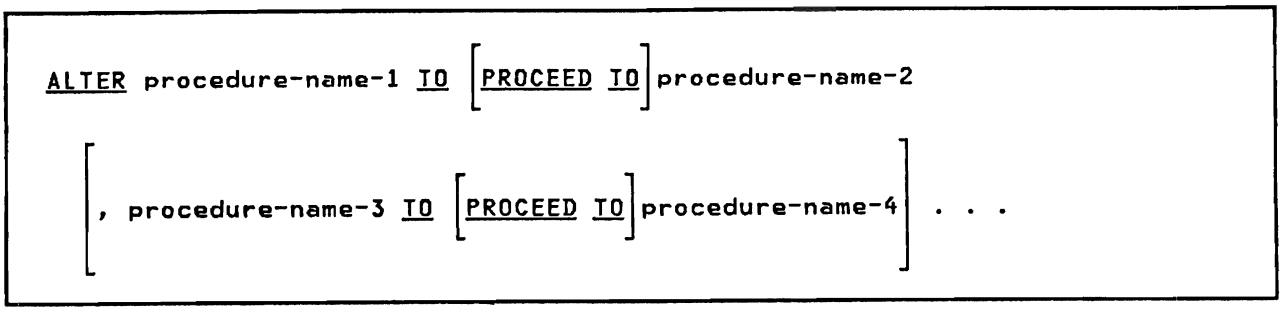
  If you specify ADD CORR ITEM-2 TO ITEM-1(X), ITEM-A and ITEM-A(X) and ITEM-B and ITEM-B(X) are considered to be corresponding. Thus, ITEM-A and ITEM-B of ITEM-2 are moved to ITEM-1(X). ITEM-C and ITEM-C(X) are not included, because ITEM-C(X) includes a REDEFINES clause in its data description.ITEM-1 is valid as either d1 or d2.

- Neither d1 nor d2 is described as a level-66, -77, or -88 item or as a FILLER or USAGE IS INDEX item.

ALTER Statement

The ALTER statement changes the transfer point specified in a GO TO statement.

Format



procedure-name-1, procedure-name-3, and so on, must each name a Procedure Division paragraph that contains only one sentence. That sentence must be a GO TO statement without the DEPENDING ON phrase.

procedure-name-2, procedure-name-4, and so on, must each name a Procedure Division section or paragraph.

When the ALTER statement is run, it modifies the GO TO statement in the paragraph named by procedure-name-1, procedure-name-3, and so on. Subsequent runs of the modified GO TO statement(s) cause control to be transferred to procedure-name-2 and, if specified, procedure-name-4, and so on. For example:

```
PARAGRAPH-1.
  GO TO BYPASS-PARAGRAPH.
PARAGRAPH-1A.
  .
  .
BYPASS-PARAGRAPH.
  .
  .
  ALTER PARAGRAPH-1 TO PROCEED TO
      PARAGRAPH-2.
  .
  .
PARAGRAPH-2.
  .
  .
```

Before the ALTER statement is run, when control reaches PARAGRAPH-1, the GO TO statement transfers control to BYPASS-PARAGRAPH. After the ALTER statement is performed, however, the next time control reaches PARAGRAPH-1, the GO TO statement transfers control to PARAGRAPH-2.

Note: The ALTER statement acts as a program switch, allowing, for example, one run sequence during initialization and another sequence during the bulk of file processing. Because altered GO TO statements are difficult to debug, it is preferable to test a switch and, based on the value of the switch, run a particular code sequence.

Segmentation Information

A GO TO statement in a section whose priority is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different priority. All other uses of the ALTER statement are valid and are performed.

Modified GO TO statements in independent segments may sometimes be returned to their initial states. For further discussion, see *Procedure Division Segmentation* in Chapter 13.

COMPUTE Statement

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

Format

```
COMPUTE identifier-1 [ROUNDED] [, identifier-2 [ROUNDED]] . . .

    = arithmetic-expression [ON SIZE ERROR imperative-statement]
```

The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on receiving data items imposed by the rules for the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements.

The identifiers that appear to the left of the equal sign must name either elementary numeric items or elementary numeric edited items.

When the COMPUTE statement is run, the value of the arithmetic expression is calculated; this value is then stored as the new value of identifier-1, identifier-2, and so on, in turn.

The arithmetic expression can be any logical combination of identifiers, numeric literals, and arithmetic operators.

An arithmetic expression consisting of a single identifier or literal that lets you set identifier-1, and so on, equal to the value of that identifier or literal.

ROUNDED Phrase

After decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is compared with the number of places provided for the fraction of the resultant identifier.

If the size of the fractional result exceeds the number of places provided for its storage, truncation occurs unless the ROUNDED phrase is specified. When the ROUNDED phrase is specified, the least-significant digit of the resultant identifier has its value increased by 1 whenever the most-significant digit of the excess is greater than or equal to 5.

When the resultant identifier is described by a PICTURE clause containing rightmost Ps, and when the number of places in the calculated result exceeds the number of integer positions specified, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

SIZE ERROR Phrase

A size error condition exists if, after decimal point alignment, the value of a result exceeds the largest value that can be contained in the resultant field. Division by 0, as well as 0 raised to the zero power, always causes a size error condition. In the COMPUTE statement, the size error condition applies only to final results.

If you specify the ROUNDED phrase, rounding takes place before size error checking.

When a size error occurs, the subsequent action of the program depends on whether or not the SIZE ERROR phrase is specified.

If you do not specify the SIZE ERROR phrase and a size error condition occurs, the value of the affected resultant identifier is unpredictable. When you specify multiple receivers, those that do not have a size error are not affected by receivers that do have the error.

If you specify the SIZE ERROR phrase and a size error condition occurs, the error results are not placed in the receiving identifier. When the arithmetic operation is completed, the imperative statement in the SIZE ERROR phrase is run.

If an individual arithmetic operation causes a size error condition for ADD CORRESPONDING and SUBTRACT CORRESPONDING statements, the SIZE ERROR imperative statement is not run until all of the individual additions or subtractions have been completed.

Note: When arithmetic operations must be combined, the COMPUTE statement is more efficient than a series of separate arithmetic statements.

DIVIDE Statement

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient and remainder. The formats of the DIVIDE statement are:

Format 1

```
DIVIDE {identifier-1} INTO identifier-2 [ROUNDED]
       {literal-1   }

    [ , identifier-3 [ROUNDED] ] . . .

   [ON SIZE ERROR imperative-statement]
```

Format 2

```
DIVIDE {identifier-1} {INTO} {identifier-2} GIVING identifier-3 [ROUNDED]
       {literal-1   } {BY  } {literal-2   }

    [ , identifier-4 [ROUNDED] ] . . . [ON SIZE ERROR imperative-statement]
```

Format 3

```
DIVIDE {identifier-1} {INTO} {identifier-2} GIVING identifier-3 [ROUNDED]
       {literal-1   } {BY  } {literal-2   }

   REMAINDER identifier-4 [ON SIZE ERROR imperative-statement]
```

Each identifier except those following the key words GIVING and REMAINDER must name an elementary numeric item. Each identifier following the key words GIVING and REMAINDER must name an elementary numeric or numeric edited item. Each literal must be a numeric literal.

In format 1, the value of literal-1 or identifier-1 is divided into the value of identifier-2; the quotient is then placed in identifier-2. If you specify identifier-3, the value of literal-1 or identifier-1 is divided into identifier-3; the quotient is then placed in identifier-3, and so on.

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 quotient is stored in identifier-3 and, if specified, identifier-4, and so on.

In format 3, the value of identifier-1 or literal-1 is divided into or by identifier-2 or literal-2. The value of the quotient is stored in identifier-3, and the value of the remainder is stored in identifier-4.

The remainder is defined as the result of subtracting the product of the quotient and the divisor from the dividend. If identifier-3 (the quotient) is a numeric edited field, the quotient used to calculate the remainder is an intermediate field that contains the unedited quotient.

ROUNDED Phrase

After decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is compared with the number of places provided for the fraction of the resultant identifier.

If the size of the fractional result exceeds the number of places provided for its storage, truncation occurs unless the ROUNDED phrase is specified. When the ROUNDED phrase is specified, the least-significant digit of the resultant identifier has its value increased by 1 whenever the most-significant digit of the excess is greater than or equal to 5.

When the resultant identifier is described by a PICTURE clause containing rightmost Ps, and when the number of places in the calculated result exceeds the number of integer positions specified, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

When you specify the ROUNDED phrase in format 3, the quotient used to calculate the remainder is an intermediate field that contains the quotient truncated rather than rounded.

SIZE ERROR Phrase

A size error condition exists if, after decimal point alignment, the value of a result exceeds the largest value that can be contained in the resultant field. Division by 0 always causes a size error condition. In the DIVIDE statement, the size error condition applies only to final results.

If you specify the ROUNDED phrase, rounding takes place before size error checking.

When a size error occurs, the subsequent action of the program depends on whether or not the SIZE ERROR phrase is specified.

If you do not specify the SIZE ERROR phrase and a size error condition occurs, the value of the affected resultant identifier is unpredictable. When you specify multiple receivers, those that do not have a size error are not affected by receivers that do have the error.

If you specify the SIZE ERROR phrase and a size error condition occurs, the error results are not placed in the receiving identifier. When the arithmetic operation is completed, the imperative statement in the SIZE ERROR phrase is run.

When the SIZE ERROR phrase is used in format 3, the following considerations apply:

- When the size error conditions occur on the quotient, no remainder calculation is meaningful. The contents of the quotient field (identifier-3) and the remainder field (identifier-4) are unchanged.

- When the size error occurs on the remainder, the contents of the remainder field (identifier-4) are unchanged.

Note: In these two cases, you must analyze the results to determine which situation has actually occurred.

GIVING Phrase

If you specify the GIVING phrase, the value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation. Because this identifier is not involved in the computation, it can be a numeric edited item.

ENTER Statement

Because the System/36 COBOL compiler does not allow another source language to be used in COBOL source programs, the ENTER statement is not required or used by the System/36 COBOL compiler.

Format

```
ENTER language-name [routine-name].
```

If the ENTER statement is inserted in the source program, it is treated as a comment. Statements in the language named in the ENTER statement must not be included in the source program.

EXIT Statement

The EXIT statement provides a common end point for a series of procedures.

Format

```
EXIT [PROGRAM].
```

The EXIT statement lets you assign a procedure name at a given point in a program. It has no other effect on the compilation or running of the program.

You must place the EXIT statement in a sentence by itself, and this sentence must be the only sentence in the paragraph.

The EXIT PROGRAM statement is discussed under *Subprogram Linkage* in Chapter 13.

Note: The EXIT statement is useful for documenting the end point in a series of procedures. If an exit paragraph is written as the last paragraph in a Declarative procedure or a series of performed procedures, it identifies the point at which control is to be transferred. When control reaches such an exit paragraph and the associated Declarative or PERFORM statement is active, control is transferred to the appropriate part of the Procedure Division. When control reaches such an exit paragraph and no associated PERFORM statement or Declarative procedure is active, control passes through the EXIT statement to the first statement of the next paragraph.

GO TO Statement

The GO TO statement transfers control from one part of the Procedure Division to another. The formats of the GO TO statement are as follows:

Format 1

```
GO TO [procedure-name-1]
```

Format 2

```
GO TO procedure-name-1 [, procedure-name-2] . . ., procedure-name-n
     DEPENDING ON identifier
```

Each procedure name specified must name a paragraph or section in the Procedure Division. The identifier must name an elementary integer item.

Format 1-Unconditional GO TO

The unconditional GO TO statement causes control to be transferred to the first statement in the paragraph or section named in procedure-name-1 unless the GO TO statement has been modified by an ALTER statement.

When an unconditional GO TO statement appears in a sequence of imperative statements, it must be the last statement in the sequence.

When a paragraph is referred to by an ALTER statement, the paragraph can consist only of a paragraph name followed by an unconditional GO TO statement.

If procedure-name-1 is not specified in an unconditional GO TO statement, an ALTER statement must have been run before the GO TO statement. The GO TO statement must immediately follow a paragraph name and must be the only statement in the paragraph.

Format 2-Conditional GO TO

Control is transferred to one of a series of procedures, depending on the value of identifier. When the identifier has a value of 1, control is transferred to the first statement in the procedure named by procedure-name-1; if it has a value of 2, control is transferred to the first statement in the procedure named by procedure-name-2, and so on.

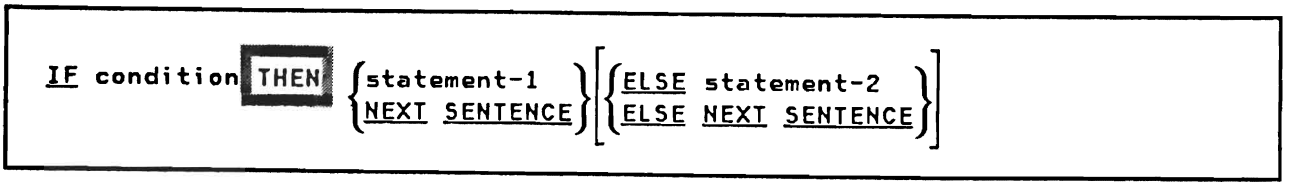
If the value of the identifier is anything other than a value within the range 1 through n (in which n is the number of procedure names specified in this GO TO statement), the conditional GO TO statement is ignored. Instead, control passes to the next statement.

The maximum number of procedure names permitted for a conditional GO TO statement is 99. The identifier field can be defined as containing up to 4 bytes.

IF Statement

The IF statement causes a condition to be evaluated, and provides for alternative actions in the object program, depending on that value.

Format



Statement-1 or statement-2 can be any one of the following:

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

If the condition tested is true, one of the following actions takes place:

- Statement-1, if specified, is run. If statement-1 contains a procedure branching statement, control is transferred according to the rules for that statement. If statement-1 does not contain a procedure-branching statement, the ELSE phrase, if specified, is ignored, and control passes to the next sentence that can be run.

- NEXT SENTENCE, if specified, is run; that is, the ELSE phrase, if specified, is ignored, and control passes to the next sentence that can be run.

If the condition tested is false, one of the following actions take place:

- ELSE statement-2, if specified, is run. If statement-2 contains a procedure-branching statement, control is transferred according to the rules for that statement. If statement-2 does not contain a procedure-branching statement, control is passed to the next sentence that can be run.

- ELSE NEXT SENTENCE, if specified, is run: statement-1, if specified, is ignored, and control passes to the next sentence that can be run.

- If ELSE clause is omitted, control passes to the next sentence that can be run.

- The ELSE NEXT SENTENCE phrase can be omitted if it immediately precedes the period that ends the conditional sentence.

Note: When the ELSE clause is omitted, all statements following the condition and preceding the period for the sentence are considered to be part of statement-1.

IBM Extension

THEN is accepted, but ignored, if present.

End of IBM Extension

Nested IF Statements

The presence of one or more IF statements within an initial IF statement constitutes a nested IF statement.

statement-1 and statement-2 in IF statements can consist of one or more imperative statements or a conditional statement or both. When an IF statement appears as statement-1 or as part of statement-1, it is considered nested statement. Nesting statements is much like specifying subordinate arithmetic expressions enclosed in parentheses and combined in larger arithmetic expressions.

IF statements contained within IF statements must be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered must be considered to apply to the immediately preceding IF that has not already paired with an ELSE.

Figure 11-13 shows the possible true or false combinations for the following nested IF statement:

```
IF condition-1
    statement-1-1
    IF condition-2
        IF condition-3
            statement-3-1
        ELSE
            statement-3-2

    ELSE
        statement-2-2
        IF condition-4
            IF condition-5
                statement-5-1
            ELSE
                statement-5-2.
```

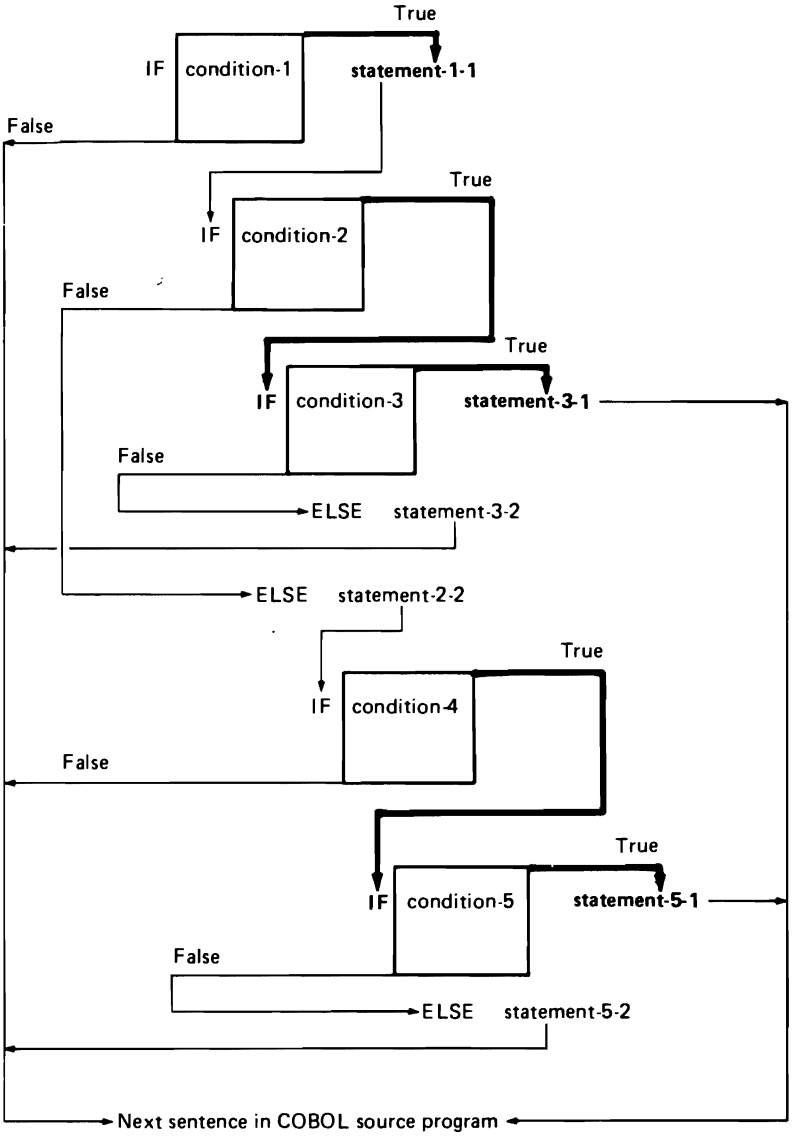


Figure 11-13. Nested IF Statement-True or False Combinations

Note: Because the logic is often difficult to follow, nested IF statements should, wherever possible, be avoided in a COBOL program. Often a series of simple IF statements can be used in place of the nested IF statement.

For example, the following series of simple IF statements give results equivalent to those achieved using the preceding nested IF statement example:

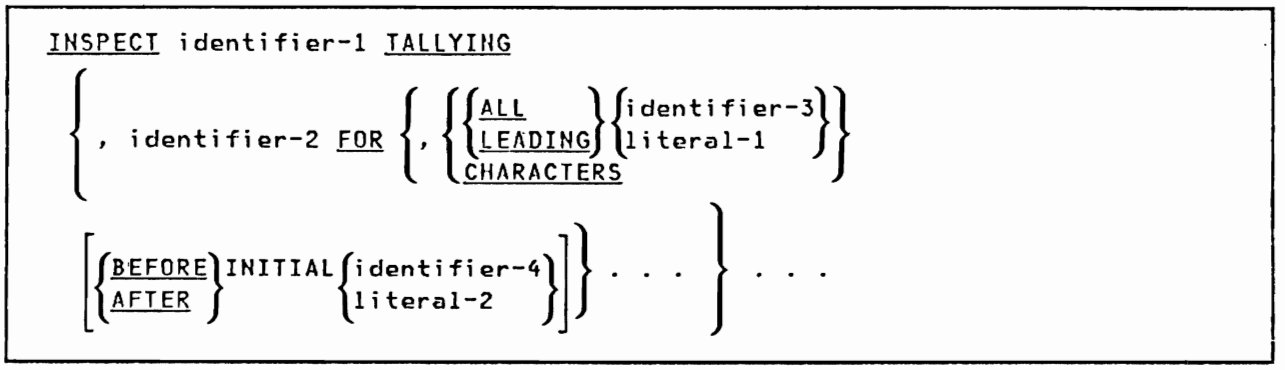
```
        IF condition-1 NEXT SENTENCE
            ELSE GO TO PARA-2.
        statement-1-1.
        IF condition-2 NEXT SENTENCE
            ELSE GO TO PARA-1.
        IF condition-3 statement-3-1 GO TO PARA-2
            ELSE statement-3-2 GO TO PARA-2.

    PARA-1.
        statement-2-2.
        IF condition-4 NEXT SENTENCE
            ELSE GO TO PARA-2.
        IF condition-5 statement-5-1
            ELSE statement-5-2.

    PARA-2.
        next-executable-statement.
```

Figure 11-13 illustrates the logic flow for the preceding series of simple IF statements, also.

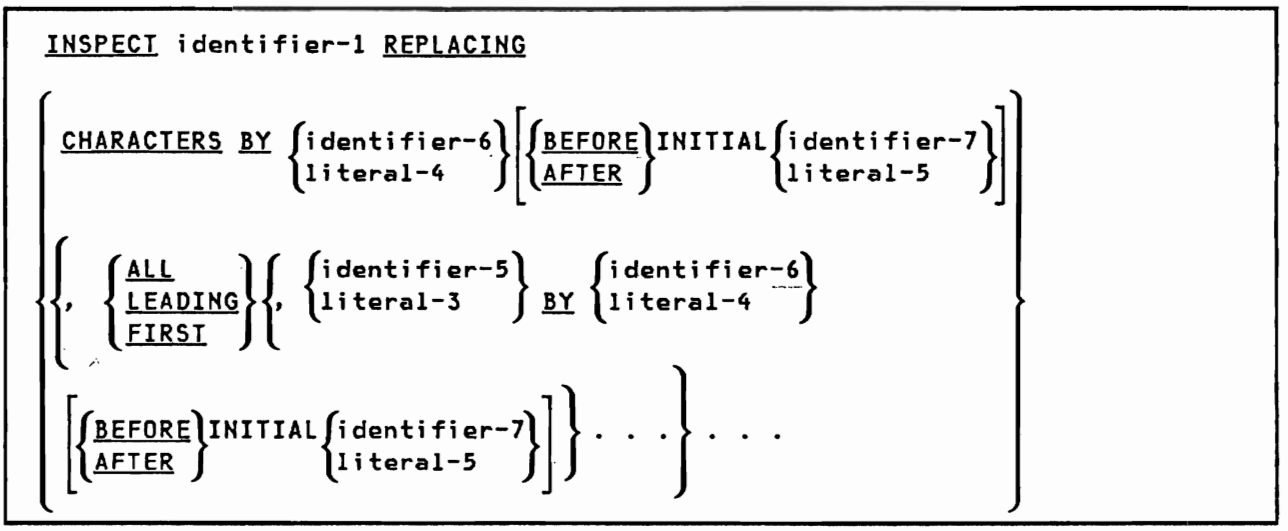
INSPECT Statement

The INSPECT statement specifies that characters in a data item are to be counted or replaced or both. The formats of the INSPECT statement are:

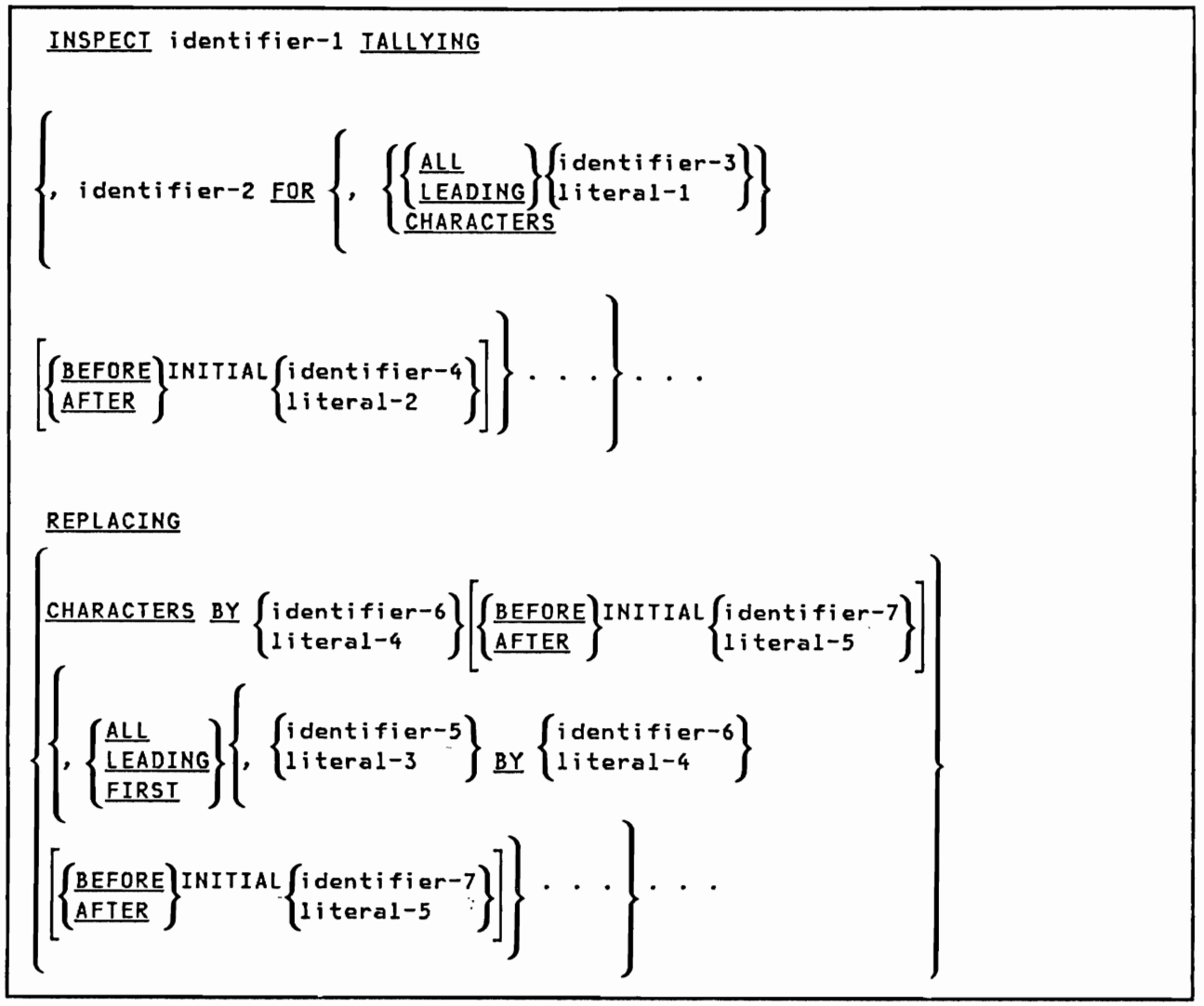
Format 1



Format 2



Format 3



You must specify either the TALLYING or the REPLACING phrase. Also, you can specify both the TALLYING and REPLACING phrases. If both TALLYING and REPLACING phrases are specified (format 3), all tallying is performed before any replacement is made.

identifier-1 is the inspected item. identifier-1 must be an elementary or group item with a USAGE 15 DISPLAY.

All other identifiers except identifier-2 (the count field) must be elementary alphabetic, alphanumeric, or zoned decimal items. Each is treated according to its data category. Each data category is treated as follows:

- Alphabetic or alphanumeric items are treated as a character string.

- Alphanumeric edited, numeric edited, or unsigned numeric (zoned decimal) items are treated as though redefined as alphanumeric and the INSPECT statement refers to the alphanumeric item.

- Signed numeric (zoned decimal) items are treated as though moved to an unsigned zoned decimal item of the same length, and then treated as though redefined as alphanumeric. The INSPECT statement refers to the alphanumeric item.

Each literal must be nonnumeric and may be any figurative constant except ALL.

The comparison operands of the TALLYING phrase (literal-1 or identifier-3, and so on) and the REPLACING phrase (literal-3 or identifier-5, and so on) are compared in the left-to-right order specified in the INSPECT statement. You can specify a maximum of 15 comparison operands for each REPLACING and each TALLYING phrase.

When the TALLYING and REPLACING operands are the compared operands, the following comparison rules apply:

1. When both the TALLYING and REPLACING phrases are specified, the INSPECT statement is run as if an INSPECT TALLYING statement were specified and immediately followed by an INSPECT REPLACING statement.

2. The first operand is compared with an equal number of leftmost contiguous characters in the inspected item. The operand matches the inspected characters only if both are equal, character for character.

3. If no match occurs for the first operand, the comparison is repeated for each successive operand until either a match is found or all operands have been acted upon.

4. If a match is found, tallying or replacing takes place as described in TALLYING or REPLACING phrase descriptions. In the inspected item, the first character following the rightmost matching character is now considered the leftmost character position. The process described in comparison rules 2 and 3 is then repeated.

5. If no match is found, the first character in the inspected item following the leftmost inspected character is now considered the leftmost character position. The process described in comparison rules 2 and 3 is then repeated.

6. The actions taken in comparison rules 1 through 5, which are defined as the comparison cycle, are repeated until the rightmost character in the inspected item has either been matched or has been considered as the leftmost character position. Inspection then ends.

Note: When either the BEFORE or the AFTER phrase is specified, the preceding rules are modified as described in *BEFORE and AFTER Phrases* later in this chapter.

Figure 11-14 illustrates INSPECT statement comparisons.

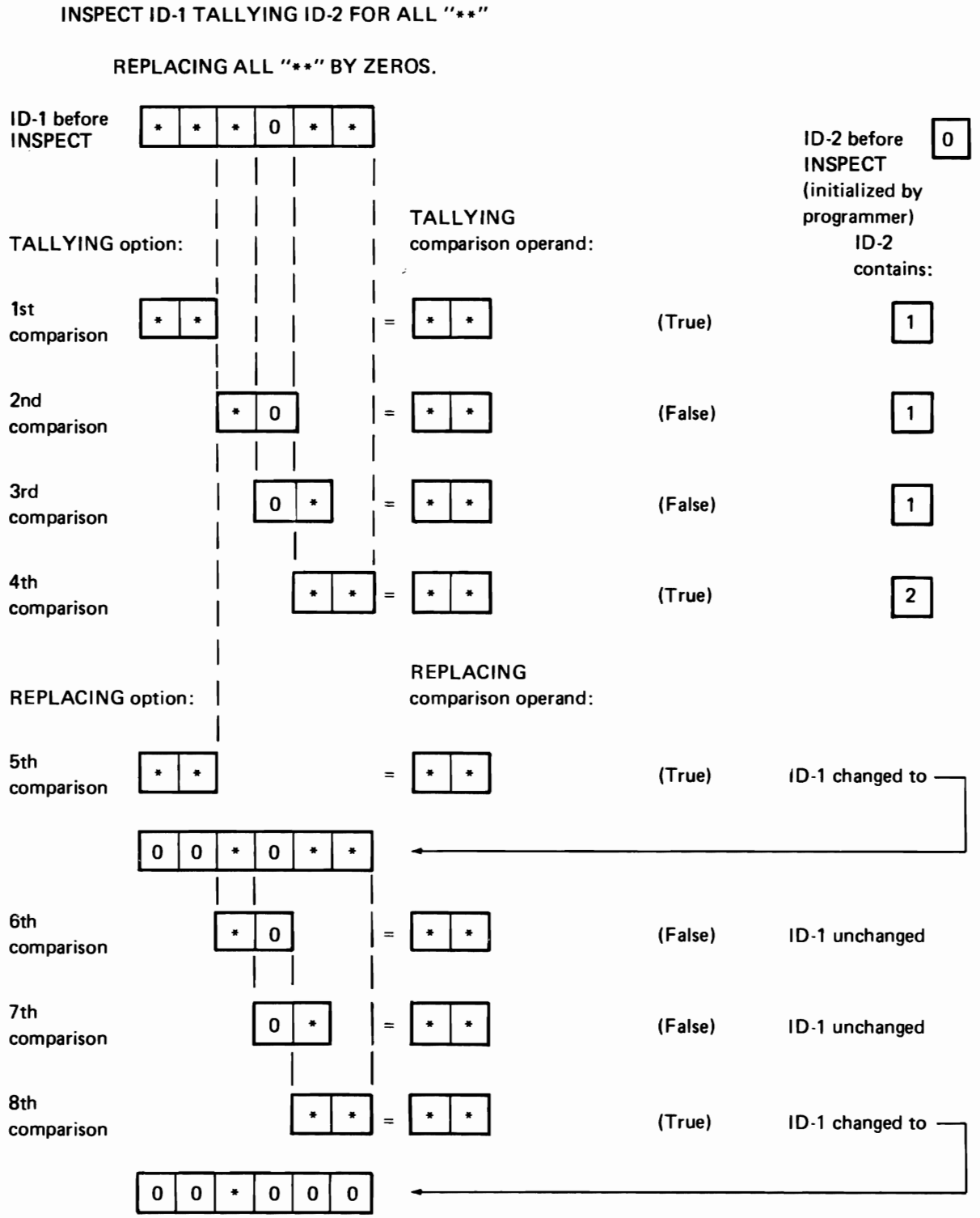


Figure 11-14. INSPECT Statement Results

INSPECT Statement Example

The following example shows an INSPECT statement:

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  ID-1 PIC X(10) VALUE 'ACADEMIANS'.
 01  CONTR-1 PIC 99 VALUE 00.
 01  CONTR-2 PIC 99 VALUE ZEROS.

 PROCEDURE DIVISION.
*    THIS ILLUSTRATES AN INSPECT STATEMENT WITH 2 VARIABLES.
 100-BEGIN-PROCESSING.
     DISPLAY CONTR-1 ' ' CONTR-2.
 101-MAINLINE-PROCESSING.
     PERFORM COUNT-IT THRU COUNT-EXIT.
     STOP RUN.
 COUNT-IT.
     INSPECT ID-1
         TALLYING CONTR-1 FOR CHARACTERS
     BEFORE INITIAL 'AD' CONTR-2 FOR ALL 'MIANS'.
 DISPLAY-COUNTS.
     DISPLAY 'CONTR-1 = ' CONTR-1.
     DISPLAY 'CONTR-2 = ' CONTR-2.

     DISPLAY '******************EOJ*******************'.
 COUNT-EXIT. EXIT.
*                RESULTANT OUTPUT


*00 00
*CONTR-1 = 02
*CONTR-2 = 01
********************EOJ*******************
```

TALLYING Phrase

identifier-2 is the tallying field and must be an elementary integer item defined without the symbol P in its PICTURE character string. It is your responsibility to initialize identifier-2 before the INSPECT statement is run.

identifier-3 or literal-1 is the comparison operand. If the comparison operand is a figurative constant, it is considered to be a 1-character nonnumeric literal.

When you do not specify either the BEFORE or the AFTER phrase, the following actions take place when the INSPECT TALLYING statement is run:

- If you specified the ALL phrase, the tallying field is increased by 1 for each nonoverlapping occurrence in the inspected item of the comparison operand. This process begins at the leftmost character position and continues to the rightmost.

- If you specify the LEADING phrase, the tallying field is increased by 1 for each contiguous nonoverlapping occurrence of the comparison operand in the inspected item. This only happens if the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which the comparison operand is eligible to participate.

- If you specify the CHARACTERS phrase, the tallying field is increased by 1 for each character (including the space character) in the inspected item. Thus, running the INSPECT TALLYING statement increases the value in the tallying field by the number of characters in the inspected item.

REPLACING Phrase

Identifier-5 or literal-3 is the comparison operand. Identifier-6 or literal-4 is the replacement field.

The comparison operand and the replacement field must be the same length. The following replacement rules apply:

- If the comparison operand is a figurative constant, it is considered to be a 1-character nonnumeric literal. Each character in the inspected item equivalent to the figurative constant is replaced by the single-character replacement field, which must be 1-character in length.

- If the replacement field is a figurative constant, it is considered to be the same length as the comparison operand. Each nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the replacement field.

- When the comparison operand and replacement fields are character strings, each nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the character string specified in the replacement field.

- Once replacement has occurred in a given character position in the inspected item, no further replacement for that character position is made in this run of the INSPECT statement.

When you do not specify either the BEFORE or the AFTER phrase, the following actions take place when the INSPECT REPLACING statement is run:

- If you specified the CHARACTERS phrase, the replacement field must be 1 character in length. Each character in the inspected field is replaced by the replacement field. This process begins at the leftmost character and continues to the rightmost.

- If you specified the ALL phrase, each nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the replacement field, beginning at the leftmost character and continuing to the rightmost.

- If you specified the LEADING phrase, each contiguous nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the replacement field, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which this replacement field is eligible to participate.

- If you specified the FIRST phrase, the leftmost occurrence of the comparison operand in the inspected item is replaced by the replacement field.

BEFORE and AFTER Phrases

When you specify either of these phrases, the preceding rules for counting and replacing are modified.

identifier-4, identifier-7, literal-2, and literal-5 are delimiters. Counting and replacement of the inspected item are bounded by their presence; however, the delimiters themselves are neither counted nor replaced.

In the TALLYING phrase, if the delimiter (literal-2) is a figurative constant, it is considered to be 1 character long.

In the REPLACING phrase, if you specify the CHARACTERS phrase, the delimiter (literal-5 or identifier-7) must be 1 character long.

When you specify the BEFORE phrase, tallying and replacement of the inspected item begins at the leftmost character and continues until the first occurrence of the delimiter is encountered. If no delimiter is present in the inspected item, counting and replacement continues to the rightmost character.

When you specify the AFTER phrase, counting and replacement of the inspected item begins with the first character to the right of the delimiter and continues to the rightmost character in the inspected item. If no delimiter is present in the inspected item, neither counting nor replacement takes place.

INSPECT Statement Examples

The following examples illustrate some uses of the INSPECT statement. In all instances, the COUNTR field is set to 0 before the INSPECT statement is run.

```
INSPECT ID-1 REPLACING CHARACTERS BY ZERO.
```

ID-1 Before	COUNTR After	ID-1 After
1234567	0	0000000
HIJKLMN	0	0000000

```
INSPECT ID-1 TALLYING COUNTR FOR CHARACTERS
REPLACING CHARACTERS BY SPACES.
```

ID-1 Before	COUNTR After	ID-1 After
1234567	7	
HIJKLMN	7	

```
INSPECT ID-1 REPLACING CHARACTERS BY ZEROS
BEFORE INITIAL QUOTE.
```

ID-1 Before	COUNTR After	ID-1 After
456'ABEL	0	000'ABEL
ANDES'12	0	00000'12
'TWAS BR	0	'TWAS BR

```
INSPECT ID-1 TALLYING COUNTR FOR CHARACTERS
AFTER INITIAL 'S' REPLACING ALL 'A'
BY 'O'.
```

ID-1 Before	COUNTR After	ID-1 After
ANSELM	3	ONSELM
SACKET	5	SOCKET
PASSED	3	POSSED

```
INSPECT ID-1 TALLYING COUNTR FOR LEADING '0'
REPLACING FIRST 'A' BY '2' AFTER INITIAL 'C'.
```

ID-1 Before	COUNTR After	ID-1 After
00ACADEMY00	2	00AC2DEMY00
0000ALABAMA	4	0000ALABAMA
CHATHAM0000	0	CH2THAM0000

Typical Uses

The INSPECT statement is useful for filling portions for all of a data item with spaces or 0's. It is also useful for counting the number of times a specific character (for example, 0, space, asterisk) occurs in a data item. In addition, it can be used to translate characters from one collating sequence to another.

MOVE Statement

The MOVE statement transfers data from one area of storage to one or more other areas. The formats of the MOVE statement are as follows:

Format 1

```
MOVE {identifier-1} TO identifier-2 [, identifier-3] . . .
     {literal     }
```

Format 2

```
MOVE {CORRESPONDING} identifier-1 TO identifier-2
     {CORR          }
```

identifier-1 and literal-1 are the sending areas. identifier-2, identifier-3, and so on are the receiving areas.

When you specify format 1, the identifiers can be either group or elementary items. The data in the sending area is moved into the first receiving area (identifier-2); it is then moved into the second receiving area (identifier-3), and so on.

You cannot specify an index data item in a MOVE statement. Any subscripting or indexing associated with the sending item is evaluated only once, immediately before the data is moved to the first receiving field. Any subscripting or indexing associated with the receiving items is evaluated immediately before the data is moved into the receiving field.

For example, the result of the statement:

```
MOVE A (B) TO B, C (B).
```

is equivalent to

```
MOVE A (B) TO TEMP.
MOVE TEMP TO B.
MOVE TEMP TO C (B).
```

in which TEMP has been defined as an intermediate result item. The subscript B changed in value between the time the first move, and the final move to C (B), took place.

After a MOVE statement is run, the sending field(s) contains the same data as before the statement was run.

Note: When using elementary data items that have been combined through the RENAMES clause in both the sending and receiving fields, the sending field will contain unpredictable data after the MOVE statement has been completed.

CORRESPONDING Phrase

The CORRESPONDING phrase lets operations be performed on elementary items of the same name. You simply specify the group items to which the elementary items belong. The results are the same as if each pair of CORRESPONDING identifiers had been referred to in a separate MOVE statement; however, the individual MOVE statements reduce compile time.

The abbreviation CORR can be used in place of the key word CORRESPONDING.

Both identifiers following the key word CORRESPONDING must name group items. In this discussion, these identifiers are referred to as d1 and d2.

A pair of subordinate data items, one from d1 and one from d2, correspond if the following conditions are true:

- At least one of the subordinate items is elementary.
- The two subordinate items have the same name and the same qualifiers up to but not including d1 and d2.
- The subordinate items are not identified by the key word FILLER.
- The subordinate items do not include a REDEFINES, a RENAMES, an OCCURS, or a USAGE IS INDEX clause in their descriptions; if such a subordinate item is a group, the items subordinate to it are also ignored. However, d1 and d2 themselves can contain or be subordinate to items containing a REDEFINES or OCCURS clause in their descriptions.

  For example, two data hierarchies are defined as follows:

```
05 ITEM-1 OCCURS 6 INDEXED BY X.
   10 ITEM-A ...
   10 ITEM-B ...
   10 ITEM-C REDEFINES ITEM-B ...
05 ITEM-2
   10 ITEM-A ...
   10 ITEM-B ...
   10 ITEM-C ...
```

  If you specify MOVE CORR ITEM-2 TO ITEM-1(X), ITEM-A and ITEM-A(X) and ITEM-B and ITEM-B(X) are considered to be corresponding. Thus, ITEM-A and ITEM-B of ITEM-2 are moved to ITEM-1(X). ITEM-C and ITEM-C(X) are not included, because ITEM-C(X) includes a REDEFINES clause in its data description. ITEM-1 is valid as either d1 or d2.

- Neither d1 nor d2 is described as a level-66, -77, or -88 item, or as a FILLER or USAGE IS INDEX item.

Elementary Moves

An elementary move is one in which both the sending and receiving items are elementary items. Each elementary item belongs to one of the following categories:

- Numeric--Includes numeric data items and numeric literals
- Alphabetic--Includes alphabetic data items and the figurative constant SPACE/SPACES
- Alphanumeric--Includes alphanumeric data items, nonnumeric literals, and all figurative constants except ZERO and SPACE
- Alphanumeric edited--Includes alphanumeric edited data items
- Numeric edited--Includes numeric edited data items
- Figurative constant ZERO/ZEROS/ZEROES.

IBM Extension

Boolean--Includes Boolean data items, Boolean literals, and the figurative constant ZERO/ZEROS/ZEROES when the receiving item is Boolean.

End of IBM Extension

Valid elementary moves take place according to the following rules:

- Any necessary conversion of data from one form of internal representation to another along with any specified editing in the receiving item takes place during the move.
- For an alphabetic receiving item:
  - Justification and any necessary space filling take place as described in the JUSTIFIED clause. Unused character positions are filled with spaces.
  - If the size of the sending item is greater than the size of the receiving item, excess characters at the right are truncated after the receiving item is filled.

IBM Extension

If the sending item is Boolean, and the receiving item is alphanumeric or alphanumeric edited, no data conversion takes place.

End of IBM Extension

- For an alphanumeric or alphanumeric edited receiving item:
  - Justification and any necessary space filling take place as described in the JUSTIFIED clause. Unused character positions are filled with spaces.

- If the size of the sending item is greater than the size of the receiving item, excess characters at the right are truncated after the receiving item is filled.
- If the sending item has an operational sign, the absolute value is used. If the operational sign occupies a separate character, that character is not moved, and the size of the sending item is considered to be 1 less than its actual size.

IBM Extension

If the sending item is Boolean, and the receiving item is alphanumeric or alphanumeric edited, no data conversion takes place.

End of IBM Extension

- For a numeric or numeric edited receiving item:
  - Alignment by decimal point and any necessary zero filling take place as described under *Standard Alignment Rules* in Chapter 10, except where 0's are replaced because of editing requirements.
  - The absolute value of the sending item is used if the receiving item has no operational sign.
  - If the sending item has more digits to the left or right of the decimal point than the receiving item can contain, excess digits are truncated.
  - The results at object time may be unpredictable if the sending item contains any nonnumeric characters.

IBM Extension

For a Boolean receiving item:

- There is no data conversion.
- The source field must be either alphanumeric or Boolean.
- Running the MOVE statement does not affect the association of an indicator number to the data name.

End of IBM Extension

Note: If the receiving field is alphanumeric or numeric edited, and the sending field is a scaled integer (that is, it has a P as the rightmost character in its PICTURE character string), the scaling positions are treated as trailing 0's when the MOVE statement is run.

Figure 11-15 shows valid and invalid elementary moves for each category.

Sending Item Category	Receiving Item Category						
	Alphabetic	Alphanumeric	Alphanumeric Edited	Numeric Integer	Numeric Noninteger	Numeric Edited	Boolean
Alphabetic and SPACE	YES	YES	YES	NO	NO	NO	NO
Alphanumeric and Figurative constant[1]	YES	YES	YES	YES	YES	YES	YES
Alphanumeric Edited	YES	YES	YES	NO	NO	NO	NO
Numeric Integer[2]	NO	YES	YES	YES	YES	YES	NO
Numeric Noninteger[2]	NO	NO	NO	YES	YES	YES	NO
Numeric Edited	NO	YES	YES	NO	NO	NO	NO
Boolean[3]	NO	YES	YES	NO	NO	NO	YES
ZERO/ZEROS/ ZEROES	NO	YES	YES	YES	YES	YES	YES

YES = move is valid
NO = move is invalid

[1]Includes nonnumeric literals and all figurative constants but SPACE and ZERO
[2]Includes numeric literals
[3]Includes Boolean literals

Figure 11-15. Valid and Invalid Elementary Moves

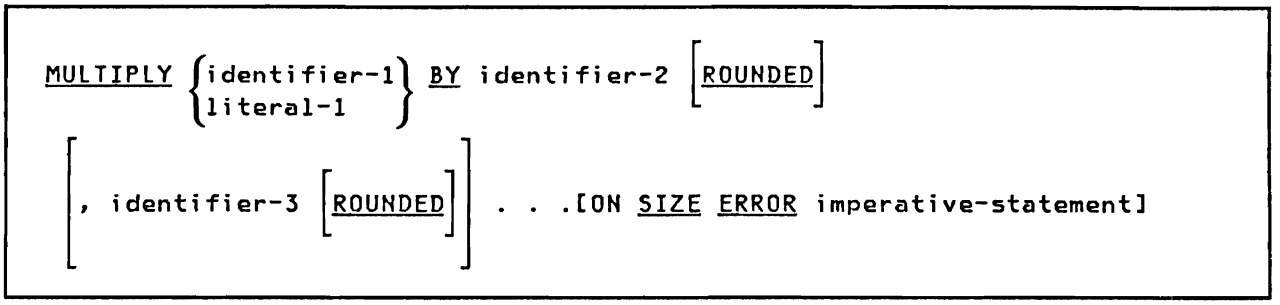
Group Moves

A group move is one in which one or both of the sending and receiving fields are a group item. A group move is treated exactly as though it were an alphanumeric elementary move except that data is not converted from one form of internal representation to another. In a group move, the receiving area is filled without consideration for the individual elementary items contained within either the sending area or the receiving area.

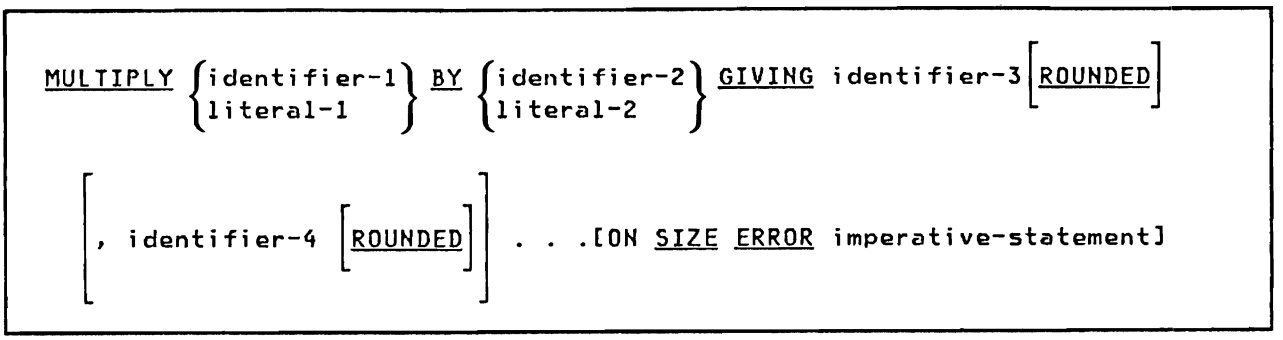
MULTIPLY Statement

The MULTIPLY statement causes numeric items to be multiplied and sets the values of data items equal to the results. The formats of the MULTIPLY statement are:

Format 1



Format 2



Each identifier except those following the key word GIVING must name an elementary numeric item. Each identifier following the key word GIVING must name an elementary numeric or numeric edited item. Each literal must be a numeric literal.

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. If you specify identifier-3, the value of identifier-1 or literal-1 is multiplied by the value of identifier-3; the product is then placed in identifier-3, and so on.

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 identifier-3, and, if specified, identifier-4, and so on.

ROUNDED Phrase

After decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is compared with the number of places provided for the fraction of the resultant identifier.

If the size of the fractional result exceeds the number of places provided for its storage, truncation occurs unless the ROUNDED phrase is specified. When the

ROUNDED phrase is specified, the least-significant digit of the resultant identifier has its value increased by 1 whenever the most-significant digit of the excess is greater than or equal to 5.

When the resultant identifier is described by a PICTURE clause containing rightmost Ps, and when the number of places in the calculated result exceeds the number of integer positions specified, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

SIZE ERROR Phrase

A size error condition exists if, after decimal point alignment, the value of a result exceeds the largest value that can be contained in the resultant field. In the MULTIPLY statement, the size error condition applies both to final results and to intermediate results.

If you specify the ROUNDED phrase, rounding takes place before size error checking.

When a size error occurs, the subsequent action of the program depends on whether or not the SIZE ERROR phrase is specified.

If you do not specify the SIZE ERROR phrase and a size error condition occurs, the value of the affected resultant identifier is unpredictable. When you specify multiple receivers, those that do not have a size error are not affected by receivers that do have the error.

If you specify the SIZE ERROR phrase and a size error condition occurs, the error results are not placed in the receiving identifier. When the arithmetic operation is completed, the imperative statement in the SIZE ERROR phrase is run.

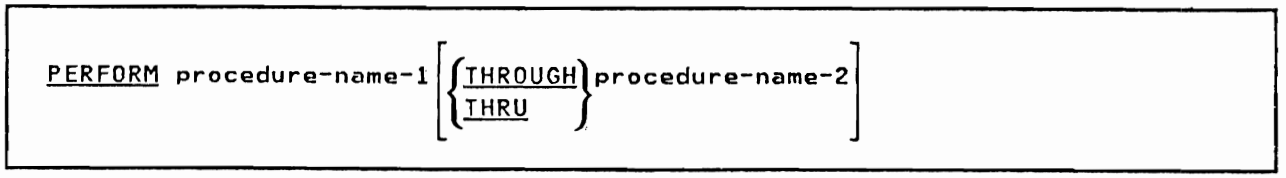
GIVING Phrase

If you specify the GIVING phrase, the value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation. Because this identifier is not involved in the computation, it can be a numeric edited item.

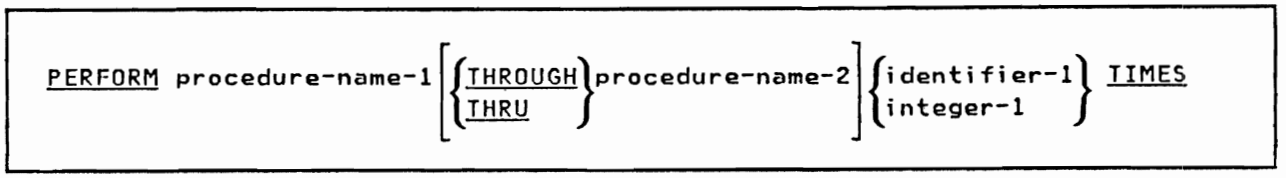
PERFORM Statement

The PERFORM statement transfers control explicitly to one or more procedures and implicitly returns control to the next statement that can be run after the specified procedure(s) has run. The formats of the PERFORM statement are as follows:

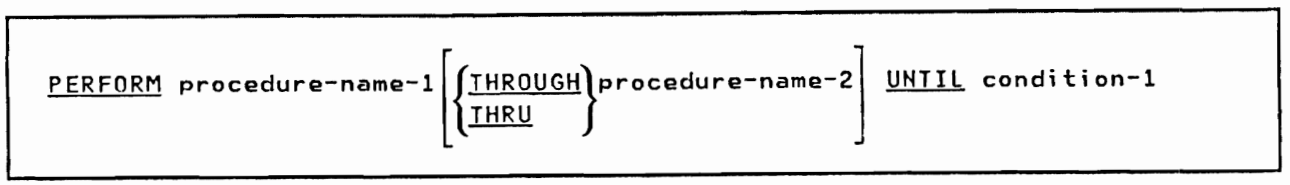
Format 1



Format 2



Format 3



Format 4

```
PERFORM procedure-name-1 [{THROUGH}procedure-name-2]
                         [{THRU   }                ]

  VARYING {identifier-1} FROM {identifier-2}
          {index-name-1}      {index-name-2}
                              {literal-2   }
  BY {identifier-3} UNTIL condition-1
     {literal-3   }

 [AFTER {identifier-4} FROM {identifier-5}
        {index-name-4}      {index-name-5}
                            {literal-5   }
  BY {identifier-6} UNTIL condition-2
     {literal-6   }

  [AFTER {identifier-7} FROM {identifier-8}
         {index-name-7}      {index-name-8}
                             {literal-8   }

   BY {identifier-9} UNTIL condition-3]
      {literal-9   }                  ]
```

Each procedure name must name a section or a paragraph in the Procedure Division.

When you specify both procedure-name-1 and procedure-name-2, if either is a procedure name in a Declarative procedure, then both must be procedure names in the same Declarative procedure.

Each identifier must name a numeric elementary item.

Each literal must be a numeric literal.

Whenever a PERFORM statement is run, control transfers to the first statement of the procedure named procedure-name-1. Control always returns to the statement following the PERFORM statement. The point from which this control returns is determined as follows:

- If procedure-name-1 is a paragraph name and you do not specify procedure-name-2, the return is made after the last statement of procedure-name-1 is run.

- If procedure-name-1 is a section name and you do not specify procedure-name-2, the return is made after the last sentence of the last paragraph in that section is run.

- If you specify procedure-name-2 and it is a paragraph name, the return is made after the last statement of that paragraph is run.

- If you specify procedure-name-2 and it is a section name, the return is made after the last sentence of the last paragraph in the section is run.

The only necessary relationship between procedure-name-1 and procedure-name-2 is that a consecutive sequence of operations is run beginning at the procedure named by procedure-name-1 and ending with the running of the procedure named by procedure-name-2.

When you specify both procedure-name-1 and procedure-name-2, GO TO and PERFORM statements can appear within the sequence of statements contained in these paragraphs or sections. When you specify only procedure-name-1, PERFORM statements can appear within the procedure. A GO TO statement can also appear, but should not refer to a procedure name outside the range of procedure-name-1. If this is done, results are unpredictable and are not diagnosed.

When the performed procedures include another PERFORM statement, the sequence of procedures associated with the embedded PERFORM statement must be totally included in or totally excluded from the performed procedures of the first PERFORM statement: An active PERFORM statement that begins within the range of performed procedures of another active PERFORM statement must not allow control to pass through the exit point of the other active PERFORM statement. Also, two or more such active PERFORM statements must not have a common exit.

When control passes to the sequence of procedures by means other than a PERFORM statement, control passes through the exit point to the next statement that can be run, as if no PERFORM statement referred to these procedures.

Figure 11-16 illustrates valid sequences for PERFORM statements.

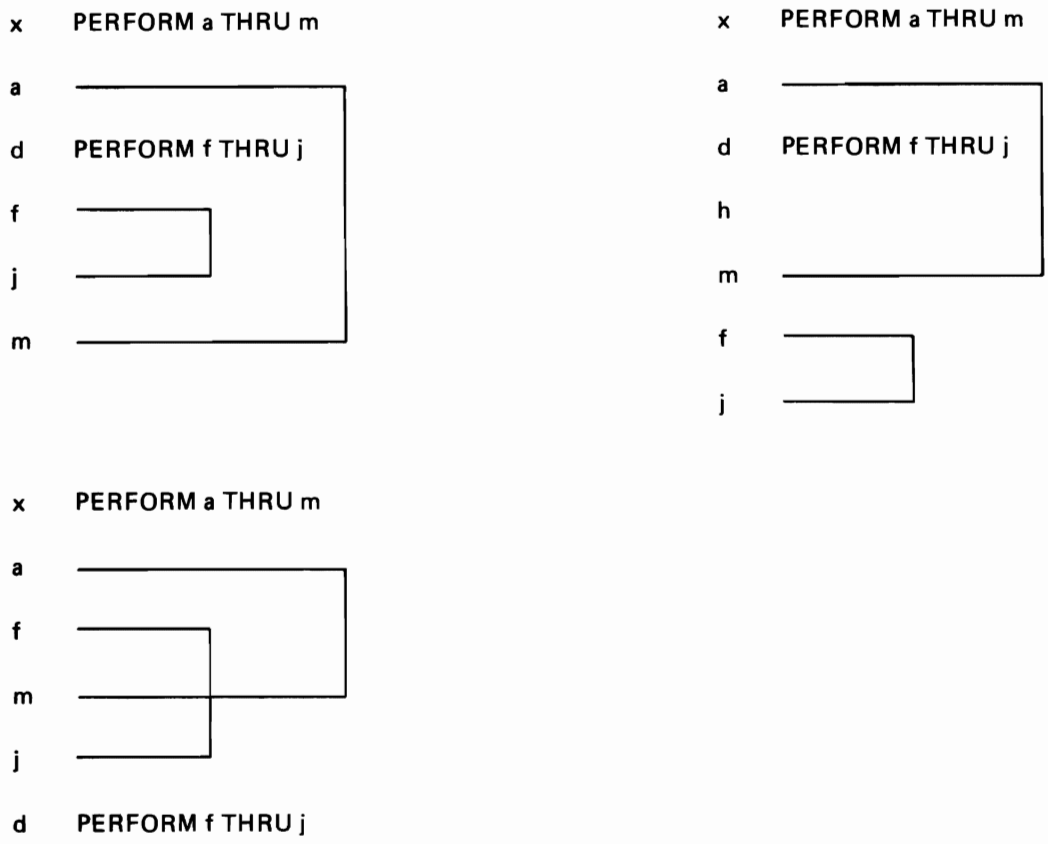


Figure 11-16. Valid PERFORM Statement Sequences

The preceding rules refer to all four formats of the PERFORM statement. The following sections give rules applying to each individual format.

Format 1

Format 1 is the basic PERFORM statement. The procedure(s) referred to is run once, and then control passes to the next runnable statement following the PERFORM statement.

Format 2

Format 2 uses the TIMES phrase. identifier-1 must name an integer item. The procedure(s) referred to is run the number of times specified by the value in identifier-1 or integer-1. Control then passes to the next runnable statement following the PERFORM statement. The following rules apply:

- If integer-1 or identifier-1 is 0 or a negative number at the time the PERFORM statement is initiated, control passes to the statement following the PERFORM statement.

- After the PERFORM statement has been initiated, any reference to identifier-1 or change in the value of identifier-1 has no effect in varying the number of times the procedures are run.

Format 3

Format 3 uses the UNTIL phrase. The procedure(s) referred to is performed until the condition specified by the UNTIL phrase is true. Control is then passed to the next runnable statement following the PERFORM statement.

If condition-1 is true at the time the PERFORM statement is encountered, the specified procedure(s) is not run.

Format 4

Format 4 uses the VARYING phrase. This phrase increments or decrements one or more identifiers or index names according to the following rules. Once the condition(s) specified in the UNTIL phrase is satisfied, control is passed to the next runnable statement following the PERFORM statement.

No matter how many variables are specified, the following rules apply:

- In the VARYING and AFTER phrases, when an index name is specified:

  - The index name is initialized and incremented or decremented according to the rules for the SET statement. For a description of the SET statement see *Procedure Division Table Handling* in Chapter 13.
  - In the associated FROM phrase, an identifier must be described as an integer and have a positive value; a literal must be a positive integer.
  - In the associated BY phrase, an identifier must be described as an integer; a literal must be a nonzero integer.

- In the FROM phrase, when an index name is specified:

  - In the associated VARYING or AFTER phrase, an identifier must be described as an integer. It is initialized as described in the SET statement.

- In the associated BY phrase, an identifier must be described as an integer and have a nonzero value; a literal must be a nonzero integer.

- In the BY phrase, identifiers and literals must have a nonzero value.

- Changing the values of identifiers or index names or both in the VARYING, FROM, and BY phrases when the procedures are running changes the number of times the procedures are run.

The way in which operands are incremented or decremented depends on the number of variables specified. In the following discussion, every reference to identifier-n refers equally to index-name-n except when identifier-n is the object of the BY phrase.

Varying One Identifier: The following actions take place:

1. identifier-1 is set equal to its starting value, identifier-2 or literal-2.

2. condition-1 is evaluated:

   a. If it is false, steps 3 through 5 are run.
   b. If it is true, control passes directly to the statement following the PERFORM statement.

3. procedure-1 through procedure-2 (if specified) are run once.

4. identifier-1 is increased by identifier-3 (or literal-3), and condition-1 is evaluated again.

5. Steps 2 through 4 are repeated until condition-1 is true.

Figure 11-17 is a flowchart illustrating the logic of the PERFORM statement when one identifier is varied.

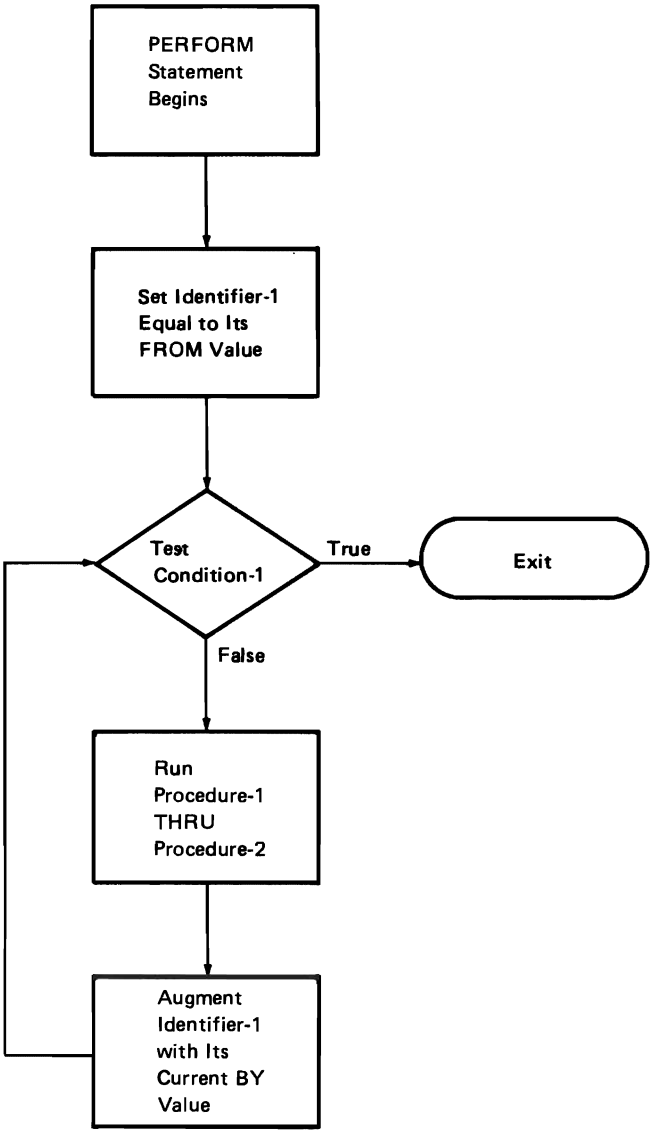


Figure 11-17. Format 4 PERFORM Statement Logic-Varying One Identifier

The following example shows a PERFORM statement varying one identifier. This PERFORM logic is run 100 times.

```
 WORKING-STORAGE SECTION.
 77  SUB1 PIC 999.
 77  TOTAL-HOLD PIC 99 VALUE 57.
 77  HOLD-2 PIC 99 VALUE 10.
 77  HOLD-THE-SUM PIC 99 VALUE ZERO.
 01  TABLE-ELEMENT.
     03 ELEMENTS-OF-TABLE OCCURS 100 TIMES PIC 9.
 PROCEDURE DIVISION.
 100-START-PROCESSING.

*    THIS PERFORM LOGIC IS EXECUTED 100 TIMES.

     PERFORM SAMPLE-PERFORM THRU PERFORM-EXIT VARYING SUB1
     FROM 1 BY 1 UNTIL SUB1 > 100.

*    THIS ADD STATEMENT IS EXECUTED AFTER PERFORM IS DONE.

     ADD TOTAL-HOLD HOLD-2 GIVING HOLD-THE-SUM.

     DISPLAY 'TOTAL OF TWO VARIABLES = ' HOLD-THE-SUM.
     PERFORM ANOTHER-WAY-TO-INITIALIZE THRU AWTI-EXIT.
*    *************************.
*
*    THE TABLE WILL BE ALL ZEROS AND SHOULD PRINT AS SUCH.
*
*    *************************.
     DISPLAY '------------------THE-------TABLE----------'.
     DISPLAY TABLE-ELEMENT.
     STOP RUN.
 SAMPLE-PERFORM.
     MOVE ZEROS TO ELEMENTS-OF-TABLE (SUB1).

 PERFORM-EXIT. EXIT.
 ANOTHER-WAY-TO-INITIALIZE.
     MOVE ZEROS TO TABLE-ELEMENT.
 AWTI-EXIT. EXIT.
*    ******************END OF PROGRAM*************************
```

Varying Two Identifiers: The following actions take place:

1. identifier-1 and identifier-4 are set to their initial values, identifier-2 (or literal-2) and identifier-5 (or literal-5), respectively.

2. condition-1 is evaluated:

   a. If it is false, steps 3 through 7 are run.
   b. If it is true, control passes directly to the statement following the PERFORM statement.

3. condition-2 is evaluated:

   a. If it is false, steps 4 through 6 are run.
   b. If it is true, identifier-4 is set to the current value of identifier-5, and identifier-1 is augmented by identifier-3 (or literal-3), and step 2 is repeated.

4. procedure-1 through procedure-2 (if specified) are performed once.

5. identifier-4 is increased by identifier-6 (or literal-6).

6. Steps 3 through 5 are repeated until condition-2 is true.

7. Steps 2 through 6 are repeated until condition-1 is true.

Upon completion of the PERFORM statement, identifier-4 contains the current value of identifier-5. identifier-1 has a value that exceeds the last-used setting by the increment or the decrement value (unless condition-1 was true at the beginning of the PERFORM statement run, in which case identifier-1 contains the current value of identifier-2).

Figure 11-18 is a flowchart illustrating the logic of the PERFORM statement when two identifiers are varied.

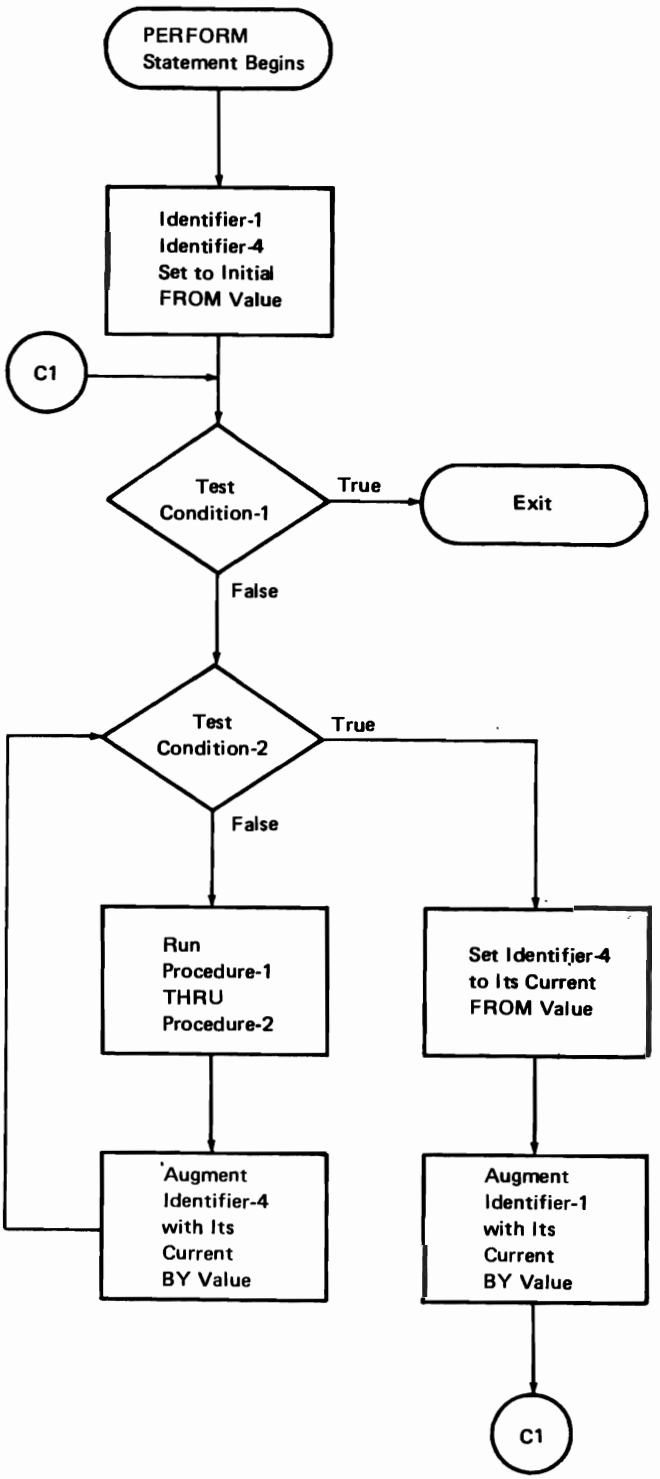


Figure 11-18. Format 4 PERFORM Statement Logic-Varying Two Identifiers

The following example shows a **PERFORM** statement varying two identifiers. This **PERFORM** logic is run 126 times. This program searches a table and gives a total of female employees.

```
DATA DIVISION.
FILE SECTION.
FD  PRINTED-REPORT
    RECORDS CONTAINS 132 CHARACTERS
    LABEL RECORDS STANDARD
    DATA RECORD IS REPORT-LINE.
01  REPORT-LINE PIC X(132).
FD  EMPLOYEE-DATA
    BLOCK CONTAINS 1 RECORDS
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS STANDARD
    DATA RECORD IS EMPLOYEE-RECORD.
01  EMPLOYEE-RECORD PIC X(80).
WORKING-STORAGE SECTION.
77  RECORDS-IN PIC 9(5) VALUE ZEROS.
77  EOF-SW PIC X VALUE 'N'.
01  HOLD-INPUT-RECORD.
    03  EMPLOYEE-SEX PIC 9.
        88 MALE VALUE IS 1.
        88 FEMALE VALUE IS 2.
    03  EMPLOYEE-RACE PIC 9.
        88 RACE-CODES VALUES ARE 1 THRU 7.
    03  EMPLOYEE-JOB-CLASS PIC 99.
    88 JOB-CLASS VALUES ARE 01 THRU 18.
    03  FILLER PIC X(76) VALUE SPACES.
01  EMPLOYEE-TABLE.
    03  E-SEX OCCURS 2 TIMES.
        05  E-RACE OCCURS 7 TIMES.
            07  E-JOB OCCURS 18 TIMES PIC 99.
77  SUB1 PIC 99.
77  SUB2 PIC 99.
77  SUB3 PIC 99.
77  TOTAL-WOMEN PIC 9(5) VALUE ZEROS.
```

```
 PROCEDURE DIVISION.
 100-START-IT.
     OPEN INPUT EMPLOYEE-DATA OUTPUT PRINTED-REPORT.
     MOVE ZEROS TO EMPLOYEE-TABLE.
 200-READ-IT.
     READ EMPLOYEE-DATA RECORD INTO HOLD-INPUT-RECORD
     AT END MOVE 'Y' TO EOF-SW.
     ADD 1 TO RECORDS-IN.
 300-MAIN-LOGIC.
*****************************************
*    THE PERFORM STATEMENT USING TWO
*    VARIABLES WILL BE DONE 126
*    TIMES BY THE COMPUTER.
*****************************************
     PERFORM LOAD-TABLE UNTIL EOF-SW = 'Y'.
     PERFORM FIND-NUMBER-OF-WOMEN VARYING SUB2 FROM 1 BY 1
     UNTIL SUB2 > 7
     AFTER SUB3 FROM 1 BY 1 UNTIL SUB3 > 18.
     PERFORM WRITE-REPORT THRU WR-EXIT.
     DISPLAY 'TOTAL RECORDS IN ' RECORDS-IN.
     STOP RUN.
 LOAD-TABLE.
     MOVE EMPLOYEE-SEX TO SUB1.
     MOVE EMPLOYEE-RACE TO SUB2.
     MOVE EMPLOYEE-JOB-CLASS TO SUB3.
     ADD 1 TO E-JOB (SUB1,SUB2,SUB).
     PERFORM 200-READ-IT.
 FIND-NUMBER-OF-WOMEN.
     ADD E-JOB (2,SUB2,SUB3) TO TOTAL-WOMEN.
 WRITE-REPORT.
     MOVE TOTAL-WOMEN TO PRINT-OUT.
     WRITE PRINT-OUT.
 WR-EXIT. EXIT.
```

Varying Three Identifiers: The actions are the same as for varying two identifiers except that identifier-7 goes through the complete cycle each time that identifier-4 is increased by identifier-6 or literal-6, which in turn goes through a complete cycle each time identifier-1 is varied.

Upon completion of the PERFORM statement, identifier-4 and identifier-7 contain the current values of identifier-5 and identifier-8, respectively. identifier-1 has a value exceeding its last-used setting by one increment or decrement value (unless condition-1 was true at the beginning of the PERFORM statement run, in which case identifier-1 contains the current value of identifier-2).

Figure 11-19 is a flowchart illustrating the logic of the PERFORM statement when three identifiers are varied.

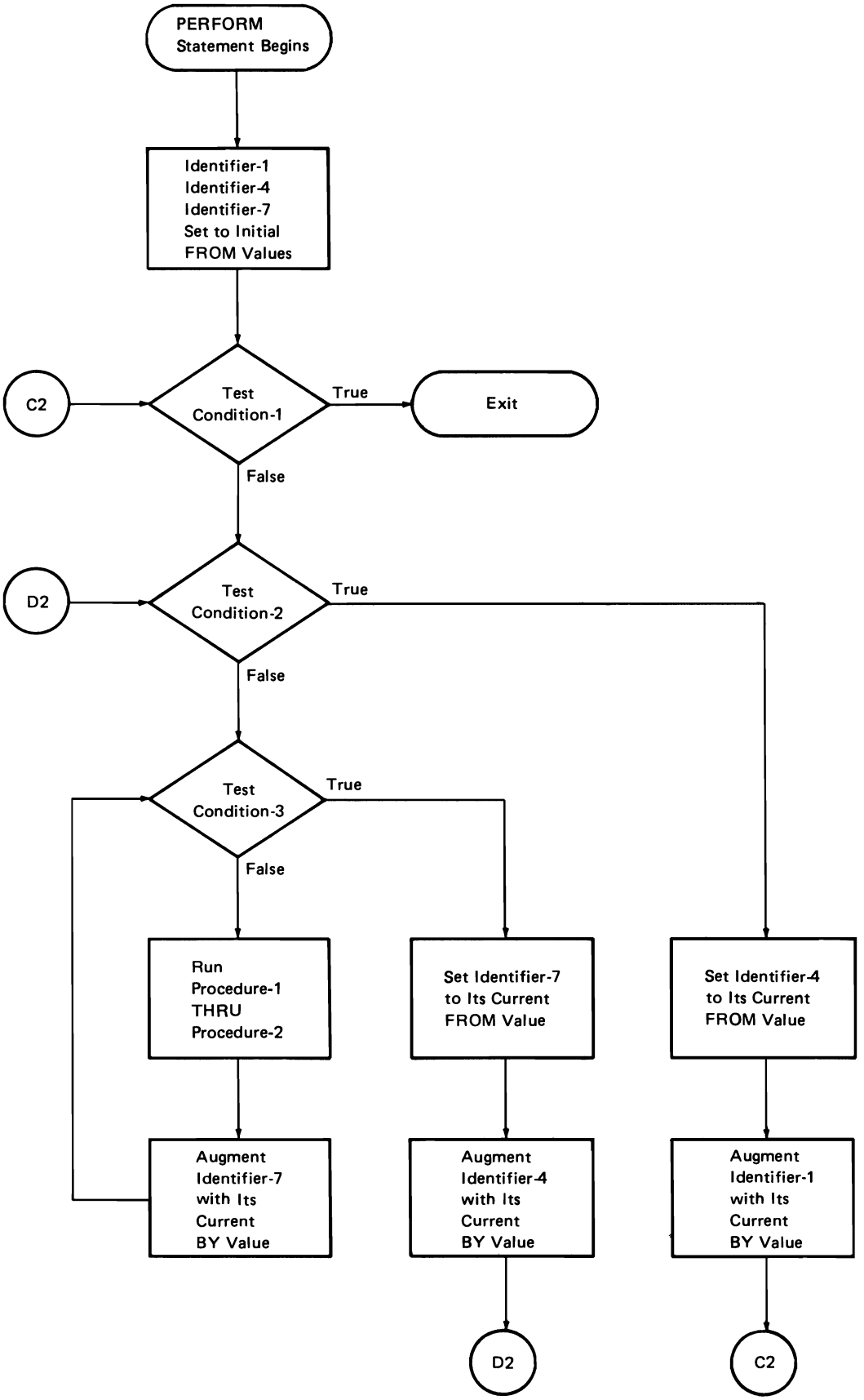


Figure 11-19. Format 4 PERFORM Statement Logic-Varying Three Identifiers

The following example shows a PERFORM statement varying three identifiers. This PERFORM logic is run 250 times.

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77  SUB1 PIC 99.
 77  SUB2 PIC 99.
 77  SUB3 PIC 99.
 77  TEST-IT PIC 99 VALUE 00.
 77  TOTAL-RECS PIC 99 VALUE ZEROS.
 01  COMPANY-TABLE.

     05  DIVISION-IN OCCURS 10 TIMES.
         10  DIVISION-NAME PIC X(10).
         10  DIVISION-NUMBER PIC 9(4).

         10  SECTION-IN OCCURS 5 TIMES.

             15  UNIT-IN OCCURS 5 TIMES.
                 20  UNIT-NAME PIC X(5).
                 20  UNIT-NUMBER PIC 9(4).
 PROCEDURE DIVISION.
 100-START-PROCESSING.
*    ************************************************************

*    THIS PERFORM LOGIC IS EXECUTED 250 TIMES BY THE COMPUTER.

*    ************************************************************

     PERFORM ZERO-OUT-BIG-TABLE VARYING SUB1 FROM 1 BY 1
     UNTIL SUB1 > 10
*    SUB1 IS VARIED LAST BY THE COMPUTER.
     AFTER SUB2 FROM 1 BY 1 UNTIL SUB2 > 5
*    SUB2 IS VARIED ******2ND****** BY THE COMPUTER.
     AFTER SUB3 FROM 1 BY 1 UNTIL SUB3> 5.
*    ******SUB3 IS VARIED FIRST BY THE COMPUTER******
     PERFORM ADDRESS-THE-VARIABLES THRU ATV-EXIT.
     DISPLAY 'VARIABLE TEST-IT = ' TEST-IT.
     STOP RUN.
 ZERO-OUT-BIG-TABLE.
     MOVE ZEROS TO UNIT-IN (SUB1, SUB2, SUB3).
 ADDRESS-THE-VARIABLES.
     IF UNIT-NUMBER OF UNIT-IN OF SECTION-IN OF DIVISION-IN
     OF COMPANY-TABLE (3, 4, 5) = 0 ADD 1 TO TEST-IT.

 ATV-EXIT. EXIT.
```

Note: The procedures run by a PERFORM statement are in effect a closed subroutine that can be entered from other points in the program.

The Format 4 PERFORM statement is especially useful in table handling. One Format 4 PERFORM statement can serially search an entire 3-dimensional table.

Segmentation Information

A PERFORM statement appearing in a permanent segment can have in its range only one of the following:

- Sections, each of which has a segment number less than 50
- Sections or paragraphs wholly contained in a single independent segment.

A PERFORM statement that appears in an independent segment can have in its range only one of the following:

- Sections, each of which has a segment number less than 50
- Sections or paragraphs wholly contained within the same independent segment as the PERFORM statement.

Control is passed to the performed procedures only once each time the PERFORM statement is run.

STOP Statement

The STOP statement halts the object program either temporarily or permanently.

Format

```
STOP { RUN     }
     { literal }
```

The literal can be numeric or nonnumeric and any figurative constant except the ALL literal. If the literal is numeric, it must be an unsigned integer.

When you specify the STOP statement with a literal, the literal is displayed at the user program display station if the program has an attached display station, or at the system console if there is no attached display station and the running of the object program is suspended. The program resumes running only after you intervene.

Your action determines whether the job continues at the next statement that can be run in the sequence, the job step is canceled, or the entire job is canceled.

When STOP RUN is specified, the program currently running ends, and control returns to the system. If a STOP RUN statement is in a sequence of imperative statements, it must be the last or the only statement in the sequence. You should close all files before a STOP RUN statement.

An implicit return to the calling program is always generated after the last statement in the source program. In a main program, this is equivalent to a STOP RUN. In a subprogram, this is equivalent to an EXIT PROGRAM.

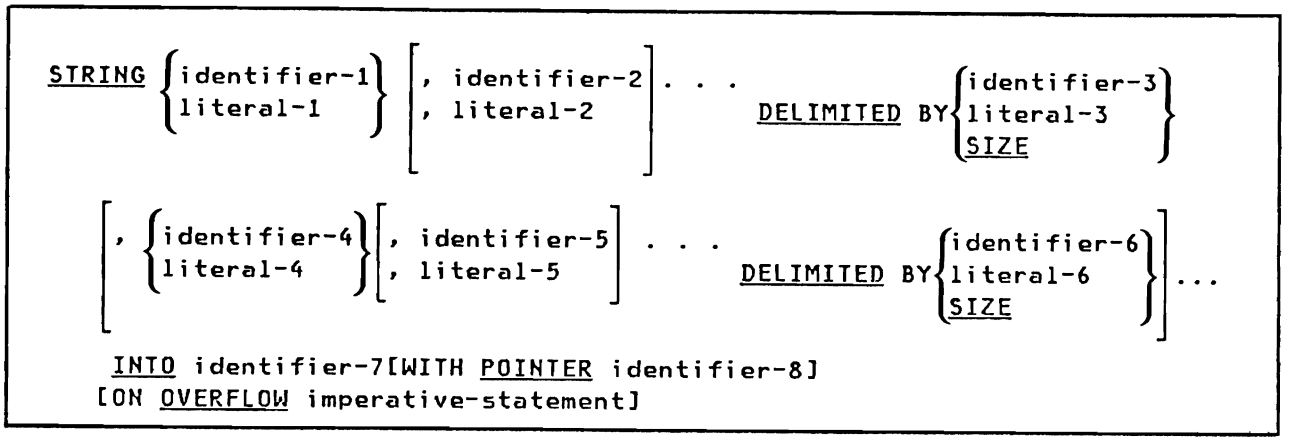
For restrictions on the STOP RUN statement in calling and called programs, see *System Dependent Considerations* in Chapter 2.

Note: The STOP literal statement is useful for special situations when you need to intervene when the program is running.

STRING Statement

The STRING statement lets you join the partial or complete contents of two or more data items into a single data item.

Format



Each literal must be a nonnumeric literal; each may be any figurative constant without the optional word ALL. When you specify a figurative constant, it is considered a 1-character nonnumeric literal.

All identifiers except identifier-8 (the POINTER item) must have a USAGE of DISPLAY, explicitly or implicitly.

The sending fields are identifier-1, identifier-2, identifier-4, identifier-5, or their corresponding literals.

The receiving field is identifier-7, which must be an elementary alphanumeric item without editing symbols and without the JUSTIFIED clause in its description.

The delimiters are identifier-3, identifier-6, or their corresponding literals, or the key word SIZE. The delimiters specify the character(s) delimiting the data to be transferred; when SIZE is specified, the complete sending area is transferred.

When the sending field or any of the delimiters are elementary numeric items, you must describe them as integers, and their PICTURE character strings must not contain the symbol P.

The pointer field is identifier-8, which must be an elementary integer data item large enough to contain a value equal to the length of the receiving area plus 1. The pointer field must not contain the symbol P in its PICTURE character string.

Running the STRING Statement

When the STRING statement is run, data is transferred from the sending fields to the receiving field. Sending fields are processed in the order in which they are specified. The following rules apply:

- Characters from the sending fields are transferred to the receiving field according to the rules for alphanumeric to alphanumeric elementary moves except that no space filling is provided.

- When you specify the DELIMITED BY identifier/literal, the contents of each sending item are transferred character by character beginning with the leftmost and continuing until either a delimiter for this sending field is reached (the delimiter itself is not transferred) or the rightmost character of this sending field has been transferred.

- When you specify DELIMITED BY SIZE, each sending field is transferred in its entirety to the receiving field.

- When the receiving field is filled or when all the sending fields have been processed, the operation is ended.

- When you specify the POINTER phrase, an explicit pointer field is available to the COBOL user to control placement of data in the receiving field. You must set the explicit pointer's initial value to any value from 1 through the character count value of the receiving field. The pointer field must be defined as large enough to contain a value equal to the length of the receiving field plus 1; this precludes arithmetic overflow when the system updates the pointer at the end of the transfer.

- When you do not specify the POINTER phrase, a pointer is not available to you; an implicit pointer with an initial value of 1 is used by the system.

- When the STRING statement is run, the initial pointer value (explicit or implicit) points to the first character position within the receiving field into which data is to be transferred. Beginning at that position, data is then positioned character by character from left to right. After each character is positioned, the explicit or implicit pointer is incremented by 1. The value in the pointer field is changed only in this manner. At the end of processing, the pointer value always indicates one character beyond the last character transferred into the receiving field.

- If, at any time during or after a STRING statement has begun running, the pointer value (explicit or implicit) is less than 1 or exceeds a value equal to the length of the receiving field, no more data is transferred into the receiving field, and if specified, the ON OVERFLOW imperative statement is run. The ON OVERFLOW statement is not run unless there was an attempt to move in one or more characters beyond the end of identifier-7.

- If you do not specify the ON OVERFLOW phrase, control passes to the next runnable statement when the preceding conditions occur.

After the STRING statement is run, only that part of the receiving field into which data was transferred is changed. The rest of the receiving field contains the data that was present before the STRING statement was run.

Figure 11-20 illustrates the rules for the STRING statement.

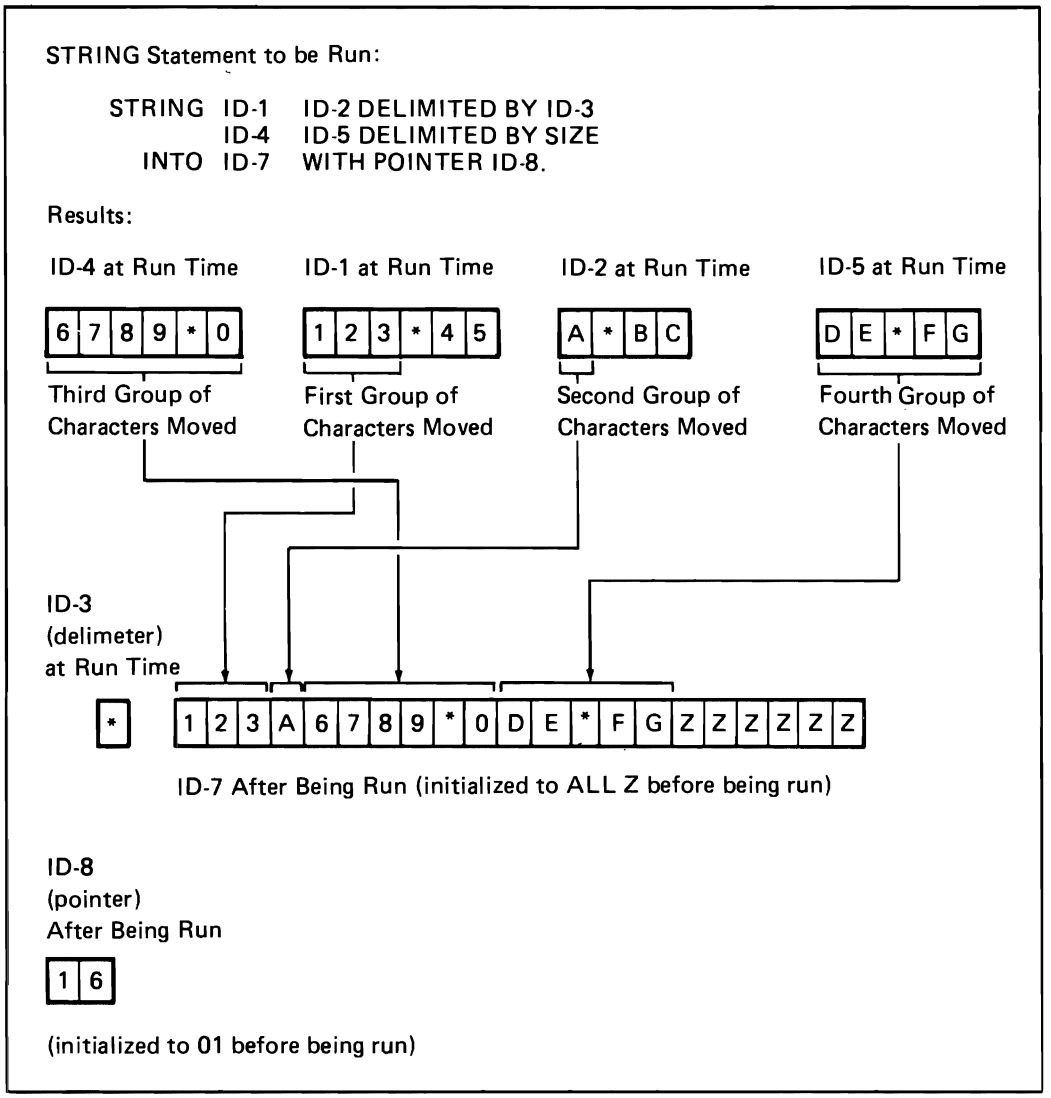


Figure 11-20. STRING Statement Results

STRING Statement Example

The following example illustrates some of the considerations that apply to the STRING statement.

In the Data Division, you have defined the following fields:

```
01  RPT-LINE      PICTURE X(120).
01  LINE-POS      PICTURE 99.
01  LINE-NO       PICTURE 9(5) VALUE 1.
01  DEC-POINT     PICTURE X VALUE '.'.
```

In the File Section, you have defined the following input record:

```
01  RCD-01.
    05  CUST-INFO.
        10 CUST-NAME   PICTURE X(15).
        10 CUST-ADDR   PICTURE X(34).
    05  BILL-INFO.
        10 INV-NO      PICTURE X(6).
        10 INV-AMT     PICTURE $$,$$$.99.
        10 AMT-PAID    PICTURE $$,$$$.99.
        10 DATE-PAID   PICTURE X(8).
        10 BAL-DUE     PICTURE $$,$$$.99.
        10 DATE-DUE    PICTURE X(8).
```

Suppose you want to construct an output line consisting of portions of the information from RCD-01. The line is to consist of a line number, a customer name and address, an invoice number, a date due, and a balance due, truncated to the dollar figure shown.

The record as read in contains the following information:

J.B. SMITHbbbbb
444bSPRINGbST.,bCHICAGObILL.bbbbb
A14275
$4,736.85
$2,400.00
09/22/76
$2,336.85
10/22/76

In the Procedure Division, you initialize RPT-LINE to SPACES and set LINE-POS (which is to be used as the POINTER field) to 4. Then you issue this STRING statement:

```
STRING LINE-NO SPACE CUST-INFO

    SPACE INV-NO SPACE DATE-DUE
    SPACE DELIMITED BY SIZE
    BAL-DUE DELIMITED BY DEC-POINT
    INTO RPT-LINE WITH POINTER LINE-POS.
```

When the statement is run, the following actions take place:

1. The field LINE-NO is moved into positions 4 through 8 of RPT-LINE.
2. A space is moved into position 9.
3. The group item CUST-INFO is moved into positions 10 through 58.
4. A space is moved into position 59.
5. INV-NO is moved into positions 60 through 65.
6. A space is moved into position 66.
7. DATE-DUE is moved into positions 67 through 74.
8. A space is moved into position 75.
9. The portion of BAL-DUE that precedes the decimal point is moved into positions 76 through 81.

After the STRING statement has been run, RPT-LINE appears as shown in Figure 11-21.

Note: You can write one STRING statement instead of a series of MOVE statements.

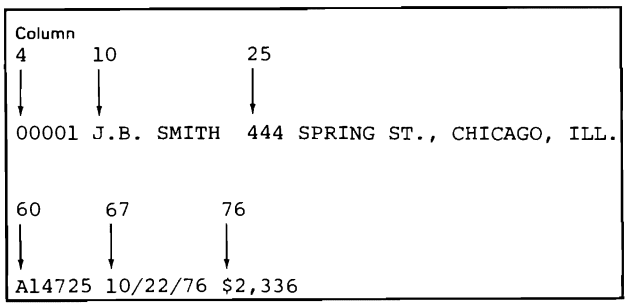


Figure 11-21. STRING Statement Example Output Data

SUBTRACT Statement

The SUBTRACT statement causes either one numeric item or the sum of two or more numeric items to be subtracted from one or more numeric items and the result to be stored. The formats of the SUBTRACT statement are:

Format 1

```
SUBTRACT {identifier-1} [, identifier-2] ... FROM identifier-3 [ROUNDED]
         {literal-1   } [, literal-2   ]

    [, identifier-4 [ROUNDED]] ...[ON SIZE ERROR imperative-statement]
```

Format 2

```
SUBTRACT {identifier-1} [, identifier-2] ... FROM {identifier-3}
         {literal-1   } [, literal-2   ]         {literal-3   }

   GIVING identifier-4 [ROUNDED] [, identifier-5 [ROUNDED]] ...

   [ON SIZE ERROR imperative-statement]
```

Format 3

```
SUBTRACT {CORRESPONDING} identifier-1 FROM identifier-2 [ROUNDED]
         {CORR         }
    [ON SIZE ERROR imperative-statement]
```

In formats 1 and 2, each identifier except those following the key word GIVING must name an elementary numeric item. In format 2, each identifier following the key word GIVING must name a numeric elementary or numeric edited elementary item. In format 3, each identifier must name a group item. In all formats, each literal must be a numeric literal.

In format 1, all identifiers or literals preceding the key word FROM are added together, and this sum is subtracted from and stored immediately in identifier-3, and then, if specified, subtracted from and stored immediately in identifier-4, and so on.

In format 2, all identifiers or literals preceding the key word FROM are added together and this sum is subtracted from identifier-3 or literal-3. The result of the subtraction is stored as the new value of identifier-4, and, if specified, identifier-5, and so on.

In format 3, elementary data items within identifier-1 are subtracted from and stored in the corresponding elementary data items within identifier-2.

If the total length of the operands is 18 digits or less, the compiler ensures that enough places are carried so that no significant digits are lost.

Note: For all three formats of the subtract statement, if identifier-3 and identifier-4 are the same data item, and identifier-1 and identifier-3 are both negative, truncation can occur.

ROUNDED Phrase

After decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is compared with the number of places provided for the fraction of the resultant identifier.

If the size of the fractional result exceeds the number of places provided for its storage, truncation occurs unless the ROUNDED phrase is specified. When the ROUNDED phrase is specified, the least-significant digit of the resultant identifier has its value increased by 1 whenever the most-significant digit of the excess is greater than or equal to 5.

When the resultant identifier is described by a PICTURE clause containing rightmost Ps, and when the number of places in the calculated result exceeds the number of integer positions specified, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

SIZE ERROR Phrase

A size error condition exists if after decimal point alignment, the value of a result exceeds the largest value that can be contained in the resultant field. In the SUBTRACT statement, the size error condition applies only to final results.

If you specify the ROUNDED phrase, rounding takes place before size error checking.

When a size error occurs, the subsequent action of the program depends on whether or not the SIZE ERROR phrase is specified.

If you do not specify the SIZE ERROR phrase and a size error condition occurs, the value of the affected resultant identifier is unpredictable. When you specify multiple receivers, those that do not have a size error are not affected by receivers that do have the error.

If you specify the SIZE ERROR phrase and a size error condition occurs, the error results are not placed in the receiving identifier. When the arithmetic operation is completed, the imperative statement in the SIZE ERROR phrase is run.

GIVING Phrase

If you specify the GIVING phrase, the value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation. Because this identifier is not involved in the computation, it can be a numeric edited item.

CORRESPONDING Phrase

The CORRESPONDING phrase lets operations be performed on elementary items of the same name. You simply specify the group items to which the elementary items belong.

The abbreviation CORR is equivalent to the key word CORRESPONDING.

Both identifiers following the key word CORRESPONDING must name group items. In this discussion, these identifiers are referred to as d1 and d2.

A pair of subordinate data items, one from d1 and one from d2, correspond if the following conditions are true:

- Both of the subordinate items are elementary numeric data items.
- The two subordinate items have the same name and the same qualifiers up to but not including d1 and d2.
- The subordinate items are not identified by the key word FILLER.
- The subordinate items do not include a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause in their descriptions; if such a subordinate item is a group, the items subordinate to it are also ignored. However, d1 and d2 themselves can contain or be subordinate to items containing a REDEFINES or OCCURS clause in their descriptions.

  For example, two data hierarchies are defined as follows:

```
05 ITEM-1 OCCURS 6 INDEXED BY X.
   10 ITEM-A ...
   10 ITEM-B ...
   10 ITEM-C REDEFINES ITEM-B ...
05 ITEM-2
   10 ITEM-A ...
   10 ITEM-B ...
   10 ITEM-C ...
```

  If you specify SUBTRACT CORR ITEM-2 FROM ITEM-1(X), ITEM-A and ITEM-A(X) and ITEM-B and ITEM-B(X) are considered to be corresponding. Thus, ITEM-A and ITEM-B of ITEM-2 are subtracted from ITEM-A and ITEM-B of ITEM-1(X). ITEM-C and ITEM-C(X) are not included because ITEM-C(X) includes a REDEFINES clause in its data description. ITEM-1 is valid as either d1 or d2.

- Neither d1 nor d2 is described as a level-66, -77 or -88 item, or as a FILLER or USAGE IS INDEX item.

UNSTRING Statement

The UNSTRING statement causes consecutive data in a sending field to be separated and placed into multiple receiving fields.

Format

```
UNSTRING identifier-1
  [ DELIMITED BY [ALL] { identifier-2 } [ , OR [ALL] { identifier-3 } ] ... ]
                       { literal-1   }               { literal-2   }
   INTO identifier-4[, DELIMITER IN identifier-5][, COUNT IN identifier-6]
   [, identifier-7[, DELIMITER IN identifier-8][, COUNT IN identifier-9]]...
  [WITH POINTER identifier-10][TALLYING IN identifier-11]
  [ON OVERFLOW imperative-statement]
```

Each literal must be a nonnumeric literal; each can be any figurative constant except the ALL literal. When you specify a figurative constant, it is considered to be a 1-character nonnumeric literal.

Sending Field

identifier-1 is the sending field. It must be an alphanumeric data item. Data is transferred from this field to the receiving fields.

DELIMITED BY Phrase: This phrase specifies delimiters within identifier-1 that control the data transfer.

The delimiters are identifier-2, identifier-3, or their corresponding literals. Each identifier or literal you specify represents one delimiter. You can specify no more than 15 delimiters, and each must be an alphanumeric data item.

If a delimiter contains 2 or more characters, it is recognized in the sending field only if the delimiter characters are consecutive and, in the sequence specified,in the delimiter item.

When you specify two or more delimiters, an OR condition exists, and each nonoverlapping occurrence of any one of the delimiters is recognized in the sending field in the sequence specified. For example, if you specify DELIMITED BY AB OR BC, either AB or BC in the sending field is considered a delimiter. An occurrence of ABC is considered an occurrence of AB, and the search for another delimiter resumes with C.

When you do not specify the DELIMITED BY ALL phrase and two or more consecutive delimiters are encountered, the current data receiving field is filled with spaces or 0's according to the description of the data receiving field.

When you specify the DELIMITED BY ALL phrase, one or more consecutive occurrences of any delimiter are treated as if they were only one occurrence, and this one occurrence is moved to the delimiter receiving field, if specified. The delimiting characters in the sending field are treated as an elementary alphanumeric item and are moved into the current delimiter receiving field according to the rules of the MOVE statement.

You can specify the DELIMITER IN and COUNT IN phrases only if you have specified the DELIMITED BY phrase.

Data Receiving Fields

identifier-4, identifier-7, and so on, are the data receiving fields and must have a USAGE of DISPLAY. These fields can be defined as:

- Alphabetic (without the symbol B in the PICTURE string)
- Alphanumeric
- Numeric (without the symbol P in the PICTURE string).

You must not define these fields as alphanumeric edited or numeric edited items. Data is transferred to these fields from the sending field.

DELIMITER IN Phrase: The delimiter receiving fields are identifier-5, identifier-8, and so on. These identifiers must be alphanumeric.

COUNT IN Phrase: The data-count fields for each data transfer are identifier-6, identifier-9, and so on. Each field holds the count of delimited characters in the sending field to be transferred to this receiving field; the delimiters are not included in this count.

POINTER Phrase: The pointer field is identifer-10: it contains a value that indicates the relative starting position in the sending field. When you specify this phrase, you must initialize this field to a value of at least 1 and not greater than the count of the sending field, before the UNSTRING statement is run.

TALLYING Phrase: The field count is identifier-11; it is incremented by the number of data receiving fields acted upon when the UNSTRING statement is run. When you specify this phrase, you must initialize it before the UNSTRING statement is run.

The data-count fields, the pointer field, and the field-count field must each be integer items without the symbol P in the PICTURE character strings.

Running the UNSTRING Statement

When you initiate the UNSTRING statement, the current data receiving field is identifier-4. Data is transferred from the sending field to the current data receiving field according to the following rules:

- If you do not specify the POINTER phrase, the sending field character string is examined, beginning with the leftmost character. If you specify the POINTER phrase, the field is examined beginning at the relative character position specified by the value in the pointer field.

- If you specify the DELIMITED BY phrase, the examination proceeds left to right character by character until a delimiter is encountered. If the end of the sending field is reached before a delimiter is found, the examination ends with the last character in the sending field.

- If you do not specify the DELIMITED BY phrase, the number of characters examined is equal to the size of the current data receiving field, depending upon its data category:

  - If the receiving field is alphanumeric or alphabetic, the number of characters examined is equal to the number of characters in the current receiving field.
  - If the receiving field is numeric, the number of characters examined is equal to the number of characters in the integer portion of the current receiving field.
  - If the receiving field is described with the SIGN IS SEPARATE clause, the characters examined are one fewer than the size of the current receiving field.
  - If the receiving field is described as a variable-length data item, the number of characters examined is determined by the current size of the current receiving field.

- The examined characters, excluding any delimiter characters, are treated as an alphanumeric elementary item, and are moved into the current data receiving field according to the rules for the MOVE statement.

- If you specify the DELIMITER IN phrase, the delimiting characters in the sending field are treated as an elementary alphanumeric item and are moved to the current delimiter receiving field according to the rules for the MOVE statement. If the delimiting condition is the end of the sending field, the current delimiter receiving field is filled with spaces.

- If you specify the COUNT IN phrase, a value equal to the number of examined characters, excluding any delimiters, is moved into the data count field, according to the rules for an elementary move.

- If you specify the DELIMITED BY phrase, the sending field is further examined, beginning with the first character to the right of the delimiter.

- If you do not specify the DELIMITED BY phrase, the sending field is further examined, beginning with the first character to the right of the last character examined.

- After data is transferred to the first data receiving field (identifier-4), the current data receiving field becomes identifier-7. For each succeeding current data receiving field, the preceding procedure is repeated, either until all of the characters in the sending field have been transferred or until there are no more unfilled data receiving fields.

- When you specify the POINTER phrase, the contents of the pointer field is incremented by 1 for each examined character in the sending field. When the UNSTRING statement is completed, the pointer field contains a value equal to its initial value plus the number of characters examined in the sending field.

- When you specify the TALLYING phrase, and the UNSTRING statement is completed, the tallying identifier contains a value equal to the initial value plus the number of data receiving areas acted upon; this count includes any null fields.

- When an overflow condition exists, the UNSTRING statement stops running. If you have specified the ON OVERFLOW phrase, that imperative statement is run. If you have not specified the ON OVERFLOW phrase, control passes to the next statement that can be run. An overflow condition exists when:

  - An UNSTRING statement is initiated and the value in the pointer field is less than 1 or greater than the length of the sending field.
  - Or, all data receiving fields have been acted upon by the UNSTRING statement, and the sending field still contains unexamined characters.

If you subscript or index any of the UNSTRING statement identifiers, the subscripts and indexes are evaluated as follows:

- Any subscripting or indexing associated with the sending field, the pointer field, or the field-count field is evaluated only once, immediately before any data is transferred.

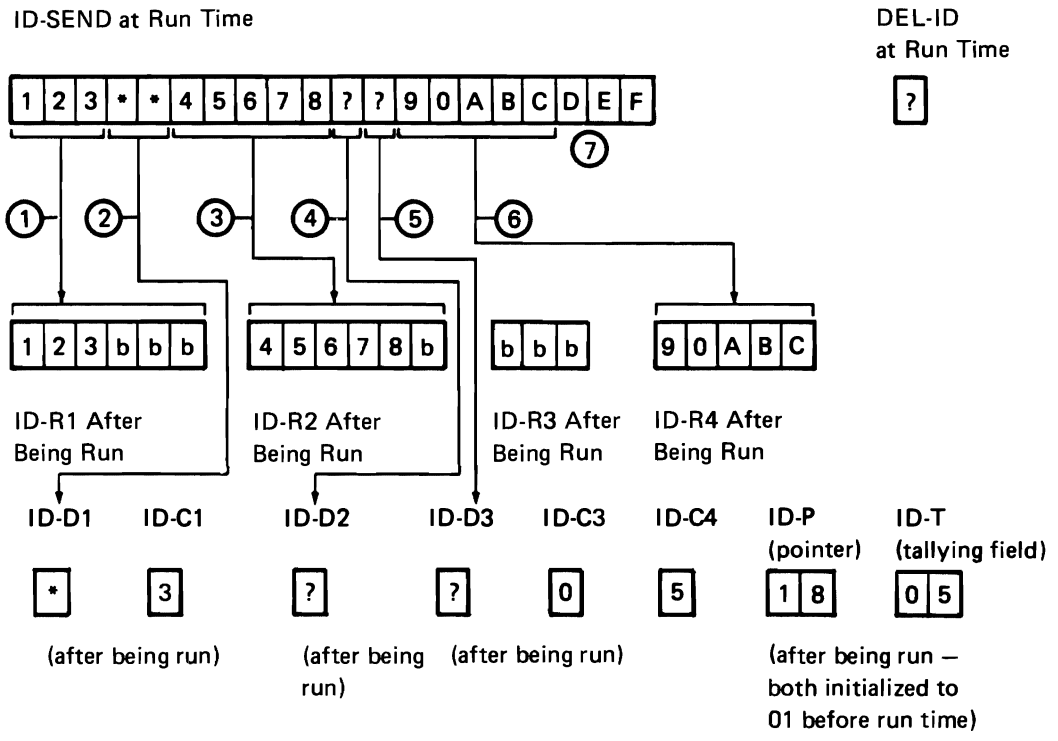
- Any subscripting or indexing associated with the delimiters, the data and delimiter receiving fields, or the data-count fields, is evaluated immediately before the transfer of data into the affected data item.

Figure 11-22 illustrates the rules for the UNSTRING statement.

The following UNSTRING statement has the results shown:

```
UNSTRING  ID-SEND DELIMITED BY DEL-ID OR ALL '*'
     INTO  ID-R1 DELIMITER IN ID-D1 COUNT IN ID-C1
           ID-R2 DELIMITER IN ID-D2
           ID-R3 DELIMITER IN ID-D3 COUNT IN ID-C3
           ID-R4 COUNT IN ID-C4
     WITH POINTER ID-P
     TALLYING IN ID-T
     ON OVERFLOW GO TO OFLOW-EXIT.
```

(All the data receiving fields are defined as alphanumeric)



The run sequence is:

1. Three characters are placed in ID-R1.
2. Because ALL * is specified, one * is placed in ID-D1.
3. Five characters are placed in ID-R2.
4. A ? is placed in ID-D2. The current receiving field is now ID-R3.
5. A ? is placed in ID-D3; ID-R3 is filled with spaces; no characters are transferred, so 0 is placed in ID-C3.
6. No delimiter is encountered before 5 characters fill ID-R4; 5 is placed in ID-C4.
7. ID-P is updated to 18; ID-T is updated to 05. There are still untransferred characters in ID-SEND, so the ON OVERFLOW exit is taken.

Figure 11-22. UNSTRING Statement Results

UNSTRING Statement Example

The following example illustrates some of the considerations that apply to the UNSTRING statement.

In the Data Division, you have defined the following input record to be acted upon by the UNSTRING statement:

```
01  INV-RCD.
    05  CONTROL-CHARS    PIC XX.
    05  ITEM-INDENT      PIC X(20).
    05  FILLER           PIC X.
    05  INV-CODE         PIC X(10).
    05  FILLER           PIC X.
    05  NO-UNITS         PIC 9(6).
    05  FILLER           PIC X.
    05  PRICE-PER-M      PIC 99999.
    05  FILLER           PIC X.
    05  RTL-AMT          PIC 9(6).99.
```

You have defined the next two records as receiving fields for the UNSTRING statement. DISPLAY-REC is to be used for printed output. WORK-REC is to be used for further internal processing.

```
01  DISPLAY-REC.
    05  INV-NO           PIC X(6).
    05  FILLER           PIC X VALUE SPACE.
    05  ITEM-NAME        PIC X(20).
    05  FILLER           PIC X VALUE SPACE.
    05  DISPLAY-DOLS     PIC 9(6).

01  WORK-REC.
    05  M-UNITS          PIC 9(6).
    05  FIELD-A          PIC 9(6).
    05  WK-PRICE
        REDEFINES
        FIELD-A           PIC 9999V99.
    05  INV-CLASS        PIC X(3).
```

You have also defined the following fields for use as control fields in the UNSTRING statement.

```
77 DBY-1          PIC X, VALUE IS '.'.
77 CTR-1          PIC 99, VALUE IS ZERO.
77 CTR-2          PIC 99, VALUE IS ZERO.
77 CTR-3          PIC 99, VALUE IS ZERO.
77 CTR-4          PIC 99, VALUE IS ZERO.
77 DLTR-1         PIC X.
77 DLTR-2         PIC X.
77 CHAR-CT        PIC 99, VALUE IS 3.
77 FLDS-FILLED    PIC 99, VALUE IS ZERO.
```

In the Procedure Division, you have written the following UNSTRING statement to move subfields of INV-RCD to the subfields of DISPLAY-REC and WORK-REC:

```
UNSTRING INV-RCD DELIMITED BY
ALL SPACES OR '/' OR DBY-1
INTO ITEM-NAME COUNT IN CTR-1
INV-NO DELIMITER IN DLTR-1 COUNT IN CTR-2
INV-CLASS
M-UNITS COUNT IN CTR-3
FIELD-A
DISPLAY-DOLS DELIMITER IN
DLTR-2 COUNT IN CTR-4
WITH POINTER CHAR-CT
TALLYING IN FLDS-FILLED
ON OVERFLOW GO TO UNSTRING-COMPLETE.
```

Before the UNSTRING statement is issued, you place the value 3 in the CHAR-CT (the POINTER item), so as not to work with the 2 control characters at the beginning of INV-RCD. In DBY-1, you place a period as a delimiter, and in FLDS-FILLED (the TALLYING item) you place the value 0. The following data is then read into INV-RCD as shown in Figure 11-23.

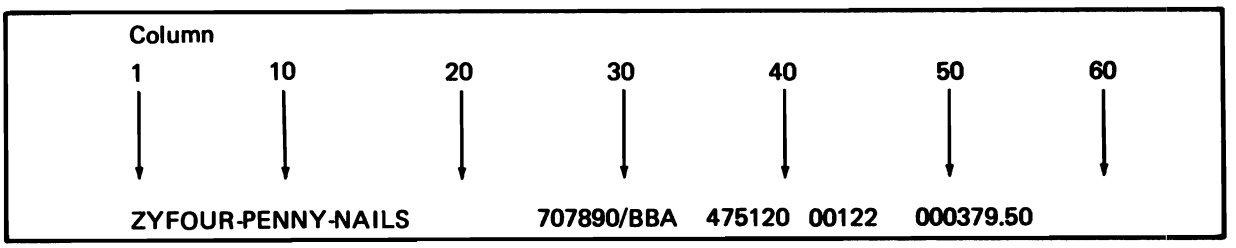


Figure 11-23. UNSTRING Statement Example--Input Data

When the UNSTRING statement is run, the following actions take place:

1. Positions 3 through 18 (FOUR-PENNY-NAILS) of INV-RCD are placed in ITEM-NAME, left-justified within the area, and the unused character positions are padded with spaces. The value 16 is placed in CTR-1.

2. Because you specified ALL SPACES as a delimiter, the 5 consecutive SPACE characters are considered to be one occurrence of the delimiter.

3. Positions 24 through 29 (707890) are placed in INV-NO. The delimiter character / is placed in DLTR-1, and the value 6 is placed in CTR-2.

4. Positions 31 through 33 are placed in INV-CLASS. The delimiter is a SPACE, but because no field has been defined as a receiving area for delimiters, the SPACE is merely bypassed.

5. Positions 35 through 40 (475120) are examined and are placed in M-UNITS. The delimiter is a SPACE, but because you have not defined a receiving field as a receiving area for delimiters, the SPACE is bypassed. The value 6 is placed in CTR-3.

6. Positions 42 through 46 (00122) are placed in FIELD-A and right-justified within the area. The high-order digit position is filled with a 0. The delimiter is a SPACE, but because you have not defined a field as a receiving area for delimiters, the SPACE is bypassed.

7. Positions 48 through 53 (000379) are placed in DISPLAY-DOLS. The period (.) delimiter character is placed in DLTR-2, and the value 6 is placed in CTR-4.

8. Because all receiving fields have been acted upon and 2 characters of data in INV-RCD have not been examined, the ON OVERFLOW exit is taken, and the UNSTRING statement is completed.

When the UNSTRING statement has run, DISPLAY-REC contains the following data:

```
707890 FOUR-PENNY-NAILS 000379
```

WORK-REC contains the following data:

```
475120000122BBA
```

CHAR-CT (the POINTER field) contains the value 55, and FLD-FILLED (the TALLYING field) contains the value 6.

Note: One UNSTRING statement can be written instead of a series of MOVE statements.

USE AFTER EXCEPTION/ERROR Statement (EXCEPTION/ERROR Declarative)

The EXCEPTION/ERROR Declarative specifies procedures for input and output exception or error handling that are to be run in addition to the standard system procedures.

Format

```
USE AFTER STANDARD { EXCEPTION } PROCEDURE ON { file-name-1[, file-name-2] ... } .
                   { ERROR     }              { INPUT                          }
                                              { OUTPUT                         }
                                              { I-O                            }
                                              { EXTEND                         }
```

The words EXCEPTION and ERROR are synonymous and can be used interchangeably.

File-Name Phrase

This phrase is valid for sequential, indexed, relative, and TRANSACTION files. When you specify this phrase, the procedure is run only for the file(s) named. No file name can refer to a sort-merge file. For any given file, you can only specify one EXCEPTION/ERROR procedure. For example, if an input file is specifically named in one EXCEPTION/ERROR procedure, there must not also be an EXCEPTION/ERROR procedure for all INPUT files.

INPUT Phrase

This phrase is valid for sequential, indexed, and relative files. When you specify this phrase, the procedure is applicable to all files opened in INPUT mode.

OUTPUT Phrase

This phrase is valid for sequential, indexed, and relative files. When you specify this phrase, the procedure is applicable to all files opened in OUTPUT mode.

I-O Phrase

This phrase is valid for sequential, indexed, relative, and TRANSACTION files. When you specify this phrase, the procedure is applicable to all files opened in I-O mode.

EXTEND Phrase

This phrase is valid for sequential files only. When you specify this phrase, the procedure applies to all files opened in EXTEND mode.

General Considerations

The EXCEPTION/ERROR procedure is run when one of the following conditions exists:

- After completing the standard system input/output error routine
- Upon recognition of an INVALID KEY or AT END condition when you have not specified an INVALID KEY or AT END phrase in the input/output statement
- When status key 1 is not equal to 0 following an I/O operation.

After the EXCEPTION/ERROR procedure has run, control returns to the statement immediately following the input/output statement that caused the error.

The EXCEPTION/ERROR procedure is performed when an input/output error occurs during a READ, WRITE, REWRITE, START, DELETE, OPEN, CLOSE, ACQUIRE, or DROP statement. For example, the procedure is activated when an input/output statement fails on a file that is in the open status.

The EXCEPTION/ERROR procedure is not performed when a CLOSE statement fails because the file is already closed.

Within a Declarative procedure, there must be no reference to any nondeclarative procedure. In the nondeclarative portion of the program, there must be no reference to procedure names that appear in an EXCEPTION/ERROR Declarative procedure, except that PERFORM statements may refer to an EXCEPTION/ERROR procedure or to procedures associated with it.

All input/output statements for a file must have an error handling routine. If you do ***not*** specify an AT END or INVALID KEY phrase, then you ***must*** specify an EXCEPTION/ERROR procedure.

Within an EXCEPTION/ERROR Declarative procedure, no statement can be run that causes the running of a USE procedure that has been previously invoked and has not yet returned control to the invoking routine.

IBM Extension

TRANSACTION File Considerations

In an EXCEPTION/ERROR Declarative for the TRANSACTION file, only the file-name or I-O phrases are allowed. All other phrases and all rules are the same as those for any EXCEPTION/ERROR Declarative for any file.

End of IBM Extension

Note: EXCEPTION/ERROR procedures can be used to check the status key values whenever an input/output error occurs.

Care should be used when you specify EXCEPTION/ERROR procedures for any file. Prior to successful completion of an initial OPEN for any file, the current

Declarative has not yet been established by the program; if any other I/O statement is run for a file that has never been opened, no Declarative can receive control. If this file has been previously opened, the last previously established Declarative procedure receives control.

For example, an OPEN OUTPUT statement establishes a Declarative procedure for a file and it is then closed without error. During later processing, if a logic error occurs, control will go to the Declarative procedure established when the file was opened for OUTPUT.

USE FOR DEBUGGING Statement

This statement is discussed under *Debugging Features* in Chapter 6.

Input and Output Statements of the Procedure Division

Chapter 12. Input and Output Statements of the Procedure Division

COBOL input and output statements transfer data to and from files. In COBOL, the unit of data made available to the program is a record. Provision is made for operations such as the movement of data into buffers and internal storage, validity checking, error correction (when feasible), and unblocking and blocking of records.

The description of the file in the Environment Division and the Data Division governs which input and output statements are allowed in the Procedure Division.

There is special processing for deleted records (deleted records are valid only for relative and index files), and there are certain restrictions when using deleted records. For a full explanation of the limitations associated with deleted record processing, see *Initial Considerations* in Chapter 2.

In this chapter, the Procedure Division input and output statements are presented alphabetically. Each statement format is followed by a discussion of its options.

ACCEPT Statement

The function of the ACCEPT statement is to obtain low-volume data from the device assigned as the system input device (SYSIN) or from a display station (SYSLOG) or an SSP-ICF session. The ACCEPT statement causes the transfer of data into the specified identifier. There is no editing or error checking of the incoming data. The formats of the ACCEPT statement are:

Format 1

```
ACCEPT identifier [FROM mnemonic-name]
```

Format 2

```
ACCEPT identifier FROM { DATE }
                       { DAY  }
                       { TIME }
```

Format 3

```
ACCEPT identifier [FROM mnemonic-name]

[FOR {identifier-2}]
     {literal     }
```

Format 1 Considerations

You can use Format 1 to transfer data from an input device to the identifier. The identifier can be a group item, an elementary alphabetic or alphanumeric item, or a numeric data item with USAGE DISPLAY or USAGE COMPUTATIONAL.

If you omit the FROM phrase, the system input device (requesting display station or invoking procedure) is assumed. If the program is invoked by a procedure, a record is read from the procedure for each ACCEPT statement until a /* is encountered. If the records in the procedure are exhausted or if the program is not invoked by a procedure, the requesting display station is used via SYSIN. If end-of-file is encountered in the procedure, the requesting display station is used via SYSIN and the program continues running as if no procedure had been invoked but no notification of the original procedure will be displayed on the status screens.

If you specify the FROM phrase, the mnemonic name must be associated with an input or output device that you have specified in the SPECIAL-NAMES paragraph. The input or output device can be the display station (REQUESTOR) or the system operator's console (SYSTEM-CONSOLE). If the mnemonic name is REQUESTOR and the job is entered by way of the JOBQ Command, the system operator's console is used otherwise, SYSLOG is used and the requester display station is prompted for input.

When the device is the system input device, the following rules apply:

- An input record size of 120 characters is assumed.
- If the identifier is more than 120 characters, characters beyond the length of the identifier are truncated.
- If the identifier is less than 120 characters, succeeding input records are read until the storage area of identifier is filled. If the identifier is not an exact multiple of 120 characters, that part of the last input record that does not fit into the identifier is truncated.

When the device is the display station keyboard, the same rules apply as when the device is the system input device except that the size is 60 characters.

The source of input data is dependent upon the type of program initiation as follows:

Method of Program Initiation	Mnemonic name Associated with System-Console	Mnemonic name Associated with Requestor	Data Source when FROM Option Omitted
JOBQ	System Console	System Console	Data from next record in the procedure. If there is no data in the procedure, the input comes from the system console.
SRT	System Console	Display Station	Display Station
MRT	System Console	System Console	Can produce undesirable results. Specify the FROM option.

Input from the device can be stopped by entering a record beginning with /*. The /* is moved into the ACCEPT identifier with blank padding or truncation on the right. Any subsequent attempt to ACCEPT from the device is in error and no further processing occurs. If the identifier is longer than the device size and the /* is entered for a succeeding input record, the identifier is padded to the right with blanks, and the /* is treated as input to the next ACCEPT from the device.

Format 2 Considerations

You can use Format 2 to transfer the system information (program date and system time) to the identifier, using the rules for the MOVE statement without the CORRESPONDING phrase. The identifier can be a group item, or an elementary alphanumeric, alphanumeric edited, zoned decimal, packed decimal, binary, or numeric edited item. The following discussion concerns the DATE, DAY, and TIME phrases:

- DATE has the implicit PICTURE 9(6) USAGE DISPLAY. The sequence of data elements from left to right is: 2 digits for year of century, 2 digits for month of year, 2 digits for day of month. Thus, July 4, 1982, is expressed as 820704.

  The date is the last date specified in OCL for this job stream, or the current program date if no date has been specified in OCL since sign-on. An MRT program uses the system date at job initialization unless you explicitly specify a date in the OCL for this job stream.

- DAY has the implicit PICTURE 9(5) USAGE DISPLAY. The sequence of data elements from left to right is: 2 digits for year of century, 3 digits for day of year. Thus, July 4, 1982, is expressed as 82186, because July 4 is the 186th day of the year 1982.

- TIME has the implicit PICTURE 9(8) USAGE DISPLAY. The sequence of data elements from left to right is: 2 digits for hour of day, 2 digits for minute of hour, 2 digits for second of minute, 2 digits for hundredths of second. Thus, 2:41 p.m. is expressed as 14410000. The time returned is the time when the ACCEPT statement is run.

  The time is always rounded up to the nearest second; therefore, hundredths of a second are always expressed as 00.

Format 3 Considerations

IBM Extension

Format 3 transfers data from the local data area or from the attribute record to identifier-1.

If the mnemonic name is associated with LOCAL-DATA, the 512-byte local data area associated with the requester display station is moved into identifier-1.

If the mnemonic name is associated with ATTRIBUTE-DATA, identifier-1 must describe an attribute data record. The attributes of the specified symbolic ID are moved into identifier-1. The TRANSACTION file must be open for this request.

The move into identifier-1 for both LOCAL-DATA and ATTRIBUTE-DATA takes place according to the rules for the MOVE statement for an alphanumeric group move without the CORRESPONDING phrase.

The FOR phrase is allowed only when you associate the mnemonic name with either ATTRIBUTE-DATA or LOCAL-DATA. The literal or identifier-2 is the symbolic ID of the display station or SSP-ICF session for which data is retrieved. A symbolic ID of blanks (or none specified) retrieves the attributes or local data from the requester for which an input or output operation was most recently performed. In a program that has no TRANSACTION file, the local data is retrieved from the requester for SRT batch jobs. The symbolic ID must be a 2-character, alphanumeric data item or literal associated with the requester.

Note: If the program is an MRT program, there is a local data area for each requester and an additional local data area for the program. Prior to the successful completion of the first requester's first input or output operation, this MRT local data area can be accessed. A symbolic ID of blanks will return the MRT's local data area.

If the mnemonic name is associated with either SYSTEM-CONSOLE or REQUESTOR, the FOR phrase is not valid.)

End of IBM Extension

ACQUIRE Statement

The ACQUIRE statement attaches a display station or an SSP-ICF session to the TRANSACTION file.

Format

The value of the literal or identifier specifies the symbolic identification of a display station or an SSP-ICF session that is to be associated with the file name. To be acquired, a display station must be in stand-by mode. To acquire an SSP-ICF session, the session identifier must be specified by the SYMID parameter of the OCL SESSION statement for the job step.

If you specify a literal, it must be a 2-character, alphanumeric literal. If you specify an identifier, it must refer to a 2-character, alphanumeric data item.

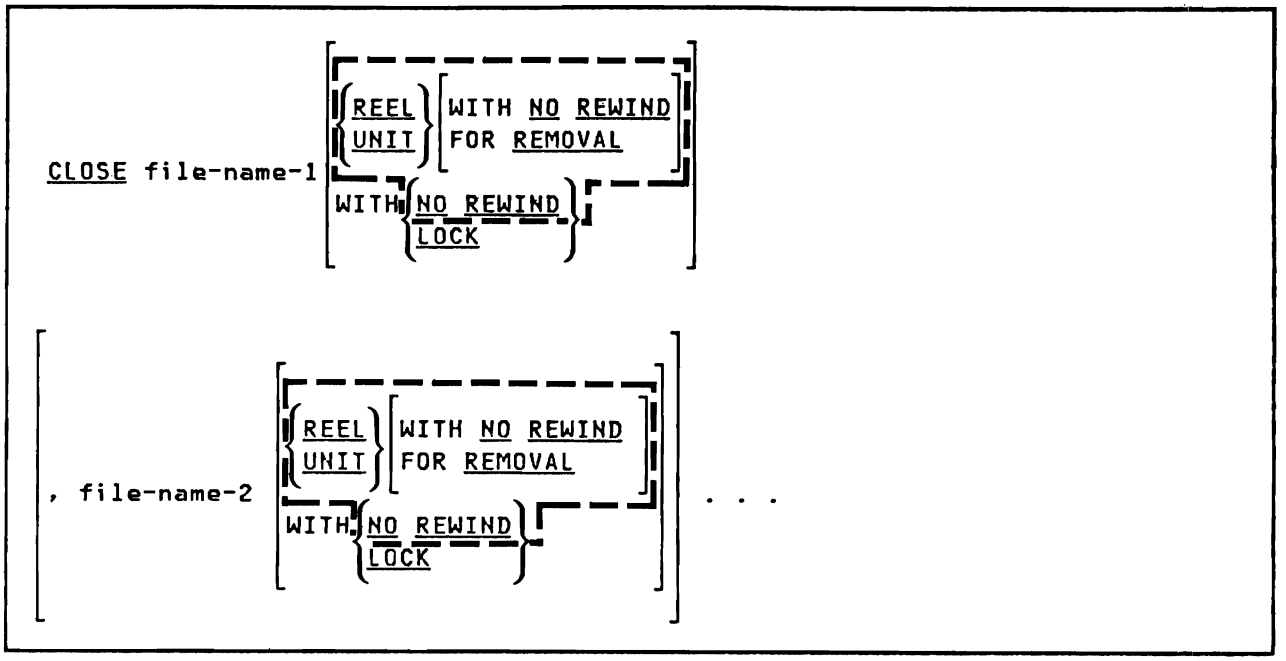
For display stations, the first character must be alphabetic (A through Z). For SSP-ICF sessions, the first character must be numeric (0 through 9) and the second character must be alphabetic (A through Z) or a special character ($, #, or @).

The file name must refer to a file whose organization is TRANSACTION. For additional information on TRANSACTION files and interactive processing, refer to Chapter 7.

CLOSE Statement

The CLOSE statement stops file processing, with an optional lock.

Format



Each file name specifies a file with which the CLOSE statement is used. The files need not have the same organization or access method. The files must not be sort or merge files.

A CLOSE statement can be run only for a file in an open mode. After a CLOSE statement is successfully run, the record area associated with the file name is no longer available. An unsuccessful CLOSE statement makes the record data area undefined.

After a CLOSE statement is successfully run for the file, an OPEN statement for the file must be run before any other input or output statement (except a SORT/MERGE statement with the USING or GIVING phrase) can refer explicitly or implicitly to the file. If you specify the FILE STATUS clause in the file control entry, the associated status key is updated when the CLOSE statement is run.

The first character of the status key is known as status key 1; the second character is known as status key 2. Combinations of possible values and their meanings are shown in Appendix D.

IBM Extension

TRANSACTION File Extended File Status Key: the extended file status key for a TRANSACTION file is four characters long. Characters 1 and 2 contain the ICF major return code; characters 3 and 4 contain the ICF minor return code. ICF return codes are described in the manual *Interactive Communications Feature: Reference*, SC21-7910. See Appendix D in this COBOL manual for a list of status keys and their meanings.

End of IBM Extension

If the file is open and a CLOSE statement is unsuccessfully run, the EXCEPTION/ERROR procedure (if specified) for this file is run. If a CLOSE statement is not run for an open file before a STOP RUN statement for this program is run, results are unpredictable.

When you specify the LOCK option, you ensure that the file cannot be opened again in the program.

The REEL/UNIT option, the FOR REMOVAL option, and the NO REWIND option are treated as comments.

For special considerations concerning spooled printer files, see *Some Initial Considerations* in Chapter 2.

DELETE Statement

The DELETE statement logically removes a record from an indexed or a relative file.

Format

```
DELETE file-name RECORD [INVALID KEY imperative-statement]
```

When the DELETE statement is run, the associated file must be opened in I-O mode. The file also must be created as a delete-capable file. This is done by specifying DFILE-YES on the FILE OCL statement or the BLDFILE procedure when the file is created. For more information on creating delete-capable files, see the manual *System Reference*. You must define the file name in an FD entry in the Data Division, and it must be the name of an indexed or a relative file. After a successful DELETE statement, the record is logically removed from the file and can no longer be accessed. For indexed files, the space that the record occupied cannot be used until the file is copied or reorganized. The DELETE statement does not affect the contents of the record area associated with the file name.

A record may be deleted by using a random access DELETE with no WITH DUPLICATES clause. This record is determined by the contents of the data names defined in the RECORD KEY clause.

Status Key--General Considerations

If you specify the FILE STATUS clause in the file control entry, a value is placed in the specified status key (the 2-character data item named in the FILE STATUS clause) during the running of any request on that file; the value indicates the status of that request. The value is placed in the status key before any EXCEPTION/ERROR Declarative or INVALID KEY/AT END option associated with the request is run.

The first character of the status key is known as status key 1; the second character is known as status key 2. Combinations of possible values and their meanings are shown in Appendix D.

INVALID KEY Condition

The INVALID KEY condition can occur when a DELETE statement is run. When the INVALID KEY condition is recognized, the actions are taken in the following order:

1. If you specify the FILE-STATUS clause in the file control entry, a value is placed into the status key (status key 1 = 2) to indicate an INVALID KEY condition (see Appendix D).

2. If you specify the INVALID KEY option in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any EXCEPTION/ERROR declarative procedure specified for this file is not performed.

3. If you do not specify the INVALID KEY option, but you do specify an EXCEPTION/ERROR declarative procedure for the file, the EXCEPTION/ERROR procedure is performed.

When an INVALID KEY condition occurs, the input or output statement that caused the condition is unsuccessful. If you do not specify the INVALID KEY option for a file, you must specify an EXCEPTION/ERROR procedure. If an error other than an INVALID KEY condition occurs, the EXCEPTION/ERROR procedure is run.

DELETE Statement with Sequential Access Mode

For a file in sequential access mode, the last I/O statement must be a successful READ statement. When the DELETE statement is run, the system logically removes the record retrieved by that READ statement. The current record pointer is not affected by the DELETE statement.

You must not specify the INVALID KEY option for a file in sequential access mode. You should, however, specify an EXCEPTION/ERROR procedure.

DELETE Statement with Random or Dynamic Access Mode

In random or dynamic access mode, DELETE statement results depend on whether the file organization is indexed or relative.

Indexed Files: When the DELETE statement is run in random or dynamic access mode and WITH DUPLICATES is not specified in the RECORD KEY clause, the system logically removes the record identified by the contents of the RECORD KEY data item. If the file does not contain such a record, an INVALID KEY condition exists.

When the DELETE statement is run in random or dynamic access mode and WITH DUPLICATES is specified in the RECORD KEY clause for the file, the last input or output statement for the file must have been a successful READ statement. The record read by that statement is the one that is deleted. The READ statement is required to ensure that the proper record is deleted when there are duplicate records. The INVALID KEY clause must not be specified.

Relative Files: When the DELETE statement is run in random or dynamic access mode, the system logically removes the record identified by the contents of the RELATIVE KEY data item. If the file does not contain such a record, an INVALID KEY condition exists.

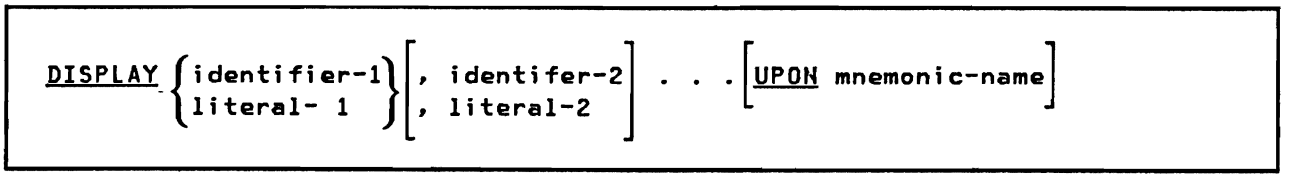
DELETE Statement Considerations

The DELETE statement logically removes the record from the file. For relative files, the space is then available for a new record with the same RELATIVE KEY value. For indexed files, a new record with the same RECORD KEY value can then be added.

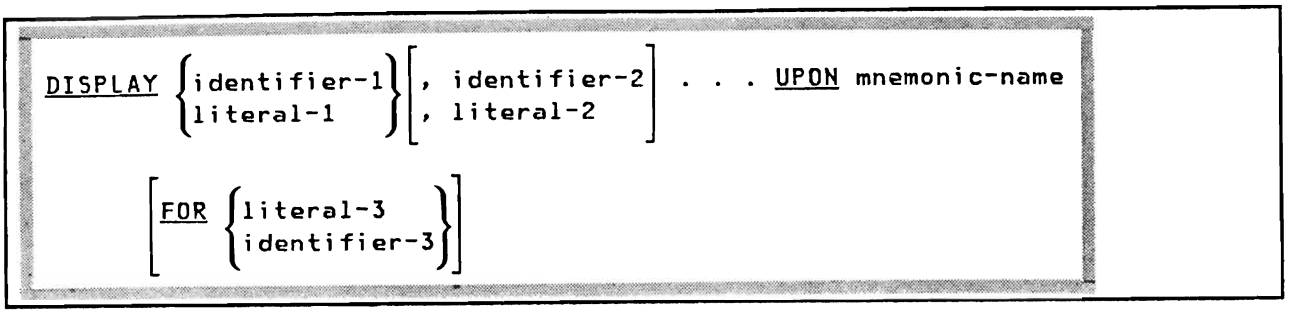
DISPLAY Statement

The DISPLAY statement transfers low-volume data to an output device.

Format 1



Format 2



Format 1 Considerations

The DISPLAY statement transfers the contents of each operand to the output device in the left-to-right order in which the operands are listed. When a DISPLAY statement is run, the data contained in the sending field is transferred to the output device. The size of the sending field is the total character count of all operands listed. If the total character count is less than the device's maximum character count, the remaining rightmost characters are padded with spaces. If the total character count exceeds the maximum, as many records are written as are needed to display all operands. Any operand being printed when the end of a record is reached is continued in the next record.

IBM Extension

Identifiers described as USAGE COMPUTATIONAL-3 or USAGE COMPUTATIONAL-4 are converted to zoned decimal. No other items require conversion. Signed noninteger numeric literals are allowed.

End of IBM Extension

Signed values in numeric fields cause the last character to show both the sign and the number. For example, if you do not specify SIGN WITH SEPARATE CHARACTER and two numeric items have the values -34 and 34, they are displayed as 3M and 34, respectively. If you specify SIGN WITH SEPARATE CHARACTER, a + or a - sign is displayed as either leading or trailing,

depending on how you specified the number. If you specify a figurative constant as one of the operands, only a single figurative constant is displayed.

If you omit the UPON phrase, data is written to the current SYSLIST device. When you specify the UPON phrase, the mnemonic name must be associated in the SPECIAL-NAMES paragraph with either the display station (REQUESTOR) or the system operator's console (SYSTEM-CONSOLE). The maximum logical record size is assumed for each device as follows:

Device	Maximum Logical Record Size
SYSLIST	120 characters
Display station	120 characters
System console	120 characters

The location of the output data is dependent upon the type of program initiation as follows:

Method of Initiation	Mnemonic name Associated with System-Console	Mnemonic name Associated with Requestor	UPON Option Omitted
JOBQ	System console	System console	Current SYSLIST device
SRT	System console	Display station	Current SYSLIST device
MRT	System console	System console	Current SYSLIST device

Format 2 Considerations

IBM Extension

Format 2 of the DISPLAY statement can be used when the mnemonic name is associated with the system name LOCAL-DATA. For a description of the LOCAL-DATA area, see the LOCAL statement in the chapter on OCL statements in the *System Reference*.

Literal-1 or identifier-1 is written to the 512-byte local data area associated with the requestor.

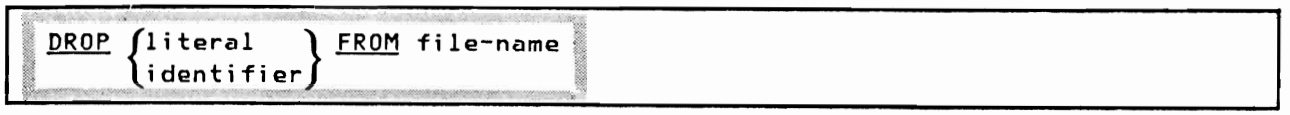
Literal-3 or identifier-3 must be the valid symbolic ID of an attached requester. Identifier-3 must be a 2-character, alphanumeric data item; literal-3 must be a 2-character, nonnumeric literal.

End of IBM Extension

DROP Statement

The DROP statement releases a display station or an SSP-ICF session from its association with the TRANSACTION file.

Format



The value of the literal or identifier specifies the symbolic identification of the attached display station or SSP-ICF session that is to be released.

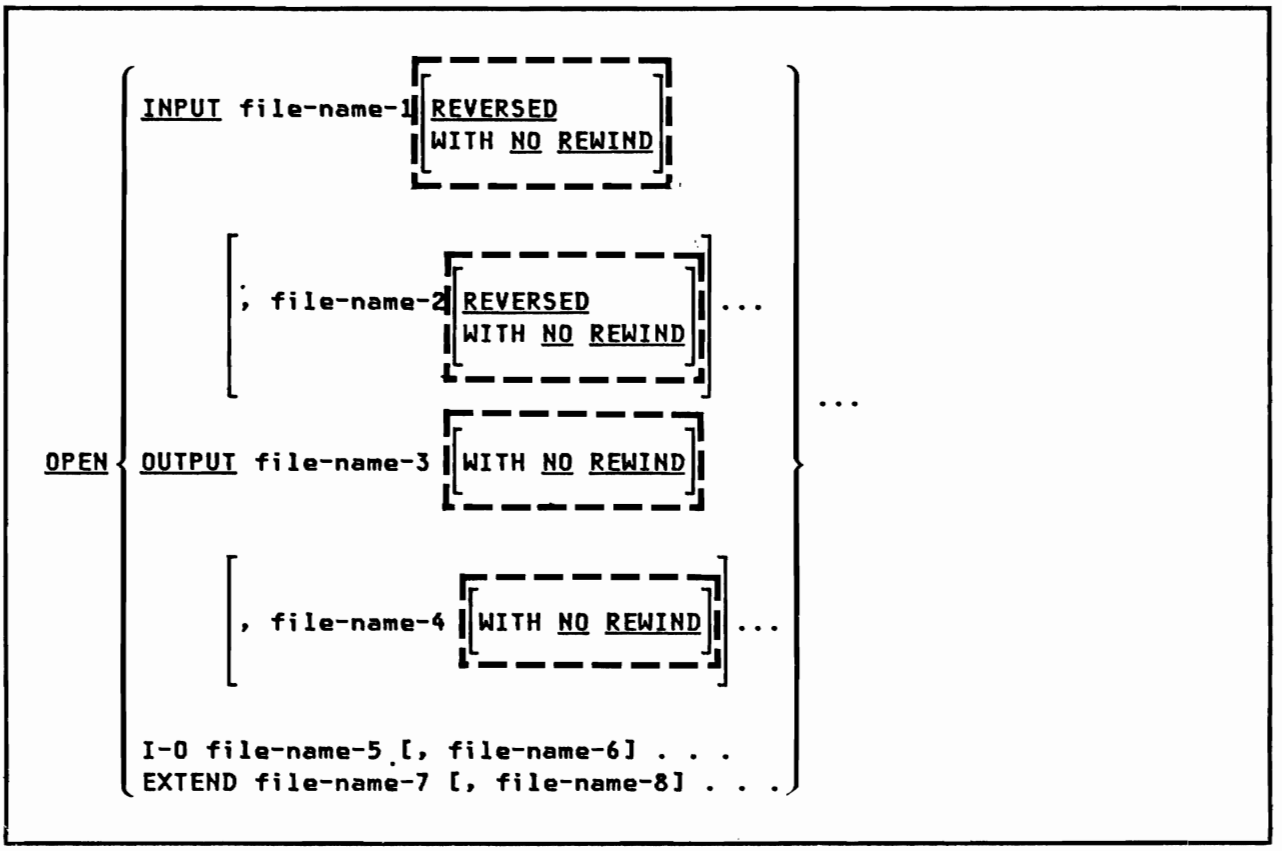
If you specify a literal, it must be a 2-character alphanumeric literal. If you specify an identifier, it must refer to a 2-character, alphanumeric data item.

You can only use the DROP statement with a TRANSACTION file. When a TRANSACTION file is closed or is at the end of a program, all attached display stations and SSP-ICF sessions are implicitly released.

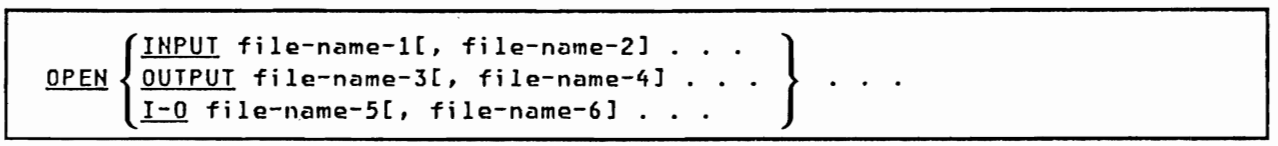
OPEN Statement

The OPEN statement initiates the processing of files. The format of the OPEN statement is as follows:

Format 1--Sequential Files



Format 2--Indexed and Relative Files



Format 3--Transaction Files

Each file name specifies a file with which the OPEN statement is used. The files need not have the same organization or access method. Each file name must be defined in an FD entry in the Data Division; it must not name a sort or merge file.

A successful OPEN statement determines the availability of the file and places that file in the open mode. Before the OPEN statement is successfully run for a given file, no statement can be run, except for a SORT or MERGE statement with the USING or GIVING phrase, that refers explicitly or implicitly to that file. A successful OPEN statement makes the associated record area available to the program; it does not obtain or release the first data record.

You must specify at least one of the phrases (INPUT, OUTPUT, I-O, or EXTEND). You can specify more than one file name in each phrase. The INPUT, OUTPUT, I-O, or EXTEND phrases can appear in any order.

The INPUT phrase lets you open the file for input operations. The I-O phrase lets you open the file for both input and output operations. You can specify the I-O phrase only for disk storage or TRANSACTION files. You must not specify the INPUT or I-O phrases when the file has not been created.

The OUTPUT phrase lets you open the file for output operations. You can specify this phrase only when the file is being created. You must not specify the OUTPUT phrase for a file that contains records or that did contain records that have been deleted. All printer files used in COBOL programs are opened for OUTPUT.

Note: The FILE OCL statement for an output file must contain a DISP-NEW parameter for proper processing.

If the FILE OCL statement for an output file contains a DISP-OLD, then OPEN will delete the file and open a new file by the same name as the data names defined in the RECORD KEY clause. This applies only to indexed or alternative indexed files.

The EXTEND phrase is valid only for sequential files and lets you open the file for output operations. It is discussed under *Format 1--Sequential Files* later in this chapter.

You can open a file for INPUT, OUTPUT, I-O, or EXTEND in the same program. After the first OPEN statement runs for a given file, each subsequent OPEN statement must be preceded by a successful CLOSE statement without the LOCK phrase.

Note: System error diagnostics will be issued if there is a difference between the key length as defined in the program and the key length as found in the index file or the alternative index file.

If you specify the FILE STATUS clause in the file control entry, the associated status key is updated when the OPEN statement is run.

The first character of the status key is known as status key 1; the second character is known as status key 2. Combinations of possible values and their meanings are shown in Appendix D.

IBM Extension

For indexed files, a file status of 95 is returned after a successful OPEN statement when:

- The WITH DUPLICATES phrase is specified and the index to the file does not have the duplicates attribute.
- The WITH DUPLICATES phrase is not specified and the index to the file has the duplicates attribute.

Processing files when either of these conditions exist can cause unpredictable results.

If an OPEN statement is not successful, the EXCEPTION/ERROR procedure (if specified) for this file is run.

End of IBM Extension

IBM Extension

TRANSACTION File Extended File Status Key: The extended file status key for a TRANSACTION file is four characters long. Characters 1 and 2 contain the ICF major return code; characters 3 and 4 contain the ICF minor return code. ICF return codes are described in the manual *Interactive Communications Feature: Reference*. See Appendix D in this COBOL manual for a list of status keys and their meanings.

End of IBM Extension

Both the REVERSED phrase and the NO REWIND phrase are treated as comments.

Current Record Pointer

The current record pointer identifies which record is accessed by a sequential input request. The OPEN statement positions the current record pointer at the first record in the file.

The concept of the current record pointer has no meaning for files that are accessed randomly, TRANSACTION files, or output files. The current record pointer is not used for random only retrieval of input records or for output files.

Format 1--Sequential Files

The EXTEND phrase lets you open the file for output operations. When an OPEN EXTEND statement is run, the file is prepared for the addition of records immediately following the last record in the file. Subsequent WRITE statements add records as if the file had been opened in OUTPUT mode. You can specify the EXTEND phrase when a file is being created. You can also specify the EXTEND phrase for a file that contains records or that did contain records that have been deleted.

The EXTEND phrase has no meaning for a printer file, and the file is opened for output.

The OPEN INPUT or OPEN I-O statement sets the current record pointer to the first record existing in the file. If no records exist in the file, the current record pointer is set so that when the first READ statement is run, an AT END condition results.

For an input file, if you specify SELECT OPTIONAL in the file control entry, the OPEN statement causes the program to check for the presence or absence of this file. If the file is absent, the first READ statement for this file causes the AT END condition to occur.

For special considerations concerning spooled printer files, see *Some Initial Considerations* in Chapter 2.

Figure 12-1 shows the open modes for sequential files and the statements that are permitted with each mode.

	Open Mode			
Statement	**Input**	**Output**	**Input/Output**	**Extend**
READ	X		X	
WRITE		X		X
REWRITE			X	

Figure 12-1. Sequential File Open Modes, and Permissible Statements

Format 2--Indexed and Relative Files

When the OPEN INPUT or OPEN I-O statement is run, the current record pointer is set to the first record existing in the file; the record with the lowest record key value (indexed file) or lowest relative record number (relative file) is considered to be the first record in the file. If no records exist in the file, the current record pointer is set so that the first READ statement results in an AT END condition.

Format 3-TRANSACTION Files

IBM Extension

A TRANSACTION file must be opened with the I-O phrase.

End of IBM Extension

Figure 12-2 shows the open modes for indexed and relative files, and the statements that are permitted with each mode.

File Access Mode	Statement	Open Mode		
		Input	Output	Input/Output
Sequential	READ	X		X
	WRITE		X	
	REWRITE			X
	START	X		X
	DELETE			X
Random	READ	X		X
	WRITE		X	X
	REWRITE			X
	START			
	DELETE			X
Dynamic	READ	X		X
	WRITE		X	X
	REWRITE			X
	START	X		X
	DELETE			X

Figure 12-2. Indexed and Relative File Open Modes, and Permissible Statements

READ Statement

The READ statement makes a record available to the object program.

For sequential access, the READ statement makes available the next logical record from a disk file. For random access, the READ statement makes available a specified record from a disk file. When the READ statement is performed, the associated file must be open in the INPUT or I-O mode. The formats of the READ statement are as follows:

Format 1--Sequential Access (Sequential Files)

```
READ file-name RECORD [INTO identifier] [AT END imperative-statement]
```

Format 2--Sequential Access (Relative and Indexed Files)

```
READ file-name [NEXT] RECORD [INTO identifier]
     [AT END imperative-statement]
```

Format 3--Random Access (Relative Files)

```
READ file-name RECORD [INTO indentifier]
   [INVALID KEY imperative-statement]
```

Format 4--Random Access (Indexed Files)

```
READ file-name RECORD [INTO identifier]

     [INVALID KEY imperative-statement]
```

Format 5--Indexed File Extensions (Dynamic Access Only)

```
READ  file-name [FIRST]  RECORD [INTO  identifier]
                [LAST ]
                [PRIOR]
    [AT END  imperative-statement]
```

Format 6--Sequential Access (TRANSACTION File)

```
READ file-name RECORD
    [INTO identifier-1] [TERMINAL IS {identifier-2}]
                                     {literal-1   }

    [NO DATA imperative-statement-1]

    [AT END imperative-statement-2]
```

The file name must be defined in a Data Division FD entry and must not name a sort or merge file. If more than one record description entry is associated with the file name, these records automatically share the same storage area; that is, they are implicitly redefined. Before a READ statement is performed, the storage area is filled with blanks.

After a READ statement is performed, only those data items within the range of the current record are replaced; data items stored beyond that range are blanks. Figure 12-3 illustrates this concept. If no data items are defined, the entire record will be blank.

The FD entry for a disk file is: FD INPUT-FILE LABEL RECORDS STANDARD. 01 RECORD-1 PICTURE X(30). 01 RECORD-2 PICTURE X(20).
After RECORD-1 is read, the input area contains: ABCDEFGHIJKLMNOPQRSTUVWXYZ1234
If RECORD-2 consists of: 01234567890123456789
After RECORD-2 is read, the input area contains: 01234567890123456789 (Characters in the input area following RECORD-2 are set to blank.)

Figure 12-3. A READ Statement Example with Multiple Record Descriptions

If you specify the FILE STATUS clause in the file control entry, the associated status key is updated when the READ statement is run.

The first character of the status key is known as status key 1; the second character is known as status key 2. Combinations of possible values and their meanings are shown in Appendix D.

IBM Extension

TRANSACTION File Extended File Status Key: The extended file status key for a TRANSACTION file is four characters long. Characters 1 and 2 contain the ICF major return code; characters 3 and 4 contain the ICF minor return code. ICF return codes are described in the manual *Interactive Communications Feature: Reference*. See Appendix D of this COBOL manual for a list of status keys and their meanings.

End of IBM Extension

Current Record Pointer

The current record pointer identifies which record will be accessed by a sequential input request.

For a sequential READ statement, the following considerations apply:

- If an OPEN or a START statement positioned the current record pointer, the record identified by the current record pointer is made available.
- If a previous READ statement positioned the current record pointer, the current record pointer is updated to point to the next existing record in the file; that record is then made available.

The concept of the current record pointer has no meaning for files that are accessed randomly, TRANSACTION files, or output files. The current record pointer is not used for random retrieval of input records or for output files.

Following an unsuccessful READ or START statement, the contents of the associated record area and the position of the current record pointer are undefined.

INTO Identifier Phrase

The identifier you specify must be the name of an entry in the Working-Storage Section or the Linkage Section or of a record description for another previously opened file. The file name and identifier must not refer to the same storage area.

The INTO identifier phrase makes a READ statement equivalent to:

```
READ file-name RECORD.

MOVE record-name TO identifier.
```

After a successful READ statement, the contents of the current record become available both in the record name and the identifier.

When you specify the INTO identifier phrase, the current record is moved from the input area to the identifier area according to the rules for the MOVE statement without the CORRESPONDING phrase. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is transferred to identifier.

You must not specify the INTO identifier phrase when the file contains records of various sizes, as indicated by their record descriptions.

Format 1 and Format 2--Sequential Access

You must use formats 1 and 2 for all files in sequential access mode and also for files in dynamic access mode when record retrieval is sequential. A format 2 READ statement makes available the next logical record from the file. The record that is considered next depends upon the file organization.

NEXT RECORD Phrase: The next record is the succeeding logical record in key sequence. For indexed files, the key sequence is the ascending values of the RECORD KEY. For relative files, the key sequence is the ascending values of relative record numbers for records that exist in the file.

Before the READ statement is performed, the current record pointer must be set by a successful OPEN, START, or READ statement. When the READ statement is performed, the record indicated by the current record pointer is made available, if it is still accessible through the path indicated by the current record pointer. If the record is no longer accessible (for example, as a result of deletion of the record), the current record pointer is updated to indicate the next existing record in the file, and that record is made available.

For a file that allows duplicate keys (the extension, WITH DUPLICATES, is specified in the RECORD KEY clause), the duplicate records are retrieved in a first-in, first-out sequence.

For files in sequential access mode, the NEXT phrase can, but need not, be specified.

If you specify the RELATIVE KEY clause in the file control entry for sequentially accessed relative files, the READ or START statement updates the RELATIVE KEY data item to indicate the relative record number of the record being made available.

AT END Condition: If no succeeding logical record exists in the file when the READ statement is performed, an AT END condition occurs, and the READ statement is unsuccessful. The following actions are taken, in the following order:

1. If you specify the FILE STATUS clause, the status key is updated to indicate an AT END condition.

2. If you specify the AT END phrase, control is transferred to the AT END imperative statement. Any EXCEPTION/ERROR procedure for this file is not performed.

3. If you do not specify the AT END phrase, any EXCEPTION/ERROR procedure for this file is performed.

When the AT END condition is recognized, the position of the current record pointer is undefined, and a sequential access READ statement for this file must not be run without one of the following first being run:

- A successful CLOSE statement followed by a successful OPEN statement
- A successful START statement for this file
- A successful random access READ statement for this file (dynamic access).

If an error other than an AT END condition occurs, the EXCEPTION/ERROR condition is run.

Format 3 and Format 4--Random Access

You must specify format 3 or 4 for indexed and relative files in random access mode and also for files in the dynamic access mode when record retrieval is random.

INVALID KEY Condition: The INVALID KEY condition can occur when a random access READ statement is run. When the INVALID KEY condition is recognized, the actions are taken in the following order:

1. If you specify the FILE-STATUS clause in the file control entry, a value of 2 is placed in status key 1 to indicate an INVALID KEY condition (see Appendix D).

2. If you specify the INVALID KEY phrase in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any EXCEPTION/ERROR declarative procedure specified for this file is not performed.

3. If you do not specify the INVALID KEY phrase, but you do specify an EXCEPTION/ERROR declarative procedure for the file, the EXCEPTION/ERROR procedure is performed.

When an INVALID KEY condition occurs, the input or output statement that caused the condition is unsuccessful. If you do not specify the INVALID KEY phrase for a file, you must specify an EXCEPTION/ERROR procedure. If an

error other than an INVALID KEY condition occurs, the EXCEPTION/ERROR procedure is run.

The manner in which the READ statement is performed depends on the file organization, as explained in following sections.

Files with Relative Organization: The format 3 READ statement sets the current record pointer to the record whose relative record number is contained in the RELATIVE KEY data item and makes that record available. If the file does not contain such a record, the INVALID KEY condition exists, the position of the current record pointer is undefined, and the READ statement is unsuccessful.

Files with Indexed Organization: The format 4 READ statement causes the value of the RECORD KEY data item to be compared with the value of the corresponding key data item in the file records until the first record having an equal value is found. The current record pointer is positioned to this record, which is then made available. If no record can be identified, an INVALID KEY condition exists, the position of the current record pointer is undefined, and the READ statement is unsuccessful.

IBM Extension

For a file that allows duplicate keys (WITH DUPLICATES is specified in the RECORD KEY clause), only the first record in a series of records with duplicate keys can be retrieved when ACCESS IS RANDOM is specified in the file control entry. Each duplicate record must be retrieved using sequential READ statements. Therefore, it is recommended that files with duplicate keys be processed with sequential or dynamic access.

End of IBM Extension

READ Statement with Dynamic Access Mode

For files with indexed or relative organization, you can specify a dynamic access mode in the file control entry. In dynamic access mode, you can specify either sequential or random record retrieval, depending on the format used.

For sequential retrieval, you must specify format 2 with the NEXT phrase. All other rules for sequential access apply.

For random retrieval, you must specify format 3 or 4. All other rules for random access apply.

IBM Extension

For a file that allows duplicate keys (WITH DUPLICATES specified in the RECORD KEY clause), sequential retrieval must be used to read any record with a duplicate key except the first such record.

End of IBM Extension

Each successful sequential or random READ updates the current record pointer to the logical record.

Format 5 - Indexed File Extensions (Dynamic Access Only)

IBM Extension

READ FIRST, LAST or PRIOR may be used for indexed files with DYNAMIC access and with a device type of DATABASE.

For records where the current record pointer is indicating the first existing record in the file, then if PRIOR is specified, an End-Of-File (EOF) condition will exist.

If READ PRIOR is issued when the current record pointer is undefined, a "no current record pointer for I/O request" condition will be met. For example, if READ PRIOR is issued after an end of file condition is encountered, then this error will occur.

For files which are empty (null file), an End-Of-File condition will exist for ***any*** READ operation.

In all remaining incidences:

- If FIRST is specified, the current record pointer is updated to indicate the first existing record in the file and that logical record is made available.
- If LAST is specified, the current record pointer is updated to indicate the last existing record in the file and that logical record is made available.
- If PRIOR is specified, and the current record pointer is not positioned at the first record in the file, then the current record pointer is updated to indicate the preceding existing record and that logical record is made available.

If you READ past the end of the file, or run any other operation that produces a file status of 94 (no current record pointer for I/O request), then you cannot recover by using READ PRIOR. You must specify a START statement, READ FIRST, READ LAST, or a RANDOM READ statement to recover.

Format 6 - TRANSACTION Files

Format 6 must be used for the TRANSACTION file. The format 6 READ statement makes a record available from the TRANSACTION file.

The TRANSACTION file must be open in the I-O mode at the time the READ statement is performed.

A successful READ statement fills in the terminal ID and function key fields of the CONTROL-AREA, if specified.

TERMINAL Phrase: The record to be made available by a READ statement is determined as follows:

- If you specify the TERMINAL phrase, the data record is made available either from identifier-2 or from literal-1 when identifier-2 is blank. Identifier-2 or literal-1 must be the symbolic ID of an attached display station or an SSP-ICF session. Identifier-2 must be a 2-character, alphanumeric data item; literal-1 must be a 2-character, nonnumeric data item. When both identifier-2 and literal-1 are blank, the READ statement is performed as though the TERMINAL phrase were omitted.

- When you omit the TERMINAL phrase, the default values are as follows:

  - a single display station or SSP-ICF session is attached to the file, the default value is that display station or SSP-ICF session.
  - multiple display stations or SSP-ICF sessions, or both, are attached to the file, there is no default value. The data record made available is the first record input from any attached display station or SSP-ICF session.

Note: Use of the TERMINAL phrase forces the next input to come from the specified display station or SSP-ICF session, unless literal-1 and identifier-2 contain blanks.

NO DATA Phrase: When you specify the NO DATA phrase, the imperative statement specified is performed if a record cannot immediately be made available at the time the READ statement is run. After the imperative statement is run, the next sequential statement is performed.

When you do not specify the NO DATA phrase, the program waits until a record becomes available.

AT END Condition: The AT END condition occurs when there are no attached display stations or SSP-ICF sessions for which an invite is outstanding when an input operation (READ) was requested, and the program is not an NEP (never-ending, or long-running, program). The AT END condition occurs for an NEP when there are no attached display stations or SSP-ICF sessions with an outstanding invite, and a STOP SYSTEM command has been entered.

Input is implicitly invited with each WRITE statement, but can be suppressed by an option on the $SFGR format or selected SSP-ICF predefined formats. When the AT END condition occurs, the READ statement is unsuccessful, and imperative-statement-2 is performed.

End of IBM Extension

Additional information can be found under the *READ Statement, AT END Phrase,* in Chapter 7.

REWRITE Statement

The REWRITE statement logically replaces an existing record in a disk file. When the REWRITE statement is performed, the associated disk file must be open in I-O mode.

Format

```
REWRITE record-name [FROM identifier] [INVALID KEY imperative-statement]
```

The record name must be the name of a logical record in the File Section of the Data Division. You must not associate the record name with a sort or merge file. You can qualify the record name, but you must not subscript or index it. The number of character positions in the record name must equal the number of character positions in the record being replaced.

The REWRITE statement replaces an existing record in the file with the information contained in the record name.

After a successful REWRITE statement, the logical record is no longer available in the record name unless the associated file is named in a SAME RECORD AREA clause. In this case, the record is also available as a record of the other files named in the SAME RECORD AREA clause.

The current record pointer is not affected when the REWRITE statement is run.

The data item(s) specified in the RECORD KEY clause must be given a value prior to the running of the WRITE statement.

Note: Alternative index considerations:

- *A duplicate key error is produced if an index is set over a sequential file or relative file and duplicate keys are not allowed. This error would result even if a program access to the file is sequential or relative when attempting to WRITE or REWRITE a record which happens to have the same KEY value as an existing record.*
- *Any WRITE to the alternative index file built over a relative file will result in an invalid operation error status.*

If you specify the FILE STATUS clause in the file control entry, the associated status key is updated when the REWRITE statement is run.

The first character of the status key is known as status key 1; the second character is known as status key 2. Combinations of possible values and their meanings are shown in Appendix D.

FROM Identifier Phrase

The identifier you specify must be the name of an entry in the Working-Storage Section or the Linkage Section or of a record description for another previously opened file. The record name and the identifier must not refer to the same storage area. When the FROM identifier phrase is specified, an implicit move operation is performed according to MOVE statement rules without the CORRESPONDING phrase.

The FROM identifier phrase makes a REWRITE statement equivalent to:

```
MOVE identifier TO record-name

REWRITE record-name.
```

After a successful REWRITE statement, the current record may no longer be available in the record name, but is still available in the identifier.

INVALID KEY Condition

The INVALID KEY condition can occur when a REWRITE statement is run. When the INVALID KEY condition is recognized, the actions are taken in the following order:

1. If you specify the FILE-STATUS clause in the file control entry, a value of 2 is placed in status key 1 to indicate an INVALID KEY condition (see Appendix D).

2. If you specify the INVALID KEY phrase in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any EXCEPTION/ERROR declarative procedure specified for this file is not performed.

3. If you do not specify the INVALID KEY phrase, but you do specify an EXCEPTION/ERROR declarative procedure for the file, the EXCEPTION/ERROR procedure is performed.

When an INVALID KEY condition occurs, the input or output statement that caused the condition is unsuccessful. If you do not specify the INVALID KEY phrase for a file, you must specify an EXCEPTION/ERROR procedure. If an error other than an INVALID KEY condition occurs, the EXCEPTION/ERROR procedure is run.

REWRITE Statement for Sequential Files

For files accessed sequentially, the last successful input or output statement for the file must be a READ statement. When the REWRITE statement is run, the record retrieved by that READ statement is logically replaced.

You must not specify the INVALID KEY phrase for a file with sequential organization. You must specify an EXCEPTION/ERROR procedure.

The record replaced is specified by the value contained in the RECORD KEY clause. The following condition holds if the indexed file has sequential access, random access, or DUPLICATE keys allowed; the last I/O operation to the file must be a READ and the RECORD KEY must not change between the READ and the REWRITE.

IBM Extension

If the WITH DUPLICATES phrase is specified in the RECORD KEY clause for the file, the last input or output statement for the file must have been a successful READ statement. The record read by that statement is the one that is replaced. The READ statement is required to ensure that the proper record is replaced when there are duplicates.

You should not change the primary key value for the record between the READ and subsequent REWRITE statements. If this is done, a File Status code 21 (sequence error) is issued.

End of IBM Extension

If the file is accessed randomly or dynamically and WITH DUPLICATES is not specified in the RECORD KEY clause, any record referenced by the RECORD KEY clause is rewritten.

IBM Extension

Note: You cannot change the value of the primary key through an alternative index. Thus, you cannot change the value of a record which is physically associated with the primary index of a file.

The record to be rewritten is determined by using the value(s) found in the data item(s) defined in the RECORD KEY clause. When using RANDOM or DYNAMIC access, the WITH DUPLICATES phrase is not specified in the RECORD KEY clause.

End of IBM Extension

An INVALID KEY condition exists when the access mode is sequential or is random or dynamic and the WITH DUPLICATES phrase was specified in the RECORD KEY clause, and the value contained in the RECORD KEY of the record to be replaced does not equal the RECORD KEY data item of the last record retrieved from the file.

If this condition exists, the INVALID KEY imperative statement is performed, the REWRITE statement is unsuccessful, the updating operation does not take place, and the data in the record name is unaffected.

REWRITE Statement for Relative Files

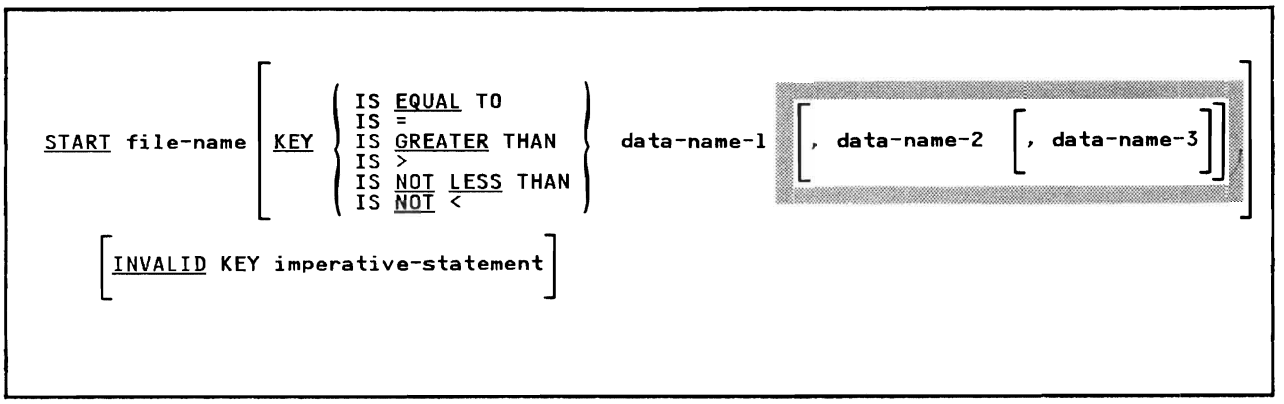
For relative files in the sequential access mode, you must not specify the INVALID KEY phrase. You must specify an EXCEPTION/ERROR procedure.

When the access mode is random or dynamic, you specify the record to be replaced in the RELATIVE KEY data item. If the file does not contain the record specified, an INVALID KEY condition exists, and if specified, the INVALID KEY imperative statement is performed. The updating operation does not take place, and the data in record-name is unaffected.

START Statement

The START statement provides a means of positioning within an indexed or relative file for subsequent sequential record retrieval. When the START statement is performed, the associated indexed or relative file must be open in INPUT or I-O mode.

Format



The file name must name a file with sequential or dynamic access. You must define the file name in an FD entry in the Data Division, and you must not name a sort or merge file.

KEY Phrase

When you do not specify the KEY phrase, the EQUAL TO relational operator is implied.

When you specify the KEY phrase, the comparison specified in the KEY relational operator is made between data-name-1 and the corresponding key field associated with the file's records. Data-name-1, data-name-2, and data-name-3 can be qualified; they cannot be subscripted or indexed. The current record pointer is positioned to the logical record in the file with a matching key field.

Data-name-1 must refer to one of the following:

- The data-item named in the RELATIVE KEY clause associated with file-name.
- The data-item named in the RECORD KEY clause.
- A data-item in the file referred to by file-name, and that has the same starting position as the RECORD KEY, and a length less than or equal to that of the RECORD KEY.

IBM Extension

When you specify the KEY phrase and more than one data name is used, the data names (data-name-1, data-name-2, and data-name-3) are interpreted as combining to form a noncontiguous key; provided there is more than one data name in the RECORD KEY clause of the ASSIGN statement, the device type DATABASE is used, and the file is an INDEXED file.

For any data names used to form the key, except the last (rightmost) one, the starting location and length must be the same as described for the file in the RECORD KEY clause. For the last data name only, the starting location must be the same as described for the file in the RECORD KEY clause; the length can be equal to or less than the length of the corresponding data name in the RECORD KEY clause. This last case (when the length of the last data item in the key phrase is less than the length of the corresponding data name in the RECORD KEY clause) represents a partial key.

Note: The name of any data name in the KEY phrase of the START statement need not be the same as the corresponding RECORD KEY data name. Only the positions and the size (except for the case described above) are critical.

End of IBM Extension

INVALID KEY Condition

The INVALID KEY condition can occur when a START statement is run. When the INVALID KEY condition is recognized, the actions are taken in the following order:

1. If you specify the FILE-STATUS clause in the file control entry, a value of 2 is placed in status key 1 to indicate an INVALID KEY condition (see Appendix D).

2. If you specify the INVALID KEY phrase in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any EXCEPTION/ERROR declarative procedure specified for this file is not performed.

3. If you do not specify the INVALID KEY phrase, but you do specify an EXCEPTION/ERROR declarative procedure for the file, the EXCEPTION/ERROR procedure is performed.

When an INVALID KEY condition occurs, the input or output statement that caused the condition is unsuccessful. If you do not specify the INVALID KEY phrase for a file, you must specify an EXCEPTION/ERROR procedure. If an error other than an INVALID KEY condition occurs, the EXCEPTION/ERROR procedure is run.

If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists; the position of the current record pointer is undefined, and the INVALID KEY imperative statement, if specified, is performed.

If you specify the FILE STATUS clause in the file control entry, the associated status key is updated by the START statement.

The first character of the status key is known as status key 1; the second character is known as status key 2. Combinations of possible values and their meanings are shown in Appendix D.

START Statement for Indexed Files

When you do not specify the KEY phrase, the key data item used for the EQUAL TO comparison is the RECORD KEY.

When you specify the KEY phrase, the key data item used for the comparison is the data name, which must be either:

- The RECORD KEY.
- An alphanumeric data item subordinate to the RECORD KEY whose leftmost character position corresponds to the leftmost character position of the RECORD KEY. This data item can be qualified.

The current record pointer identifies which record will be accessed by a sequential input request. The START statement positions the current record pointer at the first record in the file that satisfies the implicit or explicit comparison specified in the START statement.

If the operands in the comparison are of unequal length, the comparison proceeds as if the longer field were truncated on the right to the length of the shorter field. All other numeric and nonnumeric comparison rules apply except that the PROGRAM COLLATING SEQUENCE clause, if specified, has no effect.

The concept of the current record pointer has no meaning for random access files or output files.

START Statement for Relative Files

When you specify the KEY phrase, the data name must be the RELATIVE KEY data item.

Whether or not you specify the KEY phrase, the key data item used in the comparison is the RELATIVE KEY data item. The current record pointer is positioned to the logical record in the file whose key satisfies the comparison.

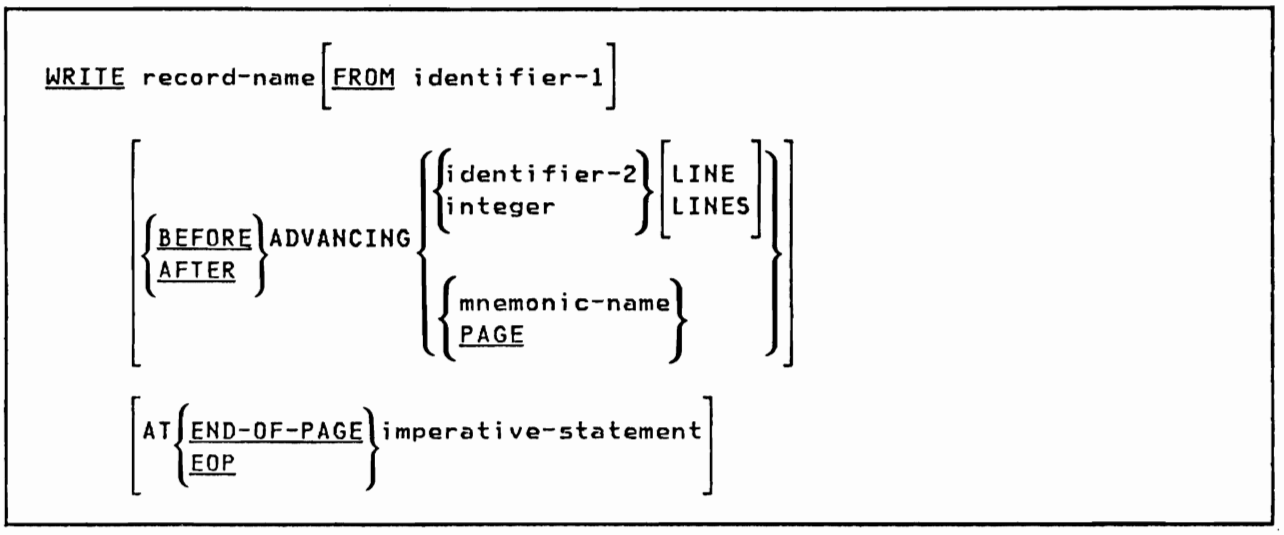
WRITE Statement

The WRITE statement releases a logical record for an output or input/output file. You can specify a WRITE statement for:

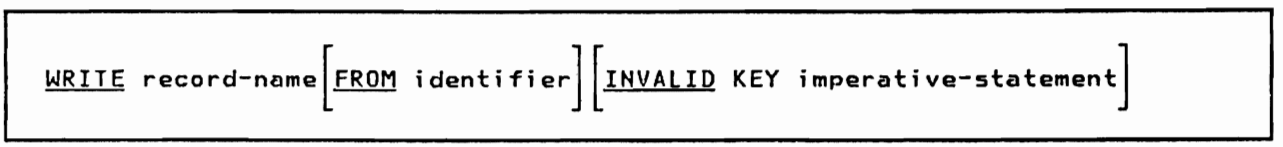
- Relative files opened in OUTPUT or I-O mode
- Indexed sequential files opened in OUTPUT mode
- Indexed random and dynamic files opened in OUTPUT or I-O mode
- Sequential files opened in OUTPUT or EXTEND mode
- TRANSACTION files.

The formats of the WRITE statement are:

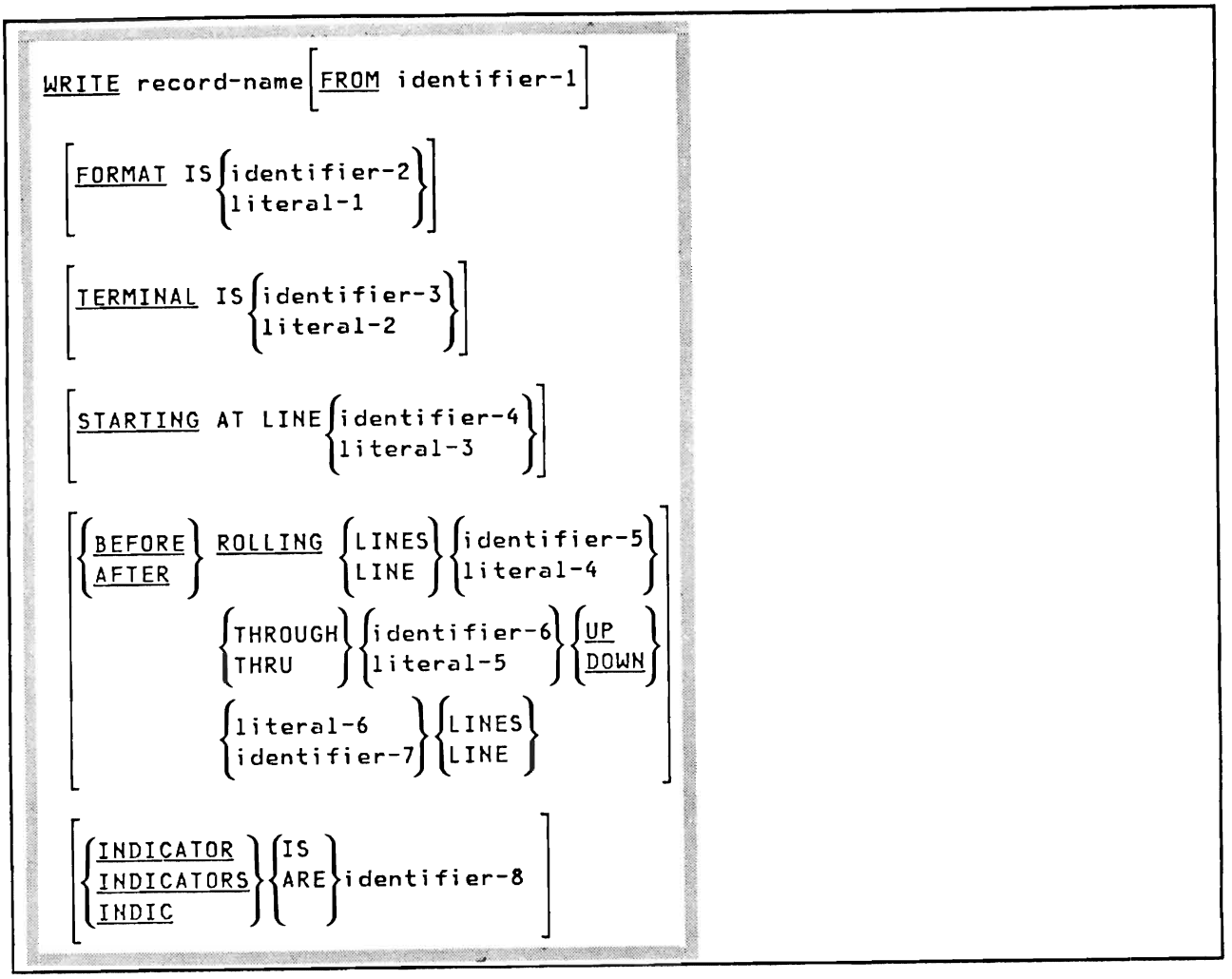
Format 1



Format 2



Format 3-TRANSACTION File



The record name must be the name of a logical record in the File Section of the Data Division. The record name can be qualified, but must not be associated with a sort or a merge file.

The maximum record size for the file is established at the time the file is created and cannot subsequently be changed. User-defined record lengths that are not compatible with the record length specified in the file may result in a nonzero file status at open time and the following results during output to the file:

- A user-defined length greater than file-specified length causes truncation. If the file is empty, the larger record length is used.
- A user-defined length less than file-specified length causes padding with blanks.

The WRITE statement releases a logical record to the file associated with the record name. After the WRITE statement is performed, the logical record is no longer available in the file associated with the record name, unless either of the following is true:

- The associated file is named in a SAME RECORD AREA clause. If so, the record is also available as a record of the other files named in the SAME RECORD AREA clause.

- The WRITE statement is unsuccessful because of a boundary violation an attempt to write beyond the externally defined boundaries of a file.

If either condition is true, the logical record is still available in the file associated with the record name.

The current record pointer is not affected by the WRITE statement.

The number of character positions required to store the record in a file may or may not be the same as the number of character positions defined by the logical description of that record in the COBOL program. See the descriptions of the PICTURE and USAGE clauses under *Data Description Entry* in Chapter 10.

The data items specified in the RECORD KEY clause must be given a value prior to the running of the WRITE statement.

Note: Alternative index considerations:

- *A duplicate key error is produced if an index is set over a sequential file or relative file and duplicate keys are not allowed. This error would result even if a program access to the file is sequential or relative when attempting to WRITE or REWRITE a record which happens to have the same KEY value as an existing record.*

- *Any WRITE to the alternative index file built over a relative file will result in an invalid operation error status.*

If you specify the FILE STATUS clause in the file-control entry, the associated status key is updated when the WRITE statement is performed, whether or not it is successful.

The first character of the status key is known as status key 1; the second character is known as status key 2. Combinations of possible values and their meanings are shown in Appendix D.

IBM Extension

TRANSACTION File Extended File Status Key: The extended file status key for a TRANSACTION file is four characters long. Characters 1 and 2 contain the ICF major return code; characters 3 and 4 contain the ICF minor return code. ICF return codes are described in the *Interactive Communications Feature: Reference*, SC21-7910. See Appendix D in this COBOL manual for a list of status keys and their meanings.

End of IBM Extension

FROM Identifier Phrase: The identifier specified must be the name of an entry in the Working-Storage Section or the Linkage Section or of a record description for another previously opened file. The record name and the identifier must not refer to the same storage area. When the FROM identifier phrase is specified, an implicit move operation is performed according to MOVE statement rules without the CORRESPONDING phrase.

The FROM identifier phrase makes a WRITE statement equivalent to:

```
MOVE identifier TO record-name

WRITE record-name.
```

After a successful WRITE or REWRITE statement, the current record may no longer be available in the record-name, but is still available in identifier.

Format 1 Considerations

This format is valid only for sequential files.

When an attempt is made to write beyond the externally defined boundaries of the file, the WRITE statement is unsuccessful, and an EXCEPTION/ERROR condition occurs. The status key, if specified, is updated, and you must specify an explicit or implicit EXCEPTION/ERROR procedure for the file.

The ADVANCING and END-OF-PAGE phrases control the vertical positioning of each line on a printed page.

ADVANCING Phrase: If you omit the ADVANCING phrase, automatic line advancing is provided. The default statement is AFTER ADVANCING 1 LINE. If you specify the ADVANCING phrase, the following rules apply:

- If you specify BEFORE ADVANCING, the line is printed before the page is advanced.
- If you specify AFTER ADVANCING, the page is advanced before the line is printed.
- If you specify identifier-2, the page is advanced the number of lines equal to the current value in identifier-2. Identifier-2 must name an elementary integer data item. Identifier-2 can be 0.
- If you specify an integer, the page is advanced the number of lines equal to the value of the integer. The integer can be 0.
- If you specify the mnemonic name, a page eject or space suppression takes place. You must equate the mnemonic name with function-name-1 in the SPECIAL-NAMES paragraph. This phrase is not valid if you specify a LINAGE clause in the FD entry for this file.
- When you specify PAGE, the record is printed on the logical page before or after (depending on the phrase used) the device is positioned to the next logical page. If PAGE has no meaning for the device used, then depending

on the phrase you specified, BEFORE or AFTER ADVANCING 1 LINE is provided.

If the FD entry contains a LINAGE clause, the repositioning is to the first printable line of the next page as specified in that clause.

Note: Given the above FD entry condition, if the first WRITE statement run after the opening of the file has and AFTER ADVANCING PAGE, a blank page will precede the remainder of the output.

If you omit the LINAGE clause from the FD entry, the repositioning is to line one of the next page.

Given the above FD entry condition, if the first WRITE statement run after the opening of the file has and AFTER ADVANCING PAGE, the output line is printed on line one of the first page.

If you specify the LINAGE clause for this file, the associated LINAGE-COUNTER special register is modified while the WRITE statement is performed, according to the following rules:

- If you specify ADVANCING PAGE, LINAGE-COUNTER is reset to 1.
- If you specify ADVANCING identifier-2 or integer, LINAGE-COUNTER is incremented by that value.
- If you omit the ADVANCING phrase, LINAGE-COUNTER is incremented by 1.
- When the device is repositioned to the first printable line of a new page, LINAGE-COUNTER is reset to 1.

END-OF-PAGE Phrase: The key words END-OF-PAGE and EOP are equivalent.

When you specify the END-OF-PAGE phrase, the FD entry for this file must contain a LINAGE clause. When you specify END-OF-PAGE, and the logical end of the printed page is reached during the WRITE statement, the END-OF-PAGE imperative statement is performed.

You specify the logical end of the printed page in the associated LINAGE clause.

An END-OF-PAGE condition is reached when a WRITE END-OF-PAGE statement causes printing or spacing within the footing area of a page body. This occurs when such a WRITE statement causes the value in the LINAGE-COUNTER to equal or exceed the value you specified in the WITH FOOTING phrase of the LINAGE clause. The WRITE statement is performed, and then the END-OF-PAGE imperative statement is performed.

An automatic page overflow condition is reached whenever any given WRITE statement with or without the END-OF-PAGE phrase cannot be completely performed within the current page body. This occurs when a WRITE statement would cause the value in the LINAGE-COUNTER to exceed the number of lines for the page body specified in the LINAGE clause. In this case, the line is

printed before or after the device is repositioned to the first printable line on the next logical page, as specified in the LINAGE clause. If you specified the END-OF-PAGE phrase, the END-OF-PAGE imperative statement is then performed.

The END-OF-PAGE condition and the automatic page overflow condition occur simultaneously when:

- You do not specify the WITH FOOTING phrase of the LINAGE clause. This happens because there is no distinction between the END-OF-PAGE condition and the page overflow condition.

- You specify the WITH FOOTING phrase, but a WRITE statement would cause the LINAGE-COUNTER to exceed both the footing value and the page body value specified in the LINAGE clause.

Format 2 Considerations

Format 2 is valid only for indexed and relative files.

INVALID KEY Condition

The INVALID KEY condition can occur when a WRITE statement is run. When the INVALID KEY condition is recognized, the actions are taken in the following order:

1. If you specify the FILE-STATUS clause in the file control entry, a value of 2 is placed in status key 1 to indicate an INVALID KEY condition (see Appendix D).

2. If you specify the INVALID KEY phrase in the statement causing the condition, control is transferred to the INVALID KEY imperative statement. Any EXCEPTION/ERROR declarative procedure specified for this file is not performed.

3. If you do not specify the INVALID KEY phrase, but you do specify an EXCEPTION/ERROR declarative procedure for the file, the EXCEPTION/ERROR procedure is performed.

When an INVALID KEY condition occurs, the input or output statement that caused the condition is unsuccessful. If you do not specify the INVALID KEY phrase for a file, you must specify an EXCEPTION/ERROR procedure. If an error other than an INVALID KEY condition occurs, the EXCEPTION/ERROR procedure is run.

Indexed Files: Before the WRITE statement is performed, you must set the record key (the RECORD KEY data item, as defined in the file control entry) to the desired value. RECORD KEY values must be unique within a file unless WITH DUPLICATES is specified in the RECORD KEY clause. When the WRITE statement is performed, the system releases the record.

When you specify ACCESS IS SEQUENTIAL in the file control entry, the file must be opened for OUTPUT for a WRITE statement. Records must be written

in ascending order of RECORD KEY values. If not written in ascending order, an INVALID KEY condition exists.

When you specify ACCESS IS RANDOM or ACCESS IS DYNAMIC in the file control entry, records can be written in any user-specified order.

You must specify the INVALID KEY phrase if an explicit or implicit EXCEPTION/ERROR procedure is not specified for this file. An INVALID KEY condition is caused by any of the following:

- You specified ACCESS IS SEQUENTIAL, the file is opened OUTPUT, and the value of the record key is not greater than that for the previous record (or equal to, if WITH DUPLICATES is specified in the RECORD KEY clause).
- The value of the record key equals that of an already existing record and duplicate keys are not allowed.
- An attempt is made to write beyond the externally defined boundaries of the file.
- The record being written creates a duplicate key in another index that does not allow duplicate keys.

Notes:

1. *If the file is opened for OUTPUT, does not allow duplicate keys, and has RANDOM or DYNAMIC access (unordered load), duplicate keys are not checked when the WRITE statement is run. This check is done at end-of-job or when the file is reopened. If you want duplicate key checking at the time the WRITE statement is run, open the file for I-O.*

2. *The BYPASS-YES parameter on the FILE OCL statement lets you suppress duplicate key checking when adding a record to an indexed file. It is your responsibility to ensure that duplicate keys are not added. The BYPASS-YES parameter is not intended as support for duplicate keys. Specifying the BYPASS-YES parameter can improve system performance, but results in nonstandard COBOL file processing. For more information on the BYPASS-YES parameter, see the FILE Statement in the manual System Reference.*

Relative Files: The WRITE statement is valid for both OUTPUT and I-O files.

For OUTPUT files, the WRITE statement causes the following actions:

- If you specify ACCESS IS SEQUENTIAL, the first record released has relative record number 1; the second, number 2; the third, number 3; and so on. If you specify RELATIVE KEY in the file control entry, the relative record number of the record just released is placed in the RELATIVE KEY while the WRITE statement is performed.
- If you specify ACCESS IS RANDOM or ACCESS IS DYNAMIC, the RELATIVE KEY must contain the desired relative record number for this record before the WRITE statement is issued. When the WRITE statement is

performed, this record is placed at the specified relative record number position in the file if this relative record position is vacant.

For I-O files, when you specify ACCESS IS RANDOM or ACCESS IS DYNAMIC, new records are inserted into the files. The RELATIVE KEY must contain the desired relative record number for this record before the WRITE statement is issued. When the WRITE statement is performed, this record is placed at the specified relative record number position in the file.

You must specify a delete-capable file for this format of WRITE to work correctly.

Using this format of WRITE with non-delete capable files may produce unexpected results that are in contradiction to COBOL standards.

You must specify the INVALID KEY phrase if you do not specify an explicit or implicit EXCEPTION/ERROR procedure for this file.

An INVALID KEY condition is caused by either of the following:

- You specified ACCESS IS RANDOM or ACCESS IS DYNAMIC, and the RELATIVE KEY specifies a record that already contains data.
- An attempt is made to write beyond the externally defined boundaries of the file.

Format 3 Considerations

IBM Extension

Format 3 is valid only for the TRANSACTION file.

The WRITE Statement releases a logical record to the TRANSACTION file. This file must be opened in the I-O mode at the time the WRITE statement is performed.

Literal-1 and literal-2 must be nonnumeric. Literal-3, literal-4, literal-5, and literal-6 must be numeric.

Identifier-2 must be an alphabetic or alphanumeric data item, and identifier-3 must an alphanumeric data item. Identifier-4, identifier-5, identifier-6, and identifier-7 must be elementary numeric items. Identifier-8 must be either an elementary Boolean data item specified without the OCCURS clause or a group item that has Boolean data elementary items subordinate to it.

FORMAT Phrase: The WRITE statement specifies the format used for output. This format must be one of the following:

- a system-defined special format
- a screen format contained in the ¥SFGR load member specified in the associated ASSIGN clause

- an IDDU communications format contained in the IDDU format file specified in the associated ASSIGN clause.

The record you specified with the record name is sent to the specified or implied destination using the named format. You must specify a format for the first WRITE verb to be performed. If subsequent WRITE operations do not include a FORMAT phrase, the format used most recently for this TRANSACTION file is used. The FORMAT phrase contains the name of the screen or IDDU format used when data is written to the display station.

Writing to the Error Line: If the format name used for the write operation is the literal 'ERRLINE', System/36 COBOL generates a write to the error line of the display station instead of a write with format. A line written to the error line cannot exceed 78 characters. A write operation to the error line causes the last line of output on the display station to be saved and the output record to replace the bottom line on the display. When you press the RESET key, the original line reappears.

Note: A WRITE statement that writes to the error line cannot specify BEFORE or AFTER ROLLING.

Interactive Communications Feature: Special format names are recognized by data management that provide you with SSP-ICF functions. The uses of these special format names and the functions of ICF are described in the *Interactive Communications Feature: Reference*. Because the system-defined special format names begin with two dollar signs ($$), you should not begin your display format names with $$.

Data formats may be defined in the files through the Interactive Data Definition Utility (IDDU). These formats are used by the Advanced Program-to-Program Communications (APPC) subsystem of the SSP-ICF.

TERMINAL Phrase: The TERMINAL phrase is used to specify where the record is to be sent. If you do not specify the TERMINAL phrase for a single device file, that device is the destination. If you do not specify the TERMINAL phrase for a multiple device file, the most-recent source or destination identifier is used as the destination.

STARTING Phrase: The STARTING phrase contains the starting line number for display formats that use the variable start line option. If the value of this element is less than 01, a value of 01 is assumed. The maximum value is 1 less than the size of the display. If the display format does not specify this phrase, display station data management (DSDM) ignores this value.

ROLLING Phrase: The ROLLING phrase lets you move the data currently displayed at the display station. All or part of the data on the display can be rolled up or down. The lines that are rolled are cleared and can have another display format written into them.

You specify rolling on the WRITE statement that is writing a new format to the display station. You must specify the number of lines you want to roll, how many lines you want to roll these lines, and whether the roll operation is up or down.

Rolling ignores field attributes. The data is rolled exactly as it appears on the display. Its associated attributes (for example, whether it is an input field or an input/output field) are not rolled with the data and are lost; therefore, after a field has been rolled, it is no longer input-capable.

For an example of using the ROLLING phrase, see *WRITE Statement* in Chapter 7.

INDICATOR Phrase: You can use the INDICATOR phrase to specify the name of an area that contains indicator information.

Indicators that are provided in the indicator area (identifier-8), but not specified on the $SFGR or IDDU formats are ignored.

End of IBM Extension

Using the Additional COBOL Functions

Chapter 13. Using the Additional COBOL Functions

This chapter describes additional System/36 COBOL functions that can help make your programs more efficient. Included are detailed discussions of:

- The Segmentation Feature, which provides for better storage use and for larger programs
- The Sort/Merge function, which provides a convenient way of arranging records
- The Table Handling function, which provides a method for data reference
- Linking modules that are written in other System/36 languages
- Considerations when using the Inter-Program Communication Feature.

USING THE SEGMENTATION FEATURE

The Segmentation Feature helps you optimize storage in the Procedure Division by letting you subdivide that division for overlays both physically and logically.

Segmentation Concepts

Although segmentation is not required, you usually write the Procedure Division of a source program as a consecutive group of sections. Each section is composed of a series of closely related operations that perform a particular function.

When you use the Segmentation Feature, you must divide the entire Procedure Division into sections. You must classify each section in the division according to physical and logical attributes by a system of segment numbers. Segment numbers must be in the range 0 through 99.

When Segments are Beneficial: Segmentation should be used when overlays are required, to indicate which sections of your program should form overlays, and which sections should remain in main storage. The execution time of your program will increase, but this feature may decrease the region size required for program execution. Under certain circumstances, segmented programs can be faster than nonsegmented programs. Assume that you have a program designed to read a large disk file, record running statistics, then print this information in the form of a graphic chart. Assume that the graphic output is complex enough to require a large amount of code. In this situation, you should create a segmented structure: one program to do the printing, and a second program to read the disk and calculate.

Program Segments

Two types of program segments are available:

- Permanent
- Independent.

Permanent Segments (0 through 49)

A permanent segment composes the fixed portion (sometimes called root segment) of the load module. A permanent segment cannot be overlaid by any other part of the program. It is always present in its last-used state.

The fixed portion is the part of the load module that logically resides in main storage while the load module is running.

Independent Segments (50 through 99)

An independent segment is the part of the load module that can overlay or be overlaid by another independent segment. An independent segment is always considered to be in its initial state each time it is made available to the program.

Segmentation Logic

In a segmented program, you classify the sections by a system of segment numbers. All sections with the same segment number constitute a program segment with that priority. You must make the segment number an integer ranging in value from 0 through 99. Segments with segment numbers ranging from 0 through 49 are permanent segments. Segments with segment numbers ranging from 50 through 99 are independent segments. If you leave a segment number out of the section header, the segment number is assumed to be 0. You must use segment numbers under 50 (permanent segments) for sections in the declaratives portion of the Procedure Division.

Use the following criteria when you assign segment numbers and segment types:

- Frequency of use--Place sections that are used often or sections that must be available for references at all times within permanent segments. Place less-frequently used sections within independent segments.
- Frequency of reference--The more frequently you refer to a section, the lower the segment number you should use. The less frequently you refer to a section, the higher the segment number you should use.
- Logical relationships--Give identical segment numbers to sections that frequently communicate with each other (you need not make sections with the same segment number adjacent in the source program).

Segmentation Control

The logical sequence of the program is the same as the physical sequence of the program except for specific transfers of control. The load module must be resequenced if a segment has its section scattered throughout the source program. When the load module is reordered, the compiler provides control transfers to maintain the logic flow of the source program. When necessary, the compiler inserts instructions necessary to load or to initialize a segment. Within a source program, you can transfer control to any paragraph in a section. You need not transfer control to the beginning of a section.

Running of the segmented load module begins in the fixed portion. All tables, literals, and system control blocks are included in the fixed portion. When you have CALL statements in a segmented program, nonsegmented subprograms are loaded with the fixed portion of the main program.

If a segmented program calls a subprogram, you can have the CALL statement in any segment.

When a segmented COBOL subprogram contains independent segments, you must use one of the following link-edit techniques:

1. First technique:

    a. Compile both called and calling programs with the OBJECT phrase in the PROCESS statement to create a module that can be relocated (subroutine member).
    b. Invoke the overlay linkage editor with OCL or the OLINK command.
    c. Use the OCL MODULE statement to give the mainline module name and the names of any subprogram modules that contain independent segments.

2. Second technique:

    a. After setting up the appropriate segment arrangement in both the subprogram(s) and the main program, compile the subprogram(s) with the PROCESS statement OBJECT phrase.
    b. Compile and link the main program with the subprogram(s).

Note: Segment names must be unique within the main program and all of its subprograms. The compiler assigns a name to each segment by using the first three characters of the program name, an asterisk, and the segment number, in that order.

Refer to the *OLE Guide* for additional information.

Size of the Object Program

When using the Segmentation Feature, you can direct the overlay linkage editor to construct an object program of a particular size. You do this by giving the size of main storage available for running in the MEMORY SIZE clause of the OBJECT-COMPUTER paragraph. The overlay linkage editor then tries to produce a program that will use the available space. You are informed if the program does not fit in the space given. If you leave out the MEMORY SIZE clause, the size of the compiler region is assumed.

PROCEDURE DIVISION SEGMENTATION

In the Procedure Division of a segmented program, classify segments by segment numbers. Include the segment number in the section header.

Format

```
section-name SECTION [segment-number] .
```

Special Segmentation Considerations

When you use segmentation, there are restrictions on the ALTER, PERFORM, and SORT and MERGE statements. You also need to give special consideration to transfer of control, calling programs, and called programs.

ALTER Statement

You can change a GO TO statement within an independent segment only with an ALTER statement that is within the same segment. You can change a GO TO statement in a permanent segment with an ALTER statement in any segment of the program.

PERFORM Statement

You can use a PERFORM statement in a permanent segment to refer only to sections wholly contained within the fixed portion of the program or to sections wholly contained within one independent segment.

You can use a PERFORM statement in an independent segment to refer only to sections wholly contained within the fixed portion of the program or sections wholly contained within the same independent segment as the PERFORM statement.

Note: When you reference procedures within an independent segment with a PERFORM statement, you must not pass control outside that independent segment. Also, when you reference the fixed portion with a PERFORM statement in an independent segment, you must not pass control to another independent segment from the fixed portion. Return linkages might be destroyed, and an abnormal termination is likely to occur.

SORT and MERGE Statements

If you place a SORT or MERGE statement in the fixed portion, then you must place any SORT input procedures or SORT/MERGE output procedures completely in either the fixed portion or one independent segment.

If you place a SORT or MERGE statement in an independent segment, you must place any SORT input procedures or SORT/MERGE output procedures

completely in either the fixed portion or the same independent segment as the SORT or MERGE statement.

Transfer of Control

The Segmentation Feature imposes no restrictions on transfers of control as long as you keep the control path within the range of permanent segments. Permanent segments are logically identical and you can treat them as though the program was not segmented.

If independent segments are involved in the transfer of control, restrictions do apply. Independent segments are loaded into main storage under control of the system management routines. You must assume that logically, only one independent segment at a time is present in main storage, even though this may not be the case physically. You must not perform statements that depend on the presence of any given independent segment, other than the one in which the statements appear, in main storage at any given time.

Calling and Called Programs

You can place the CALL statement anywhere within a segmented program. When you place a CALL statement in an independent segment, that segment is in its last-used state when control returns to the calling program if the subprogram or sections of the subprogram are not segmented.

USING THE SORT/MERGE FACILITIES

You will probably often need to arrange records in a particular order or sequence. Sorting and merging facilities help you do this. While you can use either facility to order records, the functions and capabilities of a sort and a merge are different.

A sort operation will give you an ordered file from one to eight input files that might be completely unordered as to sort sequence. Thus, the sort facility accepts unordered sort input and produces ordered sort output.

A merge operation will give you an ordered file from two to eight input files, each of which is already ordered in the merge sequence.

IBM Extension

Input files need not be sequenced before doing a merge operation.

End of IBM Extension

COBOL has special language features that assist in sort and merge operations so that you need not program these operations in detail.

For an explanation of messages that are issued by the System/36 SORT program after it has been invoked by a COBOL object program, see the *Sort Guide* or the manual *Systems Messages*.

SORT/MERGE Concepts

Sorting and merging have always been a large percentage of the workload in business data processing. COBOL standardizes the specification of these facilities making them easy for you to use and to change. Alternatively, you can use the System/36 SORT program to perform these facilities as a separate job step. The COBOL language supports these operations through the:

- File control entry in the Environment Division
- SD (sort-merge file description) entry in the Data Division
- SORT and MERGE statements in the Procedure Division.

Describe the sort or merge file through the:

- File control entry in the Environment Division
- SD entry in the Data Division.

The sort or merge file is the working file used during the sort or merge; you can consider it an internal file. However, a sort or merge file, like any file, is a set of records, and you can consider a sort-merge file description a particular type of file description.

Process the sort-merge file through a Procedure Division SORT or MERGE statement. The statement gives the key field(s) within the record upon which the sort or merge is to be arranged. You can use ascending or descending keys. When you use more than one key, a mixture of the two sequences is allowed. The sequence of sorted or merged records conforms to the mixture of keys you used.

Sort Concepts

Through the SORT statement, you have access to input procedures (used before sorting) and output procedures (used after sorting) that can add, delete, alter, edit, or otherwise change the records in the input or output files. Your COBOL program can contain any number of sorts, each of them with its own independent input or output procedures. During performance of the SORT statement, these procedures are automatically performed at the given point in processing; thus, extra passes through the sort file are avoided.

You will usually organize a COBOL program containing a sort so that one or more input files are read and operated on by an input procedure. Within the input procedure, use a RELEASE statement (corresponding to the WRITE statement) to place a record in the sort file. That is, when the input procedure is completed, you have created a sort file by placing records one at a time into the sort file through the RELEASE statement. If you do not want to change the records before the sorting operation begins, use the SORT statement USING option to release the unchanged records to the sort file.

After all the input records have been placed in the sort file, the sort operation is performed. This operation arranges the entire set of sort file records in the sequence specified by the key(s).

After the sort operation is completed, you can make sorted records available from the sort file, one at a time, through a RETURN statement for change in an output procedure. If you do not want to change the sorted records, use the SORT statement GIVING option to name the sorted output file.

Merge Concepts

Through the MERGE statement, you can access output procedures (used after merging) that can change the records in the output file. Your COBOL program can contain any number of merge operations, each with its own independent output procedures. While the MERGE statement is performed, these procedures are automatically performed at the given point in processing.

MERGE statement performance begins the merge processing. This operation compares keys within the records of the input files and arranges the records within the merged file in the sequence specified by the key(s).

You can make merged records available, one at a time, through a RETURN statement for change in an output procedure. If you do not wish to change the merged records, use the MERGE statement GIVING option to name the merged output file.

SORT/MERGE Programming Considerations

This section describes things you should consider when performing sort or merge operations.

Disk Storage Requirements

Whenever you use an input procedure with a SORT statement or an output procedure with a SORT or MERGE statement, you must have disk space available for COBOL to use for the sort or merge file. You must use a FILE OCL statement for each such sort or merge file. When you use both the USING and GIVING options for a SORT or MERGE statement, the system automatically allocates an intermediate work area to hold the sort or merge records. Thus, you need not use a FILE OCL statement for such a sort or merge file.

The System/36 SORT program called by the COBOL program requires a work area in which to perform the sort. You can create this work area with a FILE OCL statement with a NAME-WORK parameter. If you do not use this statement, the sort program allocates a scratch file large enough to perform the desired sort or merge operation. See the *Sort Guide* for details on the size of this work file.

Note: If you use a RESERVE OCL statement, you can reserve disk space for the scratch files that the sort program uses. If you do not reserve enough space, an error message is issued. This message lets you either allocate more space or cancel the job.

Performance Considerations

You can usually improve performance by using the USING or GIVING phrase on the SORT or MERGE statement. You can usually get the best performance by using both phrases. These phrases let you bypass writing the input records into the sort or merge work area (by RELEASE statements) and reading the sorted records from the sort or merge work area (by RETURN statements).

When using input and output procedures, you must not use a BLOCK CONTAINS clause on the SD statement for the sort or merge file. System/36 COBOL always defaults to one record, the minimum blocking factor. To control the blocking factor for SORT files, refer to the discussion of the DBLOCK parameter of the FILE OCL statement in the manual *System Reference*.

Environment Division SORT/MERGE

In the Environment Division, you must:

- Write file control entries for each file you use as input to or output from a sort or merge operation

and

- Write a file control entry for each unique sort file or merge file.

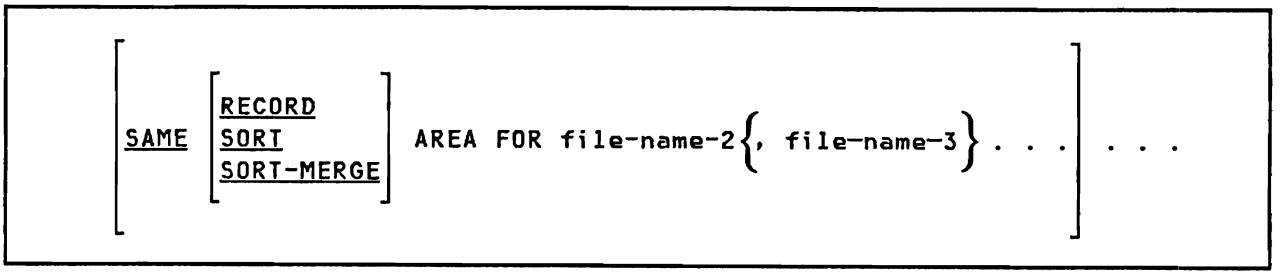
FILE-CONTROL Paragraph

For a description of input and output files of a sort or merge operation, see *FILE-CONTROL Paragraph* in Chapter 9.

I-O-CONTROL Paragraph

In the I-O-CONTROL Paragraph, use the SAME SORT AREA or SAME SORT-MERGE AREA clause.

Format



The SAME SORT AREA or SAME SORT-MERGE AREA clause reduces the storage area assigned to a given SORT or MERGE statement.

In the SAME AREA clause, SORT and SORT-MERGE are equivalent.

The SAME SORT AREA or SAME SORT-MERGE AREA clause gives one storage area for sort/merge operations by each named sort or merge file; that is, the storage allocated for one such operation is available for reuse in another.

When you use the SAME SORT AREA or SAME SORT-MERGE AREA clause, you must use at least one file name to name a sort or merge file. You can also use files that are not sort or merge files. The following rules apply:

- You can use more than one SAME SORT AREA or SAME SORT-MERGE AREA clause; however, you must not name one sort or merge file in more than one such clause.

- If you name a file that is not a sort or merge file in both a SAME AREA clause and in one or more SAME SORT AREA or SAME SORT-MERGE AREA clauses, you must also include all the files in the SAME AREA clause in that SAME SORT AREA or SAME SORT-MERGE AREA clause.

- Files that you name in a SAME SORT AREA or SAME SORT-MERGE AREA clause need not have the same organization or access.

- Files you name in a SAME SORT AREA or SORT-MERGE AREA clause that are not sort or merge files do not share storage with each other unless you name them in a SAME AREA or SAME RECORD AREA clause.

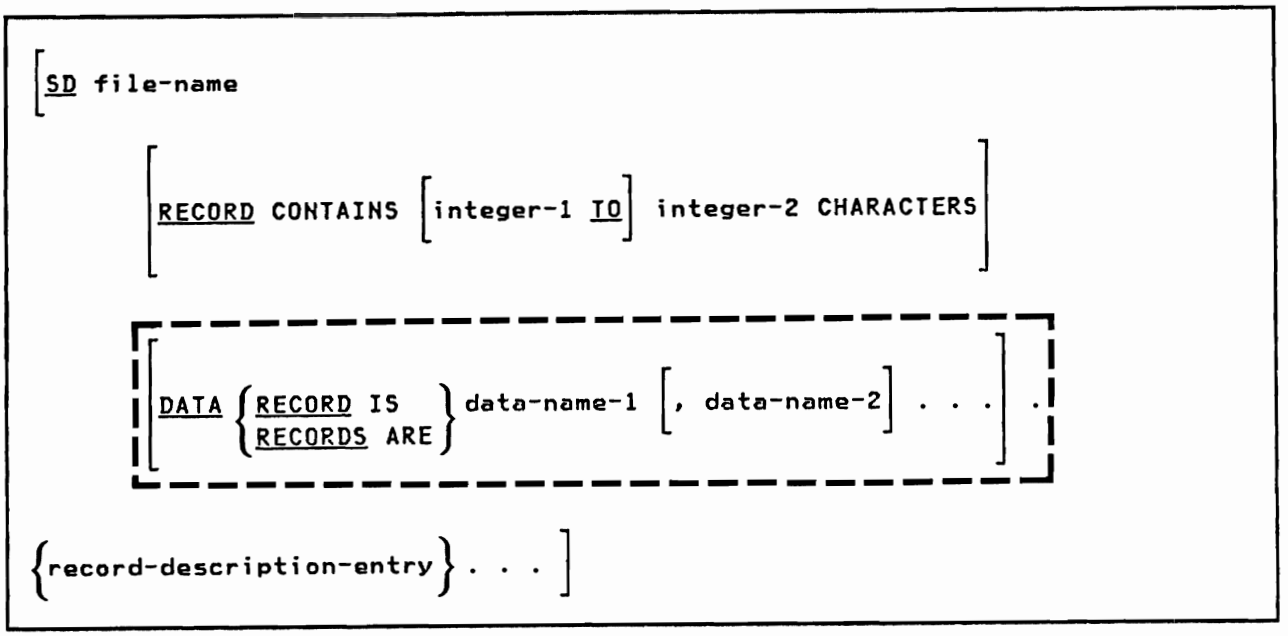
- Files you name in a SAME SORT AREA or SAME SORT-MERGE AREA clause that are not sort or merge files must not be open during the performance of a SORT or MERGE statement that refers to a sort or merge file named in the clause.

Rules for the specification of the SAME RECORD AREA clause are given under *I-O-CONTROL Paragraph* in Chapter 9.

Data Division SORT/MERGE

In the File Section, you must write an FD entry for each file that is input to or output from the sort/merge operation, as well as a record description entry. In addition, you must have an SD (sort-merge file description) entry for each sort or merge file.

Format



Place the level indicator SD, which identifies the beginning of the SD entry, before the file name. You must use a sort or merge file as the file name.

The clauses that follow the file name are optional, and their order of appearance is not important. Both the RECORD CONTAINS clause and the DATA RECORDS clause are described in Chapter 10.

You must place one or more record description entries after the SD entry; however, no input/output statements can be performed for this file.

The following example illustrates the File Section entries you need for a sort or merge file:

```
SD SORT-FILE.

01 SORT-RECORD  PICTURE X(80).
```

Procedure Division SORT/MERGE

In the Procedure Division, you include MERGE and SORT statements to describe the merge and sort operations and, optionally, sort input procedures or sort/merge output procedures. A sort input procedure must contain a RELEASE statement that makes each record available to the sort operation. A sort/merge output procedure must contain a RETURN statement that makes a sorted/merged record available to the output procedure.

You can include more than one SORT or MERGE statement in the Procedure Division. You can place these statements anywhere except in the Declaratives section or in the sort input or sort/merge output procedures.

You must describe files given in the USING and GIVING phrases of the SORT and MERGE statements explicitly or implicitly in their file control entries as having sequential organization.

USE procedures are not performed if they reference files given on a USING or GIVING phrase of a SORT or MERGE statement. The USING or GIVING files are not accessed by COBOL, but by the sort program. If you also reference these files in an I/O statement within an output procedure or a SORT input procedure, a USE procedure for the given file is invoked when necessary.

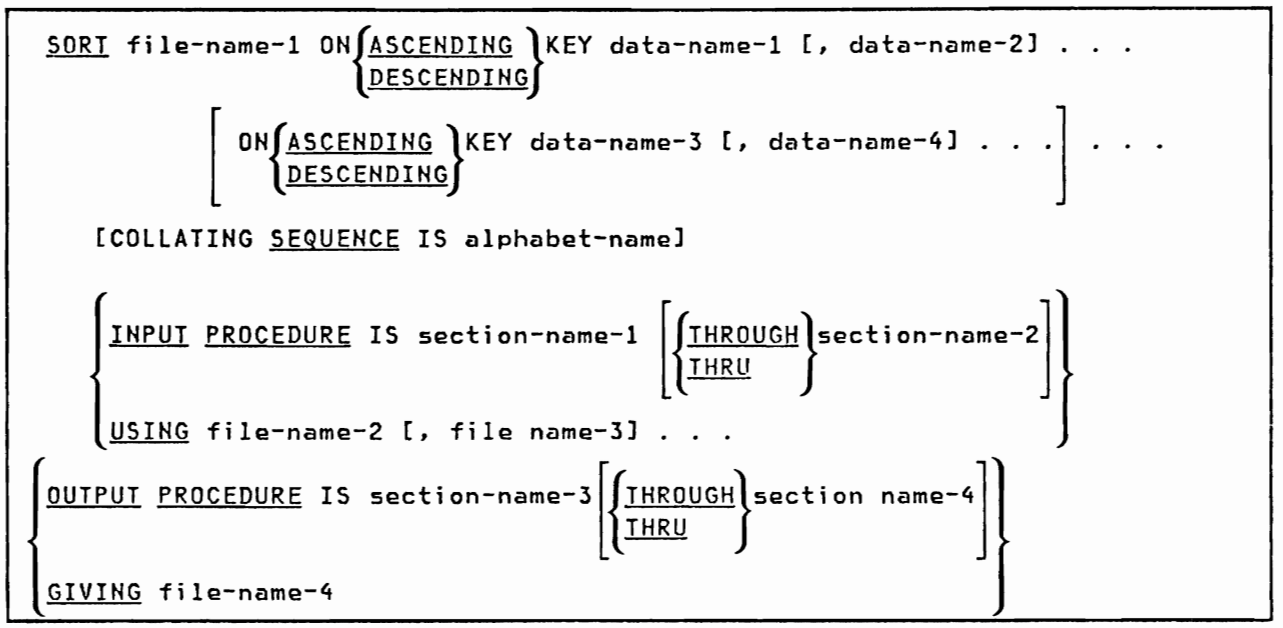
SORT Statement

The SORT statement:

- Accepts records from one or more files
- Sorts the records according to the specified key(s)
- Makes records available either through an output procedure or in an output file.

The maximum number of files accepted by the SORT statement is eight.

Format



File-name-1 is the name you give in the SD entry that describes the records being sorted.

When the SORT statement is performed, all records contained in file-name-2, file-name-3, and so on are accepted by the sort/merge program and then sorted according to the key(s) specified. These input files must not be open at the time the SORT statement is performed; they are automatically opened and closed by the SORT operation, and all implicit functions are performed. The files are closed as if you wrote the CLOSE statement without any optional processing.

MERGE Statement

The MERGE statement combines from two through eight identically sequenced files that have already been sorted according to an identical set of ascending or descending keys on one or more keys. This statement makes records available in merged order to an output procedure or output file.

Format

```
MERGE file-name-1 ON {ASCENDING } KEY data-name-1 [, data-name-2] . . .
                     {DESCENDING}

          [ ON {ASCENDING } KEY data-name-3 [, data-name-4] . . . ] . . .
               {DESCENDING}

   [COLLATING SEQUENCE IS alphabet-name]

       USING file-name-2, file-name-3 [, file-name-4] . . .

{ OUTPUT PROCEDURE IS section-name-1 [ {THROUGH} section name-2 ] }
                                     [ {THRU   }                ]
{ GIVING file-name-5                                            }
```

File-name-1 is the name you give in the SD entry that describes the records being merged. You must not reuse any file name in the MERGE statement.

When the MERGE statement is performed, all records in file-name-2, file-name-3, and so on are accepted by the sort/merge program and then merged according to the key(s) given. These files must not be open when the MERGE statement is being performed. They are automatically opened and closed by the MERGE operation, and all implicit functions are performed. The files are closed as if you used a CLOSE statement with no optional processing.

SORT Statement and MERGE Statement Phrases

Most SORT and MERGE statement phrases apply to both the statements. The common SORT/MERGE statement phrases are:

- The ASCENDING/DESCENDING KEY phrase
- The COLLATING SEQUENCE phrase
- The USING phrase
- The GIVING phrase
- The OUTPUT PROCEDURE phrase.

The INPUT PROCEDURE phrase applies only to the SORT statement.

ASCENDING/DESCENDING KEY Phrase

This phrase specifies that records are to be processed in an ascending or descending sequence based on the given sort or merge keys.

Each data name gives a KEY data item on which the sort or merge will be based. In each such data name, you must identify a data item in a record associated with file-name-1. The following rules apply:

- A specific KEY data item must be physically located in the same position and have the same data format in each input file; however, it need not have the same data name.
- If file-name-1 has more than one record description, then you need to describe the KEY data items in only one of the record descriptions.
- KEY data items must be fixed-length items.
- KEY data items must not contain an OCCURS clause or be subordinate to an item that contains an OCCURS clause.
- You can use no more than 12 KEY data items.
- The total length of all KEY data items must not be greater than 256 bytes. These 256 bytes can include bytes used by the compiler. Generally, you will use 1 additional byte per key for each numeric key given. For a merge operation, 3 additional bytes are used to maintain relative record position. The maximum number of bytes used by the compiler is 15; therefore, you have at least 241 bytes, and possibly all 256 bytes, depending on the type of operation and the data types of the keys given.
- KEY data items can be qualified; they cannot be subscripted or indexed.

The KEY data items are listed in order of decreasing significance, regardless of how they are divided into KEY phrases. Using the format as an example, data-name-1 is the most-significant key, and records are processed in ascending or descending order on that key; data-name-2 is the next most significant key, and within data-name-1, records are processed on data-name-2 in ascending or descending order. Within data-name-2, records are processed on data-name-3 in ascending or descending order; within data-name-3, records are processed on data-name-4 in ascending or descending order.

The direction of the sort or merge operation depends on the use of the ASCENDING or DESCENDING key words as follows:

- When you use ASCENDING, the sequence is from the lowest key value to the highest key value.
- When you use DESCENDING, the sequence is from the highest key value to the lowest.
- If the KEY data item is alphabetic, alphanumeric, alphanumeric edited, or numeric edited, the sequence of key values depends on the collating sequence you used.

- The key comparisons are performed according to the rules for comparison of operands in a relation condition. See *Relation Condition* in Chapter 10.

COLLATING SEQUENCE Phrase

This phrase gives the collating sequence to be used in nonnumeric comparisons for the KEY data items in this sort or merge operation.

You must use the alphabet name in the SPECIAL-NAMES paragraph alphabet-name clause. You can use any one of the alphabet-name phrase options with the following results:

- When you use NATIVE, the EBCDIC collating sequence is used for all nonnumeric comparisons.
- When you use STANDARD-1, all nonnumeric comparisons are made as if the data items were translated from EBCDIC into ASCII. For more information on the translation of EBCDIC items into ASCII, see Appendix F.
- When you use the literal phrase, the collating sequence established by the use of literals in the alphabet-name clause is used for all nonnumeric comparisons.

When you leave out the COLLATING SEQUENCE phrase, the PROGRAM COLLATING SEQUENCE clause (if used) in the OBJECT-COMPUTER paragraph gives the collating sequence to be used. When you leave out both the COLLATING SEQUENCE phrase and the PROGRAM COLLATING SEQUENCE clause, the EBCDIC collating sequence is used.

USING Phrase

When you use the USING phrase, all input files automatically transfer to file-name-1. At the time the SORT or MERGE statement is performed, these input files must not be open; the COBOL compiler opens, reads, and closes these files automatically. If you use EXCEPTION/ERROR procedures for these files, the COBOL compiler makes the necessary linkage to these procedures.

The input files must have sequential organization.

You must describe all input files in an FD entry in the Data Division, and their record descriptions must describe records of the same size as the record you described for the sort or merge file. If the elementary items that make up these records are not identical, you describe the input records as having an equal number of character positions as the sort record.

GIVING Phrase

When you use the GIVING phrase, all the sorted or merged records in file-name-1 transfer automatically to the output file (MERGE file-name-5 or SORT file-name-4). At the time the SORT or MERGE statement is performed, this output file must not be open; the COBOL compiler opens, writes, and closes the file automatically. If you use EXCEPTION/ERROR procedures for the output file, the COBOL compiler makes the necessary linkage to these procedures.

The output file must have sequential organization.

You must describe the output file in an FD entry in the Data Division, and its record description(s) must describe records of the same size as the record you described for the sort or merge file. If the elementary items that make up these records are not identical, you must describe the output record as having an equal number of character positions as the sort or merge record.

SORT INPUT PROCEDURE Phrase

This phrase specifies the section name(s) of a procedure that is to modify input records before the sorting operation begins.

Use section-name-1 to give the first (or only) section in the input procedure. Optionally, use section-name-2 to identify the last section of the input procedure.

You must make the input procedure one or more sections that you write consecutively and that do not form a part of any output procedure. You must include at least one RELEASE statement in the input procedure in order to transfer records to the sort file.

You must not let control pass to the input procedure except when a related SORT statement is being performed, because the RELEASE statement in the input procedure has no meaning unless it is controlled by a SORT statement. You can include any procedures in the input procedure that are needed to select, create, or modify records. The following restrictions apply to the procedural statements within an input procedure:

- You must not include any SORT or MERGE statements in the input procedure.
- You must not include ALTER or GO TO statements in the input procedure if they refer to procedure names outside the input procedure. The performance of a CALL statement to another program that follows standard linkage conventions, as well as the performance of USE declaratives, are not considered transfers of control outside an input procedure; hence, they can be activated within these procedures.
- You must not include any transfers of control to points inside the input procedure in the remainder of the Procedure Division, except for the return of control from a Declaratives Section.

If you use an input procedure, control passes to the input procedure when the SORT program input phase is ready to receive the first record. The compiler places a return mechanism at the end of the last section of the input procedure and when control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted. The RELEASE statement transfers records from the Input Procedure to the sort file, which is then used in the input phase of the sort operation.

SORT/MERGE OUTPUT PROCEDURE Phrase

This phrase specifies the section name(s) of a procedure that is to modify output records from the sort or merge operation.

Use section-name-3 to give the first (or only) section in the output procedure. Optionally, use section-name-4 to identify the last section of the output procedure.

You must make the output procedure one or more sections that you write consecutively and that do not form a part of any input procedure. You must include at least one RETURN statement in the output procedure in order to make sorted/merged records available for processing.

When all the records are sorted or merged, control passes to the output procedure. The RETURN statement in the output procedure is a request for the next record.

You must not let control pass to the output procedure except when a related SORT or MERGE statement is being performed, because RETURN statements in the output procedure have no meaning unless they are controlled by a SORT or MERGE statement. In the output procedure, you can include any procedures needed to select, modify, or copy the records that are being returned one at a time from the sort or merge file. There are three restrictions on the procedural statements within the output procedure:

- You must not include any SORT or MERGE statements in the output procedure.
- You must not include ALTER, GO TO, or PERFORM statements in the output procedure if they refer to procedure names outside the output procedure. The performance of a CALL statement to another program that follows standard linkage conventions, as well as the performance of USE declaratives, are not considered transfers of control outside an output procedure; hence, they can be activated within these procedures.
- You must not include any transfers of control to points inside the output procedure in the remainder of the Procedure Division, except for the return of control from a Declaratives Section.

When you use an output procedure, control passes to it after the sort or merge file (file-name-1) has been placed in sequence by the sort or merge operation. The COBOL compiler places a return mechanism at the end of the last section in the output procedure. When control passes to the last statement in the output procedure, the return mechanism ends the sort or merge, and passes control to the next performable statement after the SORT or MERGE statement.

SORT or MERGE INPUT/OUTPUT PROCEDURE Control

The INPUT or OUTPUT PROCEDURE phrases function in a manner similar to format 1 of the PERFORM statement (the simple PERFORM). For example, naming a section in an OUTPUT PROCEDURE phrase causes performance of that section during the sort or merge operation to proceed as if you named that section in a PERFORM statement. As with the PERFORM statement, performance of the section ends after performance of its last statement. You can

use the EXIT statement as the last statement in Input and Output Procedures. This is useful for documentation purposes.

RELEASE Statement (Sort Function Only)

The RELEASE statement transfers records from an input/output area to the initial phase of a sort operation.

You can use the RELEASE statement only within an input procedure associated with a SORT statement. You must use at least one RELEASE statement within an input procedure.

When the RELEASE statement is performed, the current contents of the record name are placed in the sort file (that is, made available to the initial phase of the sort operation).

Format

```
RELEASE record-name [FROM identifier]
```

You must use a record associated with the SD entry for file-name-1 for the record name. You can qualify the record name.

When you use the FROM identifier phrase, the RELEASE statement is equivalent to the statement:

MOVE identifier to record-name

followed by the statement:

RELEASE record-name.

Moving takes place according to the rules for the MOVE statement without the CORRESPONDING option.

You must not refer to the same storage area with the identifier and the record name.

After the RELEASE statement is performed, the information in the record name is no longer available unless you use file-name-1 in a SAME RECORD AREA clause, in which case the record name is still available as a record of the other files named in that clause. When you use the FROM identifier phrase, the information is still available in the identifier.

When control passes from the input procedure, the sort file consists of all those records placed in it by performance of RELEASE statements.

RETURN Statement

The RETURN statement transfers records from the final phase of a sort or merge operation to an input/output area.

You can use the RETURN statement only within an output procedure associated with a SORT or MERGE statement. Within this output procedure, you must use at least one RETURN statement.

Format

```
RETURN file-name RECORD [INTO identifier] AT END imperative-statement
```

When the RETURN statement is performed, the next record from the file name is made available for processing by the output procedure.

You must describe the file name in a Data Division SD entry.

If you associate more than one record description with the file name, these records automatically share the same storage; that is, the area is implicitly redefined. After the RETURN statement runs, only the contents of the current record are available. If any data items are beyond the length of the current record, their contents are undefined.

When you use the INTO identifier phrase, the RETURN statement is equivalent to the statement:

RETURN file-name followed by the statement:

MOVE record-name TO identifier.

Moving takes place according to the rules for the MOVE statement without the CORRESPONDING phrase. Any subscripting or indexing you associated with the identifier is evaluated after the record returns and immediately before it moves to the identifier.

The record areas you associated with the file name and identifier must not be the same storage area.

After all records have returned from the file name, the AT END imperative statement is performed, and no more RETURN statements can be performed.

USING THE TABLE HANDLING FACILITIES

You will probably often use tables in data processing. A table is a set of logically consecutive items, each of which has the same data description as the other items in the set. The items in a table can be described as separate and adjacent items; however, you might find this approach unsatisfactory for two reasons. From a documentation standpoint, the similarity of the data items is not apparent; secondly, repetitive coding to reference unique data names becomes a severe problem. Thus, you can use a method of data reference which lets you refer to all or to part of one table as a single object.

Table Handling Concepts

In COBOL, you define a table with an OCCURS clause in its data description. The OCCURS clause specifies that the named item is to be repeated as many times as you state. The named item is considered a table element, and its name and description apply to each repetition (or occurrence) of the item. Because the occurrences are not given unique data names, you can refer to a particular occurrence only by giving the data name of the table element, together with the occurrence number of the desired item within the element.

The occurrence number is known as a subscript and when you supply the occurrence number of individual table elements, you are subscripting. You can also use a related technique called indexing for table references. Both subscripting and indexing are described in the following sections.

Table Definition

COBOL lets you define and use tables in one, two, or three dimensions.

To define a one-dimensional table, you write an OCCURS clause as part of the definition of a table element. You must not place the OCCURS clause in the description of a group item that contains the table element; that is, you must not use an OCCURS clause for a 01-level item. For example:

```
01  TABLE-ONE.
    05  ELEMENT-ONE OCCURS 3 TIMES.
        10 ELEMENT-A PIC X(4).
        10 ELEMENT-B PIC 9(4).
```

TABLE-ONE is the group item that contains the table. ELEMENT-ONE names the table element of a one-dimensional table that occurs three times. ELEMENT-A and ELEMENT-B are elementary items subordinate to ELEMENT-ONE.

To define a two-dimensional table, define a one-dimensional table within each occurrence of another one-dimensional table.

For example:

```
01  TABLE-TWO.
    05  ELEMENT-ONE OCCURS 3 TIMES.
        10 ELEMENT-TWO OCCURS 3 TIMES.
           15 ELEMENT-A PIC X(4).
           15 ELEMENT-B PIC 9(4).
```

ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-TWO is an element of a two-dimensional table that occurs three times within each occurrence of ELEMENT-ONE. ELEMENT-A and ELEMENT-B are elementary items subordinate to ELEMENT-TWO.

To define a three-dimensional table, define a one-dimensional table within each occurrence of another one-dimensional table, which is itself contained within each occurrence of another one-dimensional table. For example:

```
01  TABLE-THREE.
    05  ELEMENT-ONE OCCURS 3 TIMES.
        10 ELEMENT-TWO OCCURS 3 TIMES.
        15 ELEMENT-THREE OCCURS 2 TIMES
           PICTURE X(8).
```

In this example, TABLE-THREE is the group item that contains the table. ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-TWO is an element of a two-dimensional table that occurs three times within each occurrence of ELEMENT-ONE. ELEMENT-THREE is an element of a three-dimensional table that occurs two times within each occurrence of ELEMENT-TWO. Figure 13-1 shows the storage layout for TABLE-THREE.

ELEMENT-ONE (Occurs Three Times)	ELEMENT-TWO (Occurs Three Times)	ELEMENT-THREE (Occurs Two Times)
ELEMENT-ONE (1)	ELEMENT-TWO (1, 1)	ELEMENT-THREE (1, 1, 1)
		ELEMENT-THREE (1, 1, 2)
	ELEMENT-TWO (1, 2)	ELEMENT-THREE (1, 2, 1)
		ELEMENT-THREE (1, 2, 2)
	ELEMENT-TWO (1, 3)	ELEMENT-THREE (1, 3, 1)
		ELEMENT-THREE (1, 3, 2)
ELEMENT-ONE (2)	ELEMENT-TWO (2, 1)	ELEMENT-THREE (2, 1, 1)
		ELEMENT-THREE (2, 1, 2)
	ELEMENT-TWO (2, 2)	ELEMENT-THREE (2, 2, 1)
		ELEMENT-THREE (2, 2, 2)
	ELEMENT-TWO (2, 3)	ELEMENT-THREE (2, 3, 1)
		ELEMENT-THREE (2, 3, 2)
ELEMENT-ONE (3)	ELEMENT-TWO (3, 1)	ELEMENT-THREE (3, 1, 1)
		ELEMENT-THREE (3, 1, 2)
	ELEMENT-TWO (3, 2)	ELEMENT-THREE (3, 2, 1)
		ELEMENT-THREE (3, 2, 2)
	ELEMENT-TWO (3, 3)	ELEMENT-THREE (3, 3, 1)
		ELEMENT-THREE (3, 3, 2)

Figure 13-1. Storage Layout for TABLE-THREE

Table References

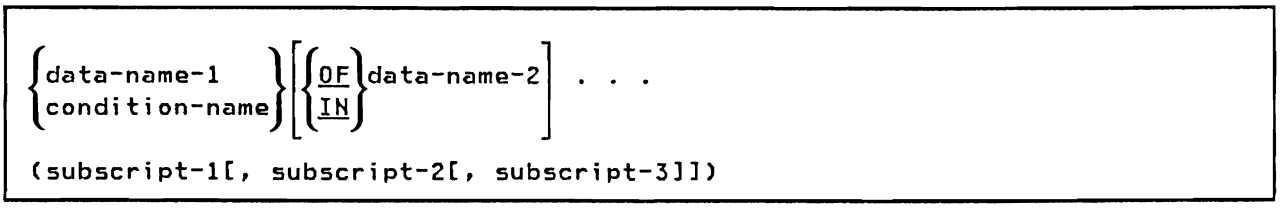
Whenever you refer to a table element or to any item associated with a table element, your reference must indicate which occurrence you want.

For a one-dimensional table, the occurrence number of the desired element gives the complete information. For tables of more than one dimension, you must supply an occurrence number for each dimension. In the three-dimensional table defined in the previous discussion, for example, a reference to ELEMENT-THREE must supply the occurrence number for ELEMENT-ONE, ELEMENT-TWO, and ELEMENT-THREE. You can use either subscripting or indexing, described in the following paragraphs, to supply the necessary references.

Subscripting

Subscripting lets you provide table references by using subscripts. A subscript is an integer value that gives the occurrence number of a table element. You can use subscripts only when you reference an individual item within a table element.

Format



necessary. Note that when you use qualification, it is data-name-1 that is subscripted, not data-name-2.

You can represent the subscript by either a literal or a data name.

You must make a literal subscript an integer, and you must give it a value of 1 or greater. You can give the literal a positive sign or make it unsigned. You cannot use negative subscript values. For example, the following are valid literal subscript references to TABLE-THREE:

```
ELEMENT-THREE (1, 2, 1)
ELEMENT-THREE (2, 2, 1).
```

You must describe a data name subscript as an elementary numeric integer data item. You can qualify a data-name subscript; you cannot subscript or index it. (See *Indexing* later in this chapter.) For example, assuming that SUB1, SUB2, and SUB3 are all items subordinate to SUBSCRIPT-ITEM, valid data-name subscript references to TABLE-THREE are:

```
ELEMENT-THREE (SUB1, SUB2, SUB3)

ELEMENT-THREE IN TABLE-THREE (SUB1 OF
   SUBSCRIPT-ITEM, SUB2 OF SUBSCRIPT-ITEM,
   SUB3 OF SUBSCRIPT-ITEM).
```

You must write the set of one to three subscripts within a balanced pair of parentheses immediately after data-name-1 or its last qualifier. You can optionally place one or more spaces before the opening parenthesis.

When you use more than one subscript, you must separate each subscript from the next either by a space or by a comma and a space.

When you need more than one subscript, write them in the order of successively less-inclusive data dimensions. For example, in the table reference ELEMENT-THREE (3, 2, 1), the first value (3) refers to the occurrence within ELEMENT-ONE, the second value (2) refers to the occurrence within ELEMENT-TWO, and the third value (1) refers to the occurrence within ELEMENT-THREE.

The lowest subscript value you can use is 1. This value points to the first occurrence within the table element. The next sequential elements are pointed to

by subscripts with values 2, 3, and so on. The highest subscript value you can use in any particular table element is the maximum number of occurrences you used in the OCCURS clause. In the example in the preceding paragraph, the highest possible subscript value for ELEMENT-ONE is 3; for ELEMENT-TWO, 3; and for ELEMENT-THREE, 2.

Figure 13-2 shows subscripting using a three-level table. In this example, UNIT-NUMBER could also be referenced as UNIT-NUMBER (3, 4, 5).

```
WORKING-STORAGE SECTION.
77  SUB1 PIC 99.
77  SUB2 PIC 99.
77  SUB3 PIC 99.
77  TEST-IT PIC 99 VALUE 00.
77  TOTAL-RECS PIC 99 VALUE ZEROS.
01  COMPANY-TABLE.
    05  DIVISION-IN OCCURS 10 TIMES.
        10  DIVISION-NAME PIC X(10).
        10  DIVISION-NUMBER PIC 9(4).
        10  SECTION-IN OCCURS 5 TIMES.
            15  UNIT-IN OCCURS 5 TIMES.
                20  UNIT-NAME PIC X(5).
                20  UNIT-NUMBER PIC 9(4).
PROCEDURE DIVISION.
100-START-PROCESSING.

    PERFORM ZERO-OUT-BIG-TABLE VARYING SUB1 FROM 1 BY 1
    UNTIL SUB1 > 10
*   SUB1 IS VARIED LAST BY THE COMPUTER.
    AFTER SUB2 FROM 1 BY 1 UNTIL SUB2 > 5
*   SUB2 IS VARIED  ******2ND****** BY THE COMPUTER.
    AFTER SUB3 FROM 1 BY 1 UNTIL SUB3 > 5.
*    ******SUB3 IS VARIED FIRST BY THE COMPUTER******.
    PERFORM ADDRESS-THE-VARIABLES THRU ATV-EXIT.
    DISPLAY 'VARIABLE TEST-IT = ' TEST-IT.
    STOP RUN.
ZERO-OUT-BIG-TABLE.
    MOVE ZEROS TO UNIT-IN (SUB1, SUB2, SUB3).
ADDRESS-THE-VARIABLES.
    IF UNIT-NUMBER OF UNIT-IN OF SECTION-IN OF DIVISION-IN
    OF COMPANY-TABLE (3, 4, 5) = 0 ADD 1 TO TEST-IT.
ATV-EXIT. EXIT.
```

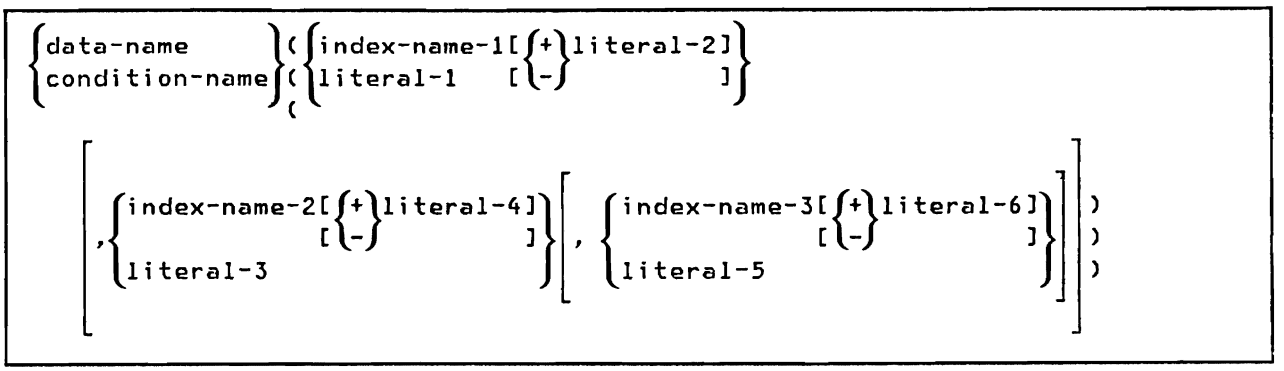
Subscripting

Figure 13-2. Subscripting with a Three-Level Table

Indexing

Indexing lets you provide table references by using indexes. An index is a compiler-generated storage area used to store table element occurrence numbers. The index contains a value that corresponds to an occurrence number.

Format



Each index name identifies an index to be used in table references. You specify the index name through the OCCURS clause.

Each index named is a compiler-generated storage area, 2 bytes long. You can use two forms of indexing:

- Direct
- Relative.

In direct indexing, you give the index name the form of a subscript. In relative indexing, you follow the index name with all of the following items:

- A space
- A + or a -
- Another space
- An unsigned numeric literal.

The literal is considered to be an occurrence number and is converted to an index value before being added to or subtracted from the index name index.

You must make an index value correspond to a table element occurrence not less than 1 or greater than the highest permissible occurrence number. This restriction applies to both direct and relative indexing.

You must initialize an index name through a SET, PERFORM (format 4), or SEARCH ALL statement before using it in a table reference.

You can use one or more index references (direct or relative) together with literal subscripts.

Further information on index names is given later in this chapter in the description of the INDEXED BY phrase of the OCCURS clause.

Restrictions on Subscripting and Indexing

- You must not subscript or index a data name when using it as a subscript or qualifier.
- You cannot subscript when you cannot index.
- You can modify an index only with a PERFORM (format 4), SEARCH, or SET statement.
- When you use a literal in a subscript, you must make it a positive or unsigned integer.
- When you use a literal in relative indexing, you must make it an unsigned integer.

Note: Improper subscripting or indexing can cause an abnormal end of your program. For more information, see *The Abnormal Program End* in Chapter 6.

Table Initialization

You can place static values or dynamic values in a table. Static values remain the same through every run of the object program. When this is true, you can give the initial values of table elements in Working-Storage in one of two ways:

- You can first describe the table as a record containing consecutive subordinate data description entries, each of which contains a VALUE clause for the initial value. You can then redescribe the record through a REDEFINES entry that contains a subordinate entry with an OCCURS clause. Because of the OCCURS clause, the subordinate entries of the redefined entry are repeated. For example:

```
01   TABLE-ONE.
     05  ELEMENT-ONE PICTURE X VALUE '1'.
     05  ELEMENT-TWO PICTURE X VALUE '2'.
     05  ELEMENT-THREE PICTURE X VALUE '3'.
     05  ELEMENT-FOUR PICTURE X VALUE '4'.
01   TABLE-TWO REDEFINES TABLE-ONE.
     05  OCCURS-ELEMENT OCCURS 4 TIMES
         PICTURE X.
```

- If the subordinate entries do not require separate handling, you can give the VALUE of the entire entry in the entry that names the table. The lower-level entries then contain OCCURS clauses, and show the hierarchy of the table structure. You must not place VALUE clauses in the subordinate entries. For example:

```
01   TABLE-ONE VALUE '1234'.
     05  TABLE-TWO OCCURS 4 TIMES
         PICTURE X.
```

Dynamic values might change during one run of the object program or from one run to another. If the dynamic values are always the same at the beginning of

values. If the initial values change from one run to the next, you can define the table without initial values, and you can place the changed values in the table before any table reference is made.

Figure 13-3 shows two ways of initializing a table with zeros.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77  SUB1 PIC 999.
77  TOTAL-HOLD PIC 99 VALUE 57.
77  HOLD-2 PIC 99 VALUE 10.
77  HOLD-THE-SUM PIC 99 VALUE ZERO.
01  TABLE-ELEMENT.
    03  ELEMENTS-OF-TABLE OCCURS 100 TIMES PIC 9.

PROCEDURE DIVISION.
100-START-PROCESSING.
    PERFORM SAMPLE-PERFORM THRU PERFORM-EXIT VARYING SUB1
    FROM 1 BY 1 UNTIL SUB1 > 100.
    ADD TOTAL-HOLD HOLD-2 GIVING HOLD-THE-SUM.
*   THIS ADD STATEMENT IS EXECUTED AFTER THE PERFORM IS DONE.
    DISPLAY 'TOTAL OF TWO VARIABLES = ' HOLD-THE-SUM.
    PERFORM ANOTHER-WAY-TO-INITIALIZE THRU AWTI-EXIT.
*   ************************
*   THE TABLE WILL BE ALL ZEROS AND SHOULD PRINT AS SUCH.
*
*
*   ************************
    DISPLAY '------------------THE-------TABLE----------'.
    STOP RUN.
SAMPLE-PERFORM.
    MOVE ZEROS TO ELEMENTS-OF-TABLE (SUB1).

PERFORM-EXIT. EXIT.
ANOTHER-WAY-TO-INITIALIZE.
    MOVE ZEROS TO TABLE-ELEMENT.
AWTI-EXIT. EXIT.
```

Initializing Table to Zeros

Figure 13-3. Initializing a Table with Zeros

Data Division Table Handling

The COBOL Data Division clauses you use for Table Handling are the OCCURS clause and the USAGE IS INDEX clause.

OCCURS Clause

The OCCURS clause eliminates the need to use separate entries for repeated data items. It also supplies the information necessary for the use of subscripts or indexes. The format of the OCCURS clause is as follows:

Format

```
[OCCURS {integer-1 TO integer-2 TIMES DEPENDING ON data-name-1}
        {integer-2 TIMES                                     }

   [{ASCENDING }KEY IS data-name-2 [, data-name-3] . . .] . . .
    {DESCENDING}

   [INDEXED BY index-name-1 [, index-name-2] . . .]]
```

The subject of an OCCURS clause is the data name of the data item containing the OCCURS clause. Except for the OCCURS clause itself, data description clauses used with the subject apply to each occurrence of the item described.

When you refer to the subject in a statement other than a SEARCH statement, or if the subject is the object of a REDEFINES clause, you must subscript or index the subject. When you subscript or index the subject, it refers to one occurrence within the table element.

Whenever you refer to the subject in a SEARCH statement or the subject is the object of a REDEFINES clause, you must not subscript or index the subject. When you do not subscript or index the subject, it represents the entire table.

You must have less than 32,768 occurrences in the table and it must be less than 32,768 bytes long.

You can qualify all data names that you use in the OCCURS clause. You cannot subscript or index them.

You must make all integers positive, nonzero integers.

You cannot use the OCCURS clause in a data description entry in which you:

- Have a level-01, -66, -77, or -88 number.
- Describe an item of variable size (an item is of variable size if any subordinate entry contains an OCCURS DEPENDING ON clause).
- Describe redefined data items. (However, you can have a redefined item that is subordinate to an item containing an OCCURS clause.) See the *REDEFINES Clause* in Chapter 10.

Fixed-Length Tables

When you do not use an OCCURS DEPENDING ON clause, integer-2 gives the exact number of occurrences.

You must make integer-2 greater than 0 and less than 32,768.

Because you are allowed three subscripts or indexes, you can use three nested levels of this format of the OCCURS clause.

Variable-Length Tables

When you use the OCCURS DEPENDING ON clause, integer-1 represents the minimum number of occurrences, and integer-2 represents the maximum number of occurrences. You must make the value of integer-1 one or greater; you must also make it less than integer-2. You must make integer-2 less than 32,768. The length of the subject item is fixed; it is only the number of repetitions of the subject item that is variable.

Data-name-1 gives the object of the OCCURS DEPENDING ON clause. The object is the data item with a current value that represents the current number of occurrences of the subject item. The object of the OCCURS DEPENDING ON clause:

- Must be described as a positive integer; that is, if you describe data-name-1 as a signed item, then it must contain positive data at running time.
- Must not occupy any storage position within the range of this table; that is, the object must not occupy any storage position from the first character position in this table through the last character position in this record description entry.
- Must contain a value within the range of integer-1 and integer-2, inclusive.

The value of the object of the OCCURS DEPENDING ON clause specifies that part of the table element available to the object program. Items with occurrence numbers that are larger than the value of the object are not available. If during run time, the value of the object is reduced, the contents of items with occurrence numbers that are larger than the new value of the object are unpredictable.

When you refer to a group item containing a subordinate OCCURS DEPENDING ON item, the current value of the object determines which part of the table area is used in the operation.

In one record description entry, you can follow any entry that contains an OCCURS DEPENDING ON clause only by items subordinate to it. You cannot have the OCCURS DEPENDING ON clause subordinate to another OCCURS clause; however, you can have the format 1 OCCURS clause subordinate to the OCCURS DEPENDING ON clause; in this way, you can specify a table of up to three dimensions.

ASCENDING/DESCENDING KEY Phrase

The ASCENDING/DESCENDING KEY phrase specifies that the repeated data is arranged in ascending or descending order according to the values contained in data-name-2, data-name-3, and so on. You should list the data names in descending order of significance. The SEARCH ALL statement uses the ASCENDING/DESCENDING KEY data items to search the table element.

Determine the order by using the rules for comparison of operands. (See *Relation Condition* in Chapter 11.)

You must make data-name-2 the name of the subject entry or the name of an entry subordinate to the subject entry. If data-name-2 names the subject entry, that entire entry becomes an ASCENDING/DESCENDING KEY. If data-name-2 does not name the subject entry, then data-name-2, data-name-3, and so on:

- Must be subordinate to the subject of the table entry itself
- Must not be subordinate to any other entry that contains an OCCURS clause
- Must not themselves contain an OCCURS clause.

The following example illustrates the specification of ASCENDING/DESCENDING KEY data items:

```
WORKING-STORAGE SECTION.
01   CURRENT-WEEK PICTURE 99.
01   TABLE-RECORD.
     05  EMPLOYEE-TABLE OCCURS 100 TIMES
         ASCENDING KEY IS WAGE-RATE
         EMPLOYEE-NO INDEXED BY A, B.
         10 EMPLOYEE-NAME PIC X(20).
         10 EMPLOYEE-NO PIC 9(6).
         10 WAGE-RATE PIC 9999V99.
         10 WEEK-RECORD OCCURS 52 TIMES
            ASCENDING KEY IS WEEK-NO
            INDEXED BY C.
            15 WEEK-NO PIC 99.
            15 AUTHORIZED-ABSENCES PIC  9.
            15 UNAUTHORIZED-ABSENCES PIC 9.
            15 LATENESSES PIC  9.
```

The keys for EMPLOYEE-TABLE are subordinate to that entry, and the key for WEEK-RECORD is subordinate to that subordinate entry.

When you use the ASCENDING/DESCENDING KEY phrase, the following rules apply:

- You must list keys in decreasing order of significance

- You must make sure that the data present in the table is arranged in ascending/descending key sequence according to the collating sequence in use.

In the preceding example, records in EMPLOYEE-TABLE must be arranged in ascending order of WAGE-RATE and in ascending order of EMPLOYEE-NO within WAGE-RATE. Records in WEEK-RECORD must be arranged in ascending order of WEEK-NO. If they are not, SEARCH ALL statement results will be unpredictable.

INDEXED BY Phrase

The INDEXED BY phrase gives the indexes that you can use with this table element. You must use the INDEXED BY phrase if you use indexing to refer to this table element.

Each index name must follow the rules for formation of a user-defined word; you must make at least one character alphabetic. Each index name specifies an index to be created by the compiler for use by the program. These index names are not data names, and you do not identify them elsewhere in the COBOL program; instead, you can regard them as special registers for the use of this object program only. They are not regarded as data or part of any data hierarchy, so you must make each one unique. You can reference an index name only with a PERFORM, SET, or SEARCH statement; as a parameter in the USING phrase in a CALL statement; or in a relational condition comparison.

USAGE IS INDEX Clause

The USAGE IS INDEX clause specifies that the data item named has an index format. Such an item is an index data item.

Format

```
[USAGE IS] INDEX
```

An index data item is an elementary item that you can use to save index name values for future reference. Through the SET statement, you can assign an index name value to an index data item. An index value corresponds to the displacement for an occurrence number in the table, that is (occurrence-number 1) * entry length.

You can directly refer to an index data item only in:

- A SEARCH statement
- A SET statement
- A relation condition
- The USING phrase of the Procedure Division header

- The USING phrase of the CALL statement.

You can make an index data item part of a group item you referred to in a MOVE statement or an input/output statement.

An index data item saves values equivalent to table occurrences; however, you do not necessarily define it as part of any table. Thus, when you directly reference an index data item in a SEARCH or SET statement or indirectly in a MOVE or an input/output statement, there is no conversion of values when the statement is performed.

You can write the USAGE IS INDEX clause at any level. If you described a group item with the USAGE IS INDEX clause, it is the elementary items within the group that are index data items. The group itself is not an index data item, and you cannot use the group name in SEARCH and SET statements or in relation conditions. You cannot have a USAGE clause of an elementary item that contradicts the USAGE clause of a group to which the item belongs.

You cannot have an index data item that is a conditional variable. The index data item cannot have a subordinate level-88 entry.

You cannot use the SYNCHRONIZED, JUSTIFIED, PICTURE, BLANK WHEN ZERO, or VALUE clauses to describe group or elementary items that you described with the USAGE IS INDEX clause.

Procedure Division Table Handling

In the Procedure Division, you can use the SEARCH and SET statements with indexed tables. There are also special rules involving table handling elements when you use them in relation conditions.

Relation Conditions

Comparisons involving index names or index data items conform to the following situations:

- When you compare two index names, you are actually comparing the corresponding occurrence numbers.
- When you compare an index name with a data item (other than an index data item) or an index name with a literal, the occurrence number that corresponds to the value of the index name is compared with the data item or literal.
- When you compare an index data item with an index name or another index data item, the actual values are compared without conversion. Results of any other comparison involving an index data item are undefined.

Figure 13-4 shows comparisons you can make for index names and index data items.

First Operand	Second Operand: Index-name[1]	Second Operand: Index Data Item[2]	Second Operand: Data name (numeric integer only)	Second Operand: Numeric Literal (integer only)
Index-name[1]	Compare occurrence number	Compare without conversion	Compare occurrence number with data-name	Compare occurrence number with literal
Index Data Item[2]	Compare without conversion	Compare without conversion	Invalid	Invalid
Data name (numeric integer only)	Compare occurrence number with data name	Invalid		
Numeric Literal (integer only)	Compare occurrence number with literal	Invalid		

[1]See *OCCURS Clause* earlier in this chapter.
[2]See *USAGE IS INDEX Clause* earlier in this chapter.

Figure 13-4. Comparisons You Can Make for Index Names and Index Data Items

SEARCH Statement

The SEARCH statement searches a table for an element that satisfies the given condition, and adjusts the associated index to indicate that element. The formats for the SEARCH statement are:

Format 1

```
SEARCH identifier-1 [VARYING {identifier-2  }] [AT END imperative-statement-1]
                             {index-name-1  }

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

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

Format 2

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

     {data-name-1      {IS EQUAL TO} {identifier-3  }}
WHEN {                 {IS =      } {literal-1     }}
     {                              {arithmetic-expression-1}}
     {condition-name-1                                       }

   [ {data-name-2      {IS EQUAL TO} {identifier-4  }}  ]
AND{                   {IS =      } {literal-2     }}  } ...
   {                              {arithmetic-expression-2}}
   {condition-name-2                                       }

{imperative-statement-2}
{NEXT SENTENCE          }
```

You must include an OCCURS clause with the INDEXED BY phrase in the Data Division description of identifier-1.

When you use identifier-1 in the SEARCH statement, identifier-1 must refer to all occurrences within the table element. You must not subscript or index identifier-1.

You can make identifier-1:

- A data item that is subordinate to a data item that contains an OCCURS clause

- Part of a two- or three-dimensional table. For each dimension of the table in this case, you must use an INDEXED BY phrase in the data description entry.

Performing the SEARCH statement changes only the value of the index name associated with identifier-1 (and, if present, of index-name-1 or identifier-2); to search an entire two- or three-dimensional table, you must perform a SEARCH statement for each dimension. Before each performance of a SEARCH statement, you must perform SET statements to reinitialize the associated index names.

In the AT END and WHEN phrases, control passes to the next sentence after the imperative statement is performed if you did not end any of the given imperative statements with a GO TO statement.

Format 1

Performance of a format 1 SEARCH statement causes a serial search to be performed, beginning at the current index setting.

If the value you use for the index name associated with identifier-1 is not greater than the highest possible occurrence number, the following actions take place when the search begins:

1. The condition(s) in the WHEN phrase are evaluated in the order you wrote them.

2. If none of the conditions are satisfied, the index name for identifier-1 is incremented to correspond to the next table element, and step 1 is repeated.

3. If upon evaluation one of the WHEN conditions is satisfied, the search ends immediately, and the imperative statement associated with that condition is performed. The index name points to the table element that satisfied the condition.

4. If the end of the table is reached (that is, the incremented index name value is greater than the highest possible occurrence number) without the WHEN condition being satisfied, the search ends as described in the next paragraph.

If, when the search begins, the value of the index name associated with identifier-1 is greater than the highest possible occurrence number, the search immediately ends, and if used, the AT END imperative statement is performed. If you leave out the AT END phrase, control passes to the next sentence.

You can make each WHEN phrase condition any condition as described under *Conditional Expressions* in Chapter 11.

VARYING Index-Name-1 Phrase: When you leave out the VARYING index-name-1 phrase, the first (or only) index name for identifier-1 is used for the search. When you use the VARYING index-name-1 phrase, one of the following actions takes place:

- If index-name-1 is an index for identifier-1, this index is used for the search; otherwise, the first (or only) index name is used.

- If index-name-1 is an index for another table element, the first (or only) index name for identifier-1 is used for the search; the occurrence number represented by index-name-1 is incremented by the same amount as the search index name and at the same time.

VARYING Identifier-2 Phrase: When you use this phrase, the first (or only)index name for identifier-1 is used for the search.

You must make identifier-2 either an index data item or an elementary integer item. During the search, one of the following actions takes place:

- If identifier-2 is an index data item, then whenever the search index is incremented, the specified index item is incremented by the same amount at the same time.

- If identifier-2 is an integer data item, then whenever the search index is incremented, the given data item is incremented by 1 at the same time.

Figure 13-5 is a flowchart of a format 1 SEARCH operation containing two WHEN phrases.

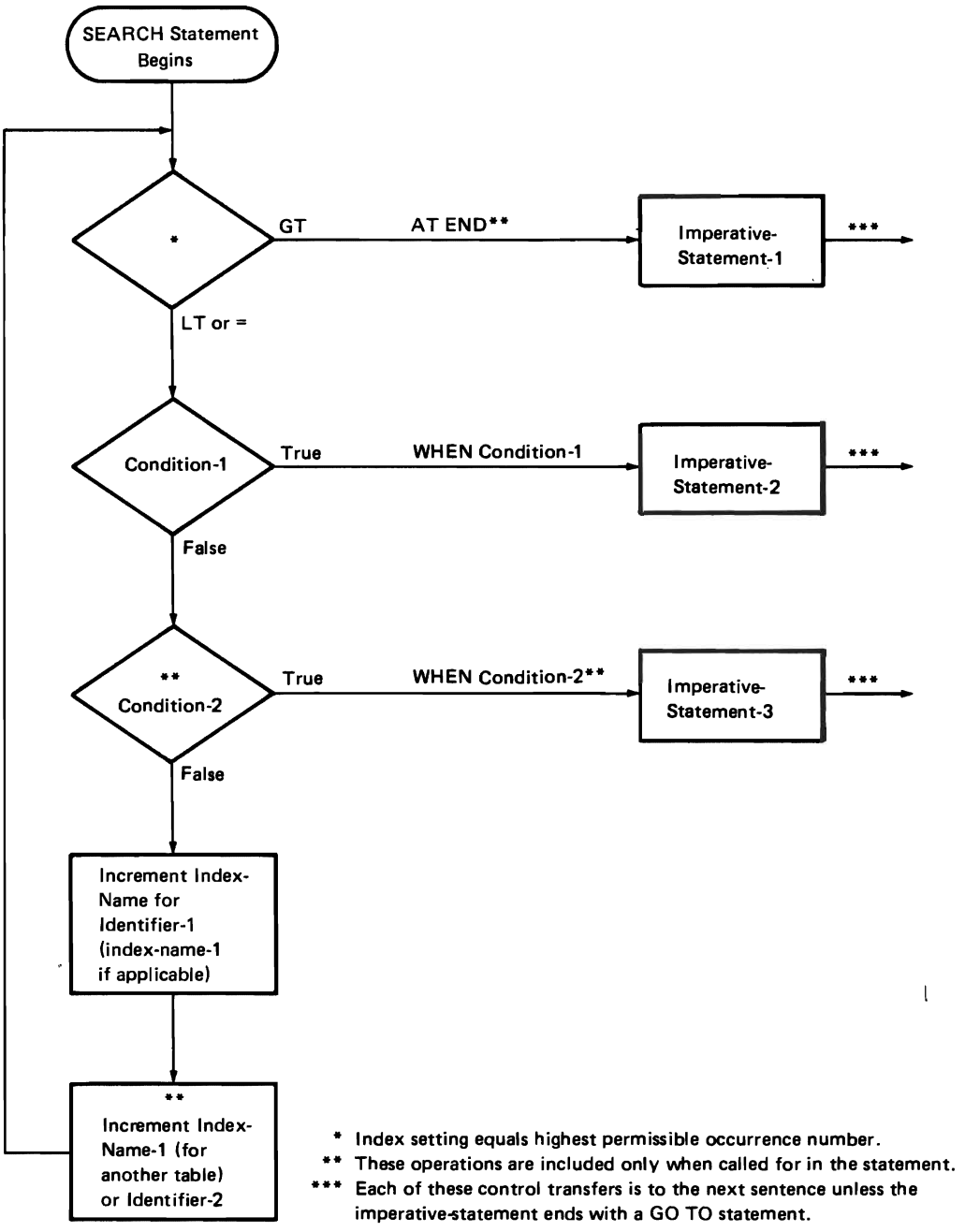


Figure 13-5. Format 1 SEARCH with Two WHEN Phrases

Format 2

Performance of a format 2 SEARCH ALL statement causes a serial search to be performed, beginning with the first element of the table. You need not initialize the search index with SET statements, because its setting is varied during the search operation. The index used is always the index that is associated with the first index name you used in the OCCURS clause.

If the WHEN phrase cannot be satisfied for any setting of the index within this range, the search is unsuccessful. If you use the AT END phrase, the AT END imperative statement is performed. If you do not use the AT END phrase, control passes to the next sentence. In either case, the final setting of the index is not predictable.

If the WHEN phrase can be satisfied, control passes to imperative-statement-2, and the index contains a value indicating an occurrence that allows the WHEN condition(s) to be satisfied.

WHEN Condition-Name-1 Phrase: If you use the WHEN condition name-1 phrase, you must give each condition name a single value only, and you must associate each with an ASCENDING/DESCENDING KEY identifier for this table element.

WHEN Relation Condition Phrase: If you use a WHEN relation condition, the following considerations apply:

- Data-name-1 or data-name-2 must give an ASCENDING/DESCENDING KEY data item in the identifier-1 table element and must be indexed by the first identifier-1 index name, along with other indexes or literals as required. You can qualify each data name.

- Identifier-3 and identifier-4 must not be an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is indexed by the first index name for identifier-1.

- Literal-1 or literal-2 must be a positive or an unsigned numeric integer.

- Arithmetic-expression-1 or arithmetic-expression-2 can be any of those defined under *Arithmetic Expressions* in Chapter 11 with the following restriction: You must not have any identifier in the arithmetic expression that is an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is indexed by the first index name for identifier-1.

- When you use an ASCENDING/DESCENDING KEY data item either explicitly or implicitly in the WHEN option, you must also use all preceding ASCENDING/DESCENDING KEY data names for identifier-1.

The results of a SEARCH ALL operation are predictable only when both of the following apply:

- You must place the data in the table in ascending or descending key sequence

- You provide a unique table reference with the contents of the ASCENDING/DESCENDING keys used in the WHEN clause.

Notes:

1. *You cannot use index data items as subscripts or indexes, because of the restrictions on direct reference to them. Using a direct indexing reference together with a relative indexing reference for the same index name lets you reference two different occurrences of a table element for comparison.*

2. *When you make the object of the VARYING phrase an index name for another table element, one format 1 SEARCH statement looks at two table elements at once.*

3. *One format 4 PERFORM statement can search an entire multidimensional table.*

4. *To ensure correct performance of a PERFORM or SEARCH statement for a variable-length table, you must make sure that the object of the OCCURS DEPENDING ON clause (data-name-1) contains a value that correctly gives the current length of the table.*

SEARCH Example

The following example searches an inventory table for items that match those from the input data. The key is INVENTORY-NUMBER.

```
DATA DIVISION.
FILE SECTION.
FD  SALES-DATA
    BLOCK CONTAINS 1 RECORDS
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS STANDARD
    DATA RECORD IS SALES-REPORTS.
01  SALES-REPORTS  PIC X(80).
FD  PRINTED-REPORT
    BLOCK CONTAINS 1 RECORDS
    RECORD CONTAINS 132 CHARACTERS
    LABEL RECORDS OMITTED
    DATA RECORD IS PRINTER-OUTPUT.
01  PRINTER-OUTPUT  PIC X(132).
FD  INVENTORY-DATA
    BLOCK CONTAINS 1 RECORDS
    RECORD CONTAINS 40 CHARACTERS
    LABEL RECORDS STANDARD
    DATA RECORD IS INVENTORY-RECORD.
01  INVENTORY-RECORD.
    03  I-NUMBER          PIC 9(4).
    03  INV-ID            PIC X(26).
    03  I-COST            PIC 9(8)V99.
WORKING-STORAGE SECTION.
77  EOF-SW                PIC X     VALUE 'N'.
77  EOF-SW2               PIC X     VALUE 'N'.
77  SUB1                  PIC 99.
77  RECORDS-NOT-FOUND     PIC 9(5)  VALUE ZEROS.
77  TOTAL-COSTS           PIC 9(10) VALUE ZEROS.
01  HOLD-INPUT-DATA.
    03  INVENTORY-NUMBER  PIC 9999.
    03  PURCHASE-COST     PIC 9(4)V99.
    03  PURCHASE-DATE     PIC 9(6).
    03  FILLER            PIC X(64).
01  PRINTER-SPECS.
    03  PRINT-LINE.
        05  OUTPUT-ITEM-NUMBER    PIC ZZZ9.
        05  FILLER                PIC X(48)   VALUE SPACES.
        05  TOTAL-COSTS-0         PIC $(8).99.
01  PRODUCT-TABLE.
    05  INVENTORY-NUMBERS OCCURS 50 TIMES
        ASCENDING KEY ITEM-NUMBER
        INDEXED BY INDEX-1.
            07  ITEM-NUMBER        PIC 9(4).
            07  ITEM-DESCRIPTION   PIC X(26).
            07  ITEM-COST          PIC 9(8)V99.
```

```
PROCEDURE DIVISION.
100-START-IT.
    OPEN INPUT SALES-DATA INVENTORY-DATA OUTPUT PRINTED-REPORT.
    MOVE HIGH-VALUES TO PRODUCT-TABLE.
READ-INVENTORY-DATA.
    READ INVENTORY-DATA AT END MOVE 'Y' TO EOF-SW2.
LOAD-TABLE-ROUTINE.
    PERFORM LOAD-IT VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 > 50.
110-READ-IT.
    READ SALES-DATA INTO HOLD-INPUT-DATA AT END
    MOVE 'Y' TO EOF-SW.
200-MAIN-ROUTINE.
    PERFORM PROCESS-DATA UNTIL EOF-SW = 'Y'.
    MOVE ZEROS TO OUTPUT-ITEM-NUMBER.
    MOVE TOTAL-COSTS TO TOTAL-COSTS-O.
    PERFORM WRITE-REPORT THRU WRITE-REPORT-EXIT.
    DISPLAY 'RECORDS NOT FOUND = ' RECORDS-NOT-FOUND.
    STOP RUN.

PROCESS-DATA.
    SEARCH ALL INVENTORY-NUMBERS AT END
    PERFORM KEY-NOT-FOUND THRU NOT-FOUND-EXIT
    WHEN INVENTORY-NUMBER IS = ITEM-NUMBER (INDEX-1)
        MOVE ITEM-NUMBER (INDEX-1) TO OUTPUT-ITEM-NUMBER
        MOVE ITEM-COST (INDEX-1) TO TOTAL-COSTS-O
        ADD ITEM-COST (INDEX-1) TO TOTAL-COSTS
        PERFORM WRITE-REPORT THRU WRITE-REPORT-EXIT.
    PERFORM 110-READ-IT.
KEY-NOT-FOUND.
    ADD 1 TO RECORDS-NOT-FOUND.
NOT-FOUND-EXIT. EXIT.
LOAD-IT.
    MOVE INVENTORY-RECORD TO INVENTORY-NUMBERS (SUB1).
    PERFORM READ-INVENTORY-DATA.

WRITE-REPORT.
    WRITE PRINTER-OUTPUT FROM PRINTER-SPECS.
WRITE-REPORT-EXIT. EXIT.
*    ********END SAMPLE SEARCH PROGRAM********************
```

SET Statement

The SET statement can:

- Establish reference points for table handling operations by setting index names to values you associated with table elements
- Transfer values between index names and other elementary data items
- Alter the status of external UPSI switches
- Alter the value of conditional variables.

The formats of the SET statement are:

Format 1

```
SET {identifier-1 [, identifier-2] . . .} TO {identifier-3}
    {index-name-1 [, index-name-2] . . .}    {index-name-3}
                                             {integer-1   }
```

Format 2

```
SET index-name-4 [, index-name-5] . . . {UP BY  } {identifier-4}
                                        {DOWN BY} {integer-2   }
```

Format 3

```
SET mnemonic-name-1[, mnemonic-name-2] . . . TO {ON }
                                               {OFF}
```

Format 4

```
SET condition-name-1[, condition-name-2]. . . TO TRUE
```

You relate index names to a given table through the INDEXED BY phrase of the OCCURS clause. Index names that you used in the INDEXED BY phrase are automatically defined.

You can make integer-1 and integer-2 signed. You must make integer-1 positive. You must make all identifiers either index data items or numeric elementary items described as integers; however, you must not have identifier-4 name an index data item.

Format 1 Considerations

When the SET statement is performed, one of the following actions occurs:

- Index-name-1 is converted to a value that corresponds to the same table element to which either index-name-3, identifier-3, or integer-1 corresponds. If identifier-3 is an index data item, no conversion takes place.
- If identifier-1 is an index data item, it is set equal to either the contents of index-name-3 or the contents of identifier-3 when identifier-3 is also an index data item. You cannot use integer-1 in this case.
- If identifier-1 is not an index data item, it is set to an occurrence number that corresponds to the value of index-name-3. You can use neither identifier-3 nor integer-1 in this case.

Format 2 Considerations

When the SET statement is performed, the contents of index-name-4 are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of integer-2 or identifier-4. You must make the value of the index correspond to an occurrence number of an element in the associated table.

IBM Extension

Format 3 Considerations

You must associate each mnemonic name with an external switch (UPSI-0 through UPSI-7), the status of which you can alter.

The status of each external switch is modified to ON if you use the ON keyword, or OFF if you use the OFF keyword.

Format 4 Considerations

You must associate each condition name with a conditional variable.

The literal in the VALUE clause that you associated with the condition name is moved to the conditional variable according to the rules for elementary moves. If you used more than one literal in the VALUE clause, the first literal in that VALUE clause is moved.

End of IBM Extension

LINKAGE BETWEEN MODULES

This section describes standard linkage conventions to use between modules produced by the System/36 language compilers or assemblers: COBOL, FORTRAN IV, and Assembler. If you use standard linkage conventions, you can code routines in the language most appropriate to the function being performed. Figure 13-6 illustrates the standard linkage convention described on the following pages.

```
*      ASSEMBLER MODULE (MODA) CALLS COBOL MODULE (MODB)
*
          EXTRN MODB
MODA      START X'0000'
*
*
*
*
          B     MODB              CALL COBOL MODULE MODB
          DC    AL2(PLIST)        PARAMETER LIST
*                                 CONTROL RETURNS HERE AFTER MODB EXECUTION
*
*
*
*
*
*
*                                 PARAMETER LIST
PLIST     EQU   *
          DC    AL2(SAVA)         ADDRESS OF SAVE AREA
          DC    AL2(PARM1)        ADDRESS OF FIRST PARAMETER
          DC    AL2(PARM2)        ADDRESS OF SECOND PARAMETER
*
*
*
          DC    XL2'FFFF'         END OF PARAMETER LIST INDICATOR
*
*                                 PARAMETERS
PARM1      EQU  *
          DC    CL5'FIRST'
PARM2     EQU   *
          DC    CL6'SECOND'
*
*
*
*                                 SAVE AREA
SAVA      DC    XL1'80'           INDICATOR BYTE - CALLING PROGRAM IS ASSEMBLER
          DC    CL6'MODA'         CALLING PROGRAM'S NAME
          END   MODA
```

```
*         SAMPLE SYSTEM/XX LINKAGE
*
*         COBOL MODULE (MODC) CALLS ASSEMBLER MODULE (MODD)
*
XR1       EQU   1                                   EQUATE REGISTER VALUES
XR2       EQU   2
ARR       EQU   8
IAR       EQU   16
*
          ENTRY MODD
MODD      START X'0000'
          ST    SAVAR1,XR1                          SAVE INDEX REGISTER ONE VALUE
          LA    SAVA,XR1                            POINT XR1 TO MODD'S SAVE AREA
          USING SAVA,XR1                            ESTABLISH XR1 AS BASE REGISTER
          ST    SAVAR2(,XR1),XR2                    SAVE INDEX REGISTER TWO VALUE
          ST    SAVART(,XR1),ARR                    SAVE ARR VALUE
          L     SAVART(,XR1),XR2                    POINT XR2 TO ARR VALUE
          L     1(,XR2),XR2                         POINT TO SECOND BYTE OF ADDRESS
          ALC   SAVART(,XR1),TWO(,XR1)              JUMP ARR VALUE PAST PARAMETER
          L     3(,XR2),XR1                         GET ADDR OF PARAM 1 INTO XR1
          L     5(,XR2),XR2                         GET ADDR OF PARAM 2 INTO XR2
*
*         BODY  OF  ROUTINE. . .
*         RETURN TO CALLING PROGRAM
*
          L     SAVAR2(,XR1),XR2                    RESTORE XR2 VALUE
          L     SAVAR1(,XR1),XR1                    RESTORE XR1 VALUE
          L     SAVART,IAR                          BRANCH TO NEXT SEQ. INSTRUCTION
*         SAVE AREA
SAVA      DC    XL1'30'
          DC    CL6'MODD'
SAVAR1    DC    XL2'00'
SAVAR2    DC    XL2'00'
SAVART    DC    AL2(00)
*         WORK VALUE
TWO       DC    IL2'2'
          END   MODD
```

Figure 13-6. Standard Linkage

Standard Linkage

Standard linkage is accomplished as follows:

1. You must define a save area for each module as follows:

   For a subprogram:

Byte 0	Bit 0	0	Not a main program
	Bits 1-3	000	FORTRAN IV
		001	COBOL
		011	Assembler
	Bits 4-7	0000	Reserved
Bytes 1-6			EBCDIC name, left-adjusted
Bytes 7-8			Value of index register 1 (XR1) at entry
Bytes 9-10			Value of index register 2 (XR2) at entry
Bytes 11-12			Return point in calling program

   For a main program:

Byte 0	Bit 0	1	Main program
	Bits 1-3	000	FORTRAN IV
		001	COBOL
		011	Assembler
	Bits 4-7	0000	Reserved
Bytes 1-6			EBCDIC name, left-adjusted

   Note: Main program refers to the program with the highest level of control.

2. You must define one or more parameter lists for each module that calls another module as follows:

Bytes 0-1	Address of save of area in this program
Bytes 2-3	Address of first parameter
Bytes (2n) - (2n + 1)	Address of nth parameter
Bytes (2n + 2)	XL2'FFFF' to indicate end of parameter list

   Notes:

   a. *You must include the first 2 bytes, as well as the end-of-parameter-list indicator (XL2'FFFF') in all parameter lists. If no parameters are to be passed, the parameter list is only 4 bytes long. In this case, bytes 3 and 4 are hex FFFF.*

b. *Addresses in parameter lists refer to the first byte (byte with the lowest address) of the item.*

3. When control reaches a program entry point, the address recall register (ARR) must point to a 2-byte field containing the first byte of the parameter list.

   The assembler language code to call a COBOL subprogram would normally be as follows:

```
            EXTRN   SUBR
            B       SUBR
            DC      AL2(PARAMS)
RETNPT      EQU     *
```

   Note: The pointer to the parameter list points to the left byte of the save area address.

4. Normal return is accomplished by placing in the hardware instruction address register (IAR) a value that is 2 larger than the contents of the ARR when the program was entered.

5. You must save index registers 1 and 2 (XR1 and XR2) upon entry into the called program's save area, and restore them at exit.

6. You need not restore the address recall register, but you must determine the return address and place it in the called program's save area.

USING INTER-PROGRAM COMMUNICATION

You can often solve complex data processing problems by using separately compiled but logically interdependent programs that at run time, form logical and physical subdivisions of a single run unit. A run unit is the total machine-language program necessary to solve a data processing problem; it includes one or more object programs, and can include object programs from source programs written in System/36 FORTRAN IV, System/36 Assembler, and System/36 COBOL.

Subprogram Linkage Concepts

When you subdivide the solution of a problem into more than one program, the constituent programs must be able to communicate with each other through transfers of control or through reference to common data.

Transfers of Control

In the Procedure Division, a calling program can transfer control to a called program, and a called program can itself transfer control to yet another called program; however, a called program must not directly or indirectly call the program that called it. For example, if program A calls program B, program B calls program C, and program C then calls program A, the results will be unpredictable.

When control passes to a called program, the program runs normally. When a called program completes processing, the program can do any of the following:

- Transfer control back to the calling program
- Call another program
- End the run unit.

Common Data

Program interaction might require that both programs have access to the same data.

In a calling program, describe the common data items in the same manner as other File and Working-Storage Section items. Allocate storage for these items in the calling program.

Describe common data items in the Linkage Section of a called program but do not allocate storage to them in the called program. If a calling program is also a called program, you can describe common data items in the Linkage Section of the calling program. In this case, do not allocate storage for these items in this calling program itself, but rather in the program that called the calling program. For example, program A calls program B, which calls program C. You can describe data items in program A in the Linkage Sections of programs B and C and make the one set of data available to all three programs.

When control transfers from the calling to the called program, you must furnish a list of the common data items in both programs. The sequence of identifiers in both lists determines the match of identifiers between the calling and called programs. A corresponding pair of identifiers in the list names a single set of data that is available to both programs. While the called program is running, any reference to one of these identifiers is a reference to the corresponding data of the calling program.

COBOL Language Considerations

In the Data Division of the source programs, you define the common data items to be used by both the calling and called programs. In the calling program, you can define these items in the File, Working-Storage, or Linkage Sections. In the called program, you must define these items in the Linkage Section. You need not give common data items the same name and data description, but you must make them the same number of characters.

In the Procedure Division, you use the USING phrase to make the list of common data items. The USING phrase names those data items available to both programs. In the called program, only those items named in the USING list of the called program are available from the data storage of the calling program. Figure 13-7 illustrates this concept.

A CALL statement in the calling program transfers control to the first nondeclarative procedural statement in the called program. When the called program has completed running, control returns to the calling program by an EXIT PROGRAM statement. You can end the entire run unit with a STOP RUN statement in either program.

Calling Program Description	Called Program Description
WORKING-STORAGE SECTION. 01 PARAM-LIST 05 PARTCODE PIC A. 05 PARTNO PIC X(4). 05 U-SALES PIC 9(5). . . . PROCEDURE DIVISION . . . CALL 'CALLPG' USING PARAM-LIST	LINKAGE SECTION. 01 USING-LIST. 10 PART-ID PIC X(5). 10 SALES PIC 9(5). . . . PROCEDURE DIVISION USING USING-LIST.

Figure 13-7. Common Data Items in Subprogram Linkage

Note: In the calling program, the code for parts (PARTCODE) and the part number (PARTNO) are referred to separately. In the called program, the code for parts and the part number are combined into one data item (PART-ID); therefore, in the called program, a reference to PART-ID is the only valid reference to them.

System Considerations

The main COBOL program and all called programs are part of the same load module. When control transfers to the called program, it is already in storage, and a branch to the called program takes place. Subsequent performing of the CALL statement makes the called program available in its last-used state, if segmentation on the called program has not been requested.

Data Division Subprogram Linkage

In the Data Division of a called program, you specify in the Linkage Section those data items that are common with the calling program.

Format

```
LINKAGE SECTION.
    [data-item-description-entry] . . .

    [record-description-entry] . . .
```

The Linkage Section has meaning only if this object program functions under control of a CALL statement that contains the USING phrase.

The Linkage Section describes data available within the calling program and referred to in both the calling and called programs. Do not allocate space in the called program for items described in the Linkage Section. Procedure Division references to these data items are resolved at object time by equating the reference in the called program to the location used in the calling program. For index names, no such correspondence is established. Index name references in the calling and called programs always refer to separate indexes.

You can refer to items defined in the Linkage Section in the Procedure Division only if the items are one of the following:

- Operands of a USING phrase in this program
- Data items subordinate to such a USING phrase operand
- Items associated with such a USING operand (such as condition names or index names).

You must make each Linkage Section record name and nonconsecutive data name unique, because you cannot qualify them. Descriptions of each clause valid in the Linkage Section are given under *Data Description* in Chapter 10. In addition, record description entries and data item entries must be considered. A brief description of each is provided.

Record Description Entries

You must group items that have a hierarchical relationship with one another into level-01 records according to the rules for formation of record descriptions. You can use data description clauses to complete the description of the entry. Except for level-88 condition names, you must not use the VALUE clause.

Data Item Description Entries

You can define items that have no hierarchical relationship with each other as nonconsecutive items with level number 77. You must use the following clauses:

- Level number 77
- Data name
- PICTURE or USAGE IS INDEX.

Other data description clauses are optional and, when necessary, can complete the description of the item. Except for level-88 condition names, you must not use the VALUE clause.

Procedure Division Subprogram Linkage

In the Procedure Division, you use the CALL statement to transfer control between COBOL object programs.

The USING phrase can be used in the CALL statement and in the Procedure Division header of the called program to reference common data.

You can use the EXIT PROGRAM statement to end processing of the called program. You can use the STOP RUN statement to end the run unit.

CALL Statement

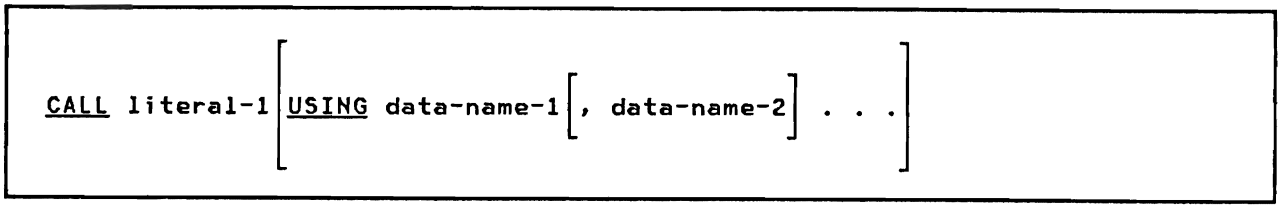
You use the CALL statement to transfer control from one object program to another within the run unit. You must include a CALL statement in the calling program at the point where another program is to be called.

Performing the CALL statement passes control to the first nondeclarative instruction of the called program. Control returns to the calling program at the instruction following the CALL statement.

You can include CALL statements in called programs; however, a called program must not contain a CALL statement that directly or indirectly calls itself.

You can call up to 100 subprograms in a calling program.

Format



Literal-1 must be nonnumeric and must conform to the rules for formation of a program name. The first six characters of the literal make the correspondence between the calling program and the called program. You must make the literal the program name of the called subprogram.

Performing the CALL statement passes control to the called subprogram. The first time a called program is entered, its state is that of a fresh copy of the program. Each subsequent time a called program is entered, the state is as it was upon the last exit from that program as long as no segmentation or overlays are in the program. Thus, you should consider that some of the following statements may be in their last-used state:

- GO TO statements that you have altered
- Data items
- PERFORM statements.

USING Phrase

The USING phrase makes data items from a calling program available to the called program. If the called program does not use data items from the calling program, you can leave out the USING phrase. If data must be passed, you must place the USING phrase in two places:

- The CALL statement of the calling program
- The Procedure Division header of the called program.

You must make the identifiers specified in the USING phrase of the Procedure Division header data items defined in the Linkage Section of the called program. You can define the identifiers specified in the calling program in the File, Working-Storage, or Linkage Section.

You must make identifiers level-01 or -77 items. You can qualify them.

IBM Extension

You can use level numbers other than 01 or 77 for the data names in the USING phrase of the CALL statement in the calling program. You can index or subscript these data names.

End of IBM Extension

You must make the data names specified in the USING phrase of the Procedure Division header data items defined in the Linkage Section of the called program with a level number 01 or 77.

The number of identifiers you use in the USING phrase of the CALL statement must equal the number of data names you use in the USING phrase of the Procedure Division header. You can use no more than 15 identifiers or data names.

Note: Unpredictable results may occur if the number of identifiers in the Procedure Division Header does not match the number of identifiers in the CALL statement, and if the data descriptions of the identifiers and data names do not correspond by position.

You do not need to make the names of identifiers and data names correspond, but you must not use the same identifier or data name more than once in the same USING phrase.

EXIT PROGRAM Statement

The EXIT PROGRAM statement specifies the logical end of a called program.

Format

```
paragraph-name. EXIT PROGRAM
```

You must precede the EXIT statement with a paragraph name, and you must make it the only statement in the paragraph.

If control reaches an EXIT PROGRAM statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement. If control reaches an EXIT PROGRAM statement and no CALL statement is active, control passes through the exit point to the first sentence of the next paragraph. The EXIT PROGRAM statement is required to exit from the called program.

STOP RUN Statement

The STOP RUN statement is discussed under the *STOP Statement* in Chapter 11.

Segmentation Considerations

You can place a CALL statement anywhere within a segmented program; the compiler ensures that the proper logic flow is maintained. Therefore, if you place a CALL statement in an independent segment, that segment is made available in its last-used state when control returns from the called program.

Subprogram Linkage Feature Examples

The CALL statement is illustrated in the following program example:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  CALLSTAT.
      .
      .
      .
DATA DIVISION.
      .
      .
      .
WORKING-STORAGE SECTION.
01    RECORD-2 PIC X.
01    RECORD-1.
      05    SALARY          PICTURE S9(5)V99.
      05    RATE            PICTURE S9V99.
      05    HOURS           PICTURE S99V9.
      .
      .
      .
PROCEDURE DIVISION.
      .
      .
      .
      CALL 'SUBPRG' USING RECORD-1, RECORD-2.
      .
      .
      .
      STOP RUN.
```

The following called subprogram is associated with the preceding calling program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SUBPRG.
      .
      .
      .
DATA DIVISION
      .
      .
      .
LINKAGE SECTION.
01    PAYREC.
      10    PAY                    PICTURE S9(5)V99.
      10    HOURLY-RATE            PICTURE S9V99.
      10    HOURS                  PICTURE S99V9.
01    CODE PIC X.
      .
      .
      .
PROCEDURE DIVISION USING PAYREC, CODE.
      .
      .
      .
      EXIT-SUBPRG-PARAGRAPH.
      EXIT PROGRAM.
      .
      .
      .
```

Processing begins in the calling program, CALLSTAT. When the first CALL statement is performed, control transfers to the first statement of the Procedure Division in SUBPRG, which is the called program.

When SUBPRG receives control, the values within RECORD-1 are made available to SUBPRG; however, in SUBPRG they are referred to as PAYREC. The data items within PAYREC and CODE contain the same number of characters as RECORD-1 and RECORD-2. When processing within SUBPRG reaches the EXIT PROGRAM statement, control returns to the calling program.

Considerations When Using Inter-Program Communication

Inter-Program Communication lets you maintain a single version of a common processing routine. You can integrate changes made to this routine throughout your applications by recompiling this common processing routine and then recompiling any or all modules that call this routine.

Because a COBOL program can call other language programs as well as another COBOL program, you must create a program save area for each COBOL program to maintain pertinent data for each program. Therefore, you could notice a sizable increase in the amount of overhead space that your program uses if the number of subprograms being used is significant.

Some of the information in this save area is used to maintain the program register values, as well as supply parameter data to a given subprogram.

Data items you define in the Linkage Section of a program indicate that the particular data is used by a calling or called program. Since COBOL routines pass parameters by sending the actual address of the parameters, any reference to an item in the Linkage Section will have to contain run time logic to resolve the actual address of the data item. If you use numerous references in a subprogram to an item defined in the Linkage Section, much more space is needed to run the program because of the logic needed to resolve these references.

One way to avoid this problem is to move the desired Linkage Section item into another similarly defined item in the Working Storage Section, at the start of the subprogram. Then, before returning to the calling program, you can move the Working Storage item back to the Linkage Section item.

Using Ideographic Characters

Chapter 14. Using Ideographic Characters

This chapter describes the System/36 ideographic support in COBOL. For COBOL to successfully process ideographic data, your system must have the ideographic version of the SSP, and ideographic-capable input and output devices.

An ideographic character is a pictogram or graphic that requires two bytes of storage, whereas an alphanumeric character requires only one byte of storage.

With ideographic support, COBOL can process IBM-supplied or user-defined character sets. The IBM-supplied ideographic character set consists of 3226 Kanji characters, and 481 additional characters.

In general, COBOL handles ideographic characters in the same way it handles all alphanumeric data. Therefore, it is up to you to know (or have the COBOL program check) which data items contain ideographic characters, and to make sure the program receives and processes all ideographic data correctly.

GRAPHIC Option

For the COBOL program to recognize the use of ideographic characters, specify the GRAPHIC keyword on the PROCESS statement. The GRAPHIC compile-time option indicates to the COBOL compiler that ideographic literals can be present in the program.

If you do *not* specify the GRAPHIC option, ideographic literals are *not* recognized by the COBOL compiler. See Chapter 4 for more information about the PROCESS statement.

Rules for Ideographic Literals

Ideographic literals follow the same rules as alphanumeric literals, with the following additional requirements:

- A shift-out (S/O) control character (hex "0E") must immediately follow the opening quotes. The S/O character indicates the start of a string of ideographic characters.
- The ideographic literal must end with a shift-in (S/I) control character (hex "0F") followed immediately by quotes. The S/I character indicates the end of a string of ideographic characters.

Note: Although this discussion uses the term *quotes* to describe the delimiters for literals, the characters used as delimiters can vary depending on the option you specify on the PROCESS statement. If you specify the APOST option, apostrophes (') are used as delimiters. Otherwise, quotation marks (") are used as delimiters. In this discussion, *quotes* refers to either apostrophes or quotation marks. The delimiter used does not affect the rules for ideographic literals. For more information about the APOST compile-time option, see the description of the PROCESS statement in Chapter 4.

You can use any ideographic character in an ideographic literal. Each ideographic character has a two-byte hexadecimal representation. (An ideographic blank also occupies two bytes.) An ideographic literal should consist of only ideographic characters. Mixing ideographic characters and standard alphanumeric characters in the same literal causes COBOL to handle the entire literal as an alphanumeric literal.

Because each ideographic character occupies two bytes of storage, define any field that can contain ideographic data as having *an even number of bytes*. Because an ideographic literal must contain both the S/O and S/I control characters, the minimum length of an ideographic literal is two bytes.

The maximum length of an ideographic literal is 120 bytes, including the S/O and S/I control characters. Since each ideographic character occupies two bytes, a maximum of 59 ideographic characters can be coded in one COBOL ideographic literal.

Compiler Checking of Ideographic Literals

When you specify the GRAPHIC option on the PROCESS statement, and the compiler finds a literal that begins with quotes immediately followed by the S/O control character, the compiler checks for a valid ideographic literal. This is done by scanning two bytes at a time.

The following conditions cause a literal to be diagnosed as an *invalid* ideographic literal:

- An odd number of bytes are found between the S/O and S/I control characters.
- The terminating S/I control character is not immediately followed by quotes.
- The ideographic literal takes up more than one line and does *not* follow the rules for continuation of ideographic literals. (See *Continuing Ideographic Literals on a New Line,* below).

If a literal is found to be an invalid ideographic literal, the compiler processes the literal as a standard alphanumeric literal.

Examples of Ideographic Literals

Figure 14-1 shows an example of ideographic literals:

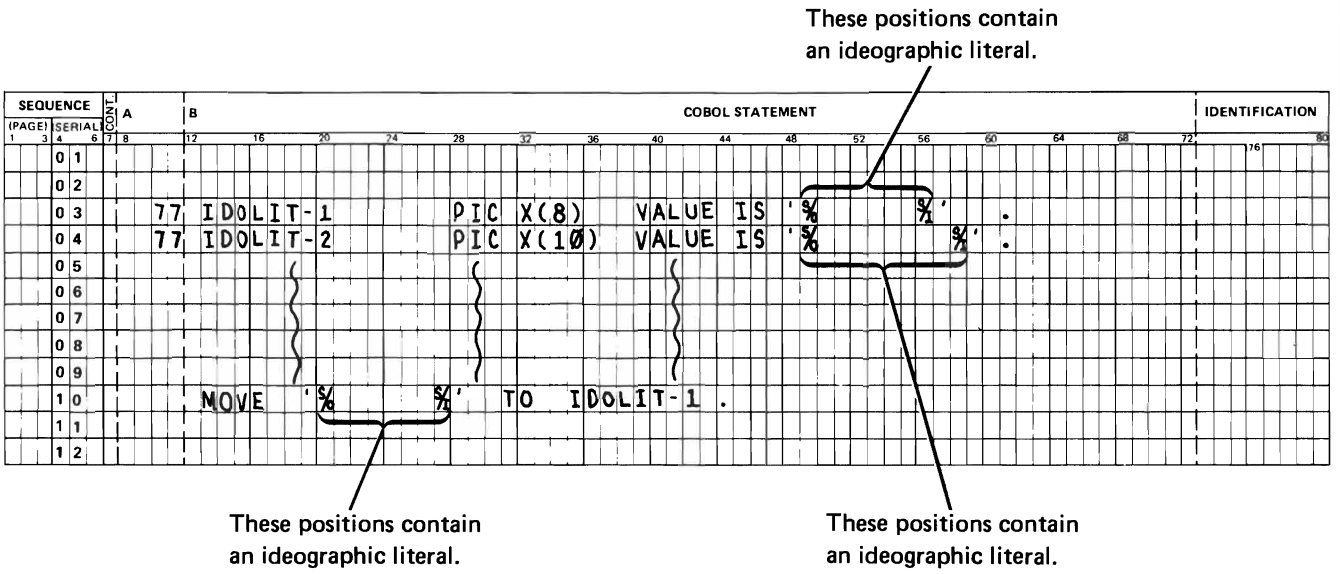


Figure 14-1. Example of Ideographic Literals

Continuing Ideographic Literals on a New Line

To continue an ideographic literal on another line of source code, do *all* of the following:

- Place a S/I control character in either column 71 or column 72 of the continued line. If the S/I is in column 71, column 72 must contain a blank.
- Place a hyphen (-) in column 7 (the continuation area) of the next line.
- Place quotes immediately followed by a S/O control character and the rest of the literal in Area B of the continuation line(s).

The S/I control character, quotes, and S/O control character that indicate a continuation are *not* counted in the length of the ideographic literal; the initial S/O and final S/I control characters *are* counted. Figure 14-2 shows an example of how to continue ideographic literals:

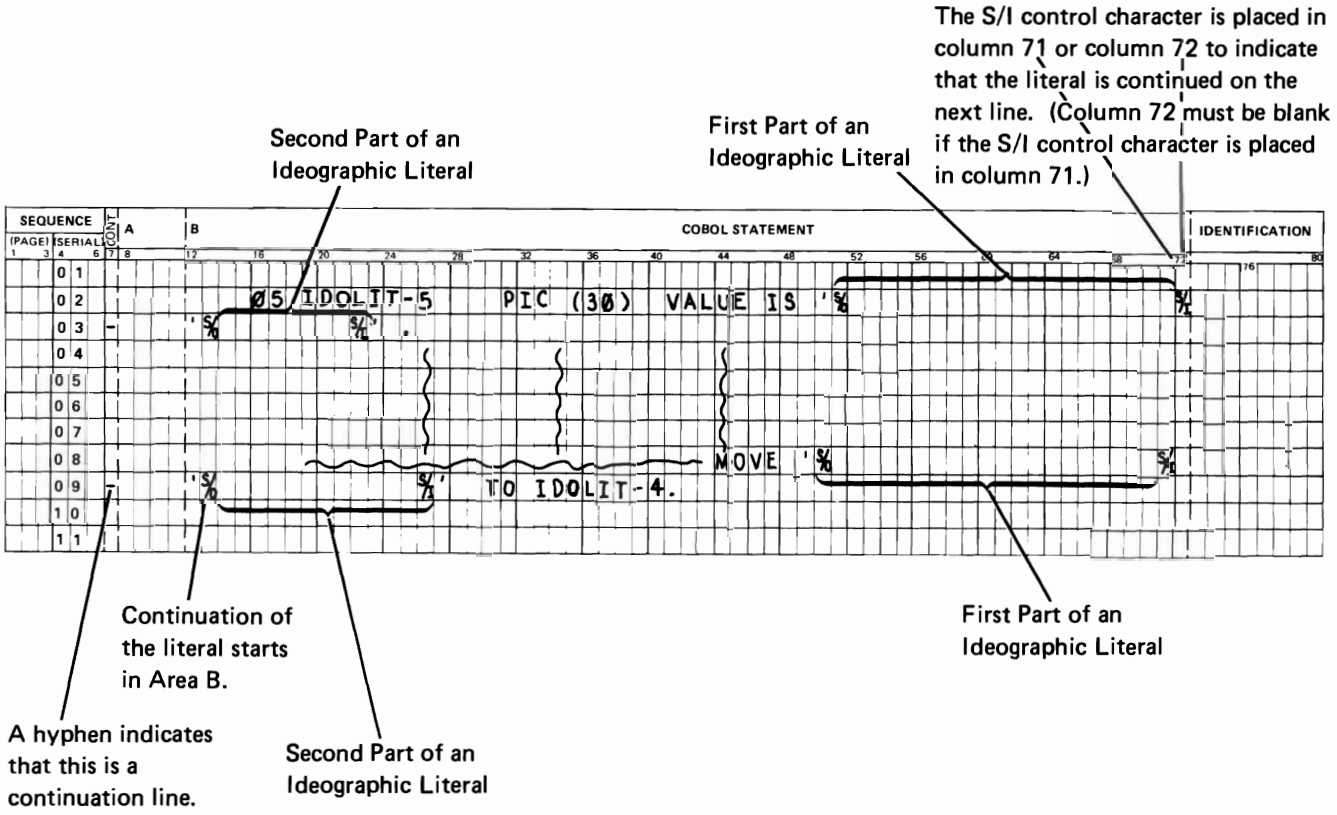


Figure 14-2. Example of continuing Ideographic Literals

Testing for Ideographic Support

To test whether ideographic support is available to your program, you can use the ATTRIBUTE-DATA mnemonic-name with the ACCEPT statement to access the attribute record. See *SPECIAL-NAMES Paragraph* in Chapter 7 for a description of the ATTRIBUTE-DATA mnemonic-name and for an example of using the ACCEPT statement.

The following values are returned in the attribute record for a display station:

Byte 9	Display type A - Alphanumeric or Katakana I - Ideographic
Byte 10	Keyboard Type A - Alphanumeric or Katakana I - Ideographic
Byte 11	Sign-on mode A - Alphanumeric or Katakana I - Ideographic
Byte 12	Work station mode N - Not 132-character capable. 1 - 132-character capable but currently in 80-character mode. 2 - 132-character capable and currently in 132-character mode.
Bytes 13 - 16	Reserved

Subroutines That Handle Ideographic Data

COBOL provides two subroutines to handle ideographic data. These subroutines are necessary to move ideographic data to a field of different length, and to insert or remove the S/O and S/I control characters.

System/36 requires that ideographic data be enclosed in the S/O and S/I control characters. Because you might need to send ideographic data to, or receive ideographic data from another system that does not require the S/O and S/I control characters, COBOL provides two subroutines to aid in that process. They are:

- CBINST -- This subroutine moves ideographic data and inserts the S/O and S/I control characters.
- CBREMV -- This subroutine moves ideographic data and removes the S/O and S/I control characters.

Both subroutines have the same calling sequence:

```
CALL 'CBINST' USING data-name-1, data-name-2, data-name-3,
                    data-name-4, data-name-5.
```

```
CALL 'CBREMV' USING data-name-1, data-name-2, data-name-3,
                    data-name-4, data-name-5.
```

The data-names are elementary items with the following definitions:

data-name	Definition	Purpose
data-name-1	PIC X( )...	Sending field
data-name-2	PIC X( )...	Receiving field
data-name-3	PIC X(1)...	Return code
data-name-4	PIC 9(3) USAGE IS COMP	Length of sending field
data-name-5	PIC 9(3) USAGE IS COMP	Length of receiving field

The maximum length of the sending and receiving fields is 256 bytes.

Move Ideographic Data and Insert Control Characters -- CBINST

CBINST is a move and edit subroutine that moves the contents of one field into another field. If the S/O and S/I control characters are not found in the first and last positions of the field to be moved, CBINST inserts them into the field when it is moved.

If you want the receiving field to contain all the data that is in the sending field, you must specify a receiving field length that is two positions longer than the length of the sending field. The two extra positions are to hold the S/O and S/I control characters. If you specify a receiving field that is longer than the sending field plus two, the data is padded on the right when it is moved. If the receiving field is specified either longer or shorter than the sending field plus two positions, the S/I control character is still placed in the rightmost position.

Subroutine CBINST produces return codes that show the status of the move operation. Figure 14-3 gives those return codes and their meanings:

Return Code	Explanation
0	Move executed; no errors.
1	Move executed; padding occurred to the left of the S/I control character.
2	Move executed; data truncated to the left of the S/I control character.
3	Move executed; S/O and S/I control characters already present.
4	Move not executed. Either odd field length found, length of zero found, length greater than 256 bytes, or invalid character found in field length.

Figure 14-3. Return codes for subroutine CBINST

If more than one return code can be issued, only the return code with the highest value is issued.

Move Ideographic Data and Remove Control Characters -- CBREMV

CBREMV is a move and edit subroutine that moves the contents of one field to another field. If the S/O and S/I control characters are found as the first and last characters in the field, CBREMV removes them.

If you want the receiving field to contain all the data that was present in the sending field, you must specify a receiving field length that is two positions less than the length of the sending field. This allows two positions for each ideographic character, while removing the S/O and S/I control characters (and the two positions they occupied). If you specify a receiving field longer than the sending field minus two positions, all the data from the sending field is moved and the receiving field is padded on the right with blanks. If the receiving field is shorter than the sending field minus two positions, the data being moved is truncated on the right.

Subroutine CBREMV produces return codes that show the status of the move operation. Figure 14-4 gives those return codes and their meanings:

Return Code	Explanation
0	Move executed; no errors.
1	Move executed; padding occurred on the right.
2	Move executed; data truncated on the right control character.
3	Move executed; S/O and S/I control characters not found in sending field.
4	Move not executed. Either odd field length found, length of zero found, length greater than 256 bytes, or invalid character found in field length.

Figure 14-4. Return codes for subroutine CBREMV

If more than one return code can be issued, only the return code with the highest value is issued.

Problem Determination

Chapter 15. COBOL Problem Determination

If a problem occurs while you are using COBOL, the cause of the problem may not be obvious. An error in your application or in system operation could have caused the problem. The problem determination procedure in this chapter can help you solve or circumvent the problem. If you need more information, refer to the following publications before contacting your service representative:

- *IBM System/36 System Problem Determination - 5360* (SC21-7919) if you use a System/36 System Unit 5360
- *IBM System/36 System Problem Determination - 5362* (SC21-9063) if you use a System/36 System Unit 5362
- Chapter 13 in the *Operating Your Computer - 5364* (SC21-9085) if you use a System/36 Unit 5364.

How to Use this Procedure

The Problem Determination procedure is arranged in a sequence of questions that you can answer with a *Yes* or *No*. Based on your answer, you are directed to another question or to a recommendation for action.

Start at the beginning of the procedure and follow the question-and-answer sequence, answering each question to which you are directed based on your previous answer. If the problem is a condition that requires more detailed procedures, you are referred to those procedures.

Identifying COBOL Problems

When a COBOL problem occurs, you can use the following series of questions to identify its possible cause:

1. **Have changes been made to the user program since the last time it ran successfully?**

   No **Yes**

   Read on, but consider what has been changed. For example, have operating procedures changed, are new device files being used, or have program changes been applied recently? A good starting point for problem determination is a changed item.

2. **Are you having a nonprogramming problem, such as spooled output that is not produced or a device that is not working?**

   No **Yes**

   You probably have a system problem. Call your system operator and have the operator use the appropriate procedure in the the appropriate manual referred to at the beginning of this chapter.

3. **Have all IBM PTFs (Program Temporary Fixes) that apply to the current release of COBOL been installed? (Check with your system operator)**

   Yes **No**

   Install the program changes you have received that have not yet been applied and run the program again.

4. **Are you using the current release of SSP?**

   Yes **No**

   Install the current release of SSP.

5. **Have all IBM PTFs you have received that apply to the current release of SSP been installed?**

   Yes **No**

   Install the program changes you have received that have not yet been applied.

6. **Have any non-IBM changes been made to COBOL or to SSP?**

   No **Yes**

   If COBOL has been changed, install its current release and program changes (PTFs), and run the program again. If SSP has been changed, install its current release and program changes (PTFs).

7. **Are you using the current release of COBOL? The release number is printed on the first line of the source listing for any COBOL program listed using SOURCE.**

   Yes **No**
   ↓ Install the current release of COBOL and compile or run the program again.

8. **Is the running time of the program much greater than you expect?**

   No **Yes**
   ↓ The program may be in a loop. Use the ATTN key to interrupt the program. Use TRACE and DEBUG statements to find out whether the program is in a loop. If a loop is found, cancel the program. Correct the problems in the program. Run the program again.

   Note: For further information about loops see ***Chapter 6 - Debugging Your Program***.

9. **Are no I/O operations taking place but you expect them to occur?**

   No **Yes**
   ↓ Check the status codes to see which one applies to your program. Check your program. Correct any errors. Run your program again.

10. **Did you encounter an abnormal program end while executing your program?**

    No **Yes**
    ↓ Check if:

    - A field you are using as a subscript or an index contains an invalid value, that is, a value less than 1 or greater than the OCCURS integer for the table. Correct the problems in the program. Run the program again.
    - The arguments passed in a CALL statement do not match those in the Procedure Division header in number, position, and description. Correct the problems in the program. Run the program again.
    - An attempt has been made to return from the root or first segment in an overlay program to an independent segment that has been overlaid. Check your program for such a problem. Correct the problems in the program. Run the program again.

    Note: For further information about Abnormal Program End see ***Chapter 6 - Debugging Your Program***.

11. **Does your program execute, but you receive unexpected results?**

No **Yes**

- A field you are using as a subscript or an index contains an invalid value, that is, a value less than 1 or greater than the OCCURS integer for the table. Check your subscripts or indexes to see that they are not out of range. Check all E, C, and W messages. Make sure that the object is from the last compile in case the output library was changed or the load module was not stored. Correct the problems in the program. Recompile and run the program again.

- The arguments passed in a CALL USING statement do not match those in the Procedure Division header in number, position, and description. Check to see that the parameters match in the CALL USING. Check all E, C, and W messages. Make sure that the object is from the last compile in case the output library was changed or the load module was not stored. Correct the problems in the program. Recompile and run the program again.

Note: For further information about Abnormal Program End see ***Chapter 6 - Debugging Your Program***.

12. **Does the SOURCE printout show the correct program?**

Yes **No**

Check if the program was not replaced the last time changes were made. Compare the SOURCE printouts of the program to a SOURCE printout made before the last changes were entered. Determine whether they match. Enter the changes again.

13. **Did you receive a compiler listing at compile time?**

Yes **No**

Check if:

- You chose NOPRINT as the Listing output option on the COBOLC procedure. Change CRT or NOPRINT to PRINT as the Listing output option on the COBOLC procedure. Recompile your program using the COBOLC procedure.
- You coded NOSOURCE on the *PROCESS statement. Use SEU/DSU to change NOSOURCE to SOURCE on the *PROCESS statement or choose SOURCE as the Override source print option of the COBOL or COBOLC procedure. Recompile your program using the COBOL or COBOLC procedure.
- You chose NOSOURCE as the Override source print option on the COBOL, COBOLONL or COBOLC procedure. Change NOSOURCE to SOURCE as the Override source print option of the COBOL, COBOLONL or COBOLC procedure. Recompile your program using the COBOL or COBOLC procedure.
- Your display station is configured to the wrong printer. Use the SSP procedure DISPLAY STATUS to see how the system is configured. Reconfigure the display station to the correct printer.

14. **Was a Load Module produced at compile time that cannot be found?**

No **Yes**

Look at the prolog on the compiler listing to see if the name of the output library you are looking for is the same as the Output library name you specified on the COBOL, COBOLONL or COBOLC procedure. Look for the load module in the library you specified in the COBOL or COBOLC procedure.

15. **Was a Load Module (Executable Program) produced at compile time?**

Yes **No**

Check if:

- You coded NOLINK on the *PROCESS statement. Use DSU/SEU to change NOLINK to LINK on the *PROCESS statement or choose LINK as the Create executable option on the COBOL or COBOLC procedure. Recompile your program using the COBOL or COBOLC procedure.
- You chose NOLINK as the Create executable module option on the COBOL or COBOLC procedure. Change NOLINK to LINK as the Create executable module option on the COBOL or COBOLC procedure. Recompile your program using the COBOL or COBOLC procedure.

16. **Was an Object Module produced at compile time that cannot be found?**

No **Yes**

Look at the prolog on the compiler listing to see if the name of the output library you are looking for is the same as the Output library name you specified on the COBOL or COBOLC procedure. Look for the subroutine module in the library you specified in the COBOL or COBOLC procedure.

17. **Was an Object Module (non-executable program) produced at compile time?**

Yes **No**

Check if:

- You did not code OBJECT on the *PROCESS statement. Use SEU/DSU to change NOOBJECT to OBJECT on the *PROCESS statement or choose OBJECT as the override creation of non-executable module option of the COBOL or COBOLC procedure. Recompile your program using the COBOL or COBOLC procedure.

- You chose NOOBJECT as the Create Non-executable module option on the COBOL or COBOLC procedure. Change NOOBJECT to OBJECT as the Create Non-executable module option on the COBOL or COBOLC procedure. Recompile your program using the COBOL or COBOLC procedure.

18. **Was a diagnosed source member produced at compile time?**

Yes **No**

Check if:

- You chose NODSM as the Create diagnosed source member option on the COBOLC procedure. The prolog of the compiler listing tells you what option you chose. Change NODSM to DSM as the Create diagnosed source member option on the COBOLC procedure. Recompile your program using the COBOLC procedure.

- You received a compiler error that canceled the compilation. Check all E, C, and W messages. Make sure that the object is from the last compile in case the output library was changed or the load module was not stored. Correct the problems in the program. Recompile and run the program again.

If after using this procedure, you or your system operator have not solved the problem, consult the *System Problem Determination* manual for your system unit referred to at the beginning of this chapter before calling the service representative.

Contacting Your Service Representative

If you cannot solve a problem using the problem determination procedures listed in this chapter and in the appropriate *System Problem Determination* manual referred to at the beginning of this chapter, you may want to contact your service representative. Before contacting your service representative, prepare the following:

- For compile time problems:
  - A task dump at the time of the failure
  - Run the APAR procedure and include the entire history file
  - A diskette copy of the COBOL user source program
  - A diskette copy of the user procedure
  - A diskette copy of the user source copy members
  - A listing of the source compilation.
- For execution time problems, provide the above required information as well as:
  - A diskette copy of the user files
  - A diskette copy of the user display screens
  - A diskette copy of the user execution procedure
  - A diskette copy of the user subprograms
  - A diskette copy of the user load module.

The procedures for obtaining the above information are explained in the *System Problem Determination Guide*.

System-Dependent Considerations

Appendix A. System-Dependent Considerations

This appendix describes the various system-defined limits and flexibilities that apply to System/36 COBOL. You should consider these items when designing a COBOL program for System/36.

GENERAL CONSIDERATIONS

Library Name, Program Name, and Text Name

You must specify unique entries for the library name, the program name, and the COPY text name. Although these names can be a maximum of 30 characters long, they must meet the following restrictions:

- The library name entry must be unique within the first 8 characters.
- The program name entry must be unique within the first 6 characters.
- The COPY text name entry must be unique within the first 8 characters.

Source Statements

Your System/36 COBOL program can contain no more than 65,535 source statements.

PROGRAM STRUCTURE

An application is easier to code and to maintain when it is designed carefully. You can use structured programming and top-down design to help you create programs that are easier to code and to maintain. Programs created in this manner are also usually more efficient.

If you are not familiar with structured programming, many manuals are available that can help you with this technique, including the *IBM Structured Programming Textbook*, SR20-7149, the *IBM Structured Programming Workbook*, SR20-7150, or *Improved Programming Technologies, An Overview*, GC20-1850.

Following is a short description of one method of developing an application. This method is an example of *top-down design*. For a more detailed description of how to develop an application, see the *Concepts and Programmer's Guide*.

Source Program Library

If you do not specify the OF/IN option of the COPY statement, the library defaults to the LIBRARY option of the PROCESS statement (unless overridden by a COBOL procedure). If you do not specify the LIBRARY option, and it was not overridden, the default value is #LIBRARY.

User-Defined Words

Your program can contain no more than 32,767 user-defined words.

Files

In a COBOL program, you can define a maximum of 25 files using file description (FD) and sort-merge file description (SD) entries.

Disk Data Management

System/36 offers you the flexibility of defining and processing indexed and relative files as if they were defined as physical sequential files. You can also define and process sequential indexed files as if they were defined as relative files. For more information on file organization and access modes, see *File Processing Summary* in Chapter 9.

Indexed and Relative File Contents

Position 1 of indexed or relative files cannot contain hexadecimal FF (for NATIVE collating sequence, this corresponds to HIGH-VALUE). Binary fields (COMPUTATIONAL-4) should be avoided in position 1, because they could contain this value. The key for an indexed file cannot exceed 120 characters. The key for a relative file must be no longer than 7 bytes.

ENVIRONMENT DIVISION CONSIDERATIONS

ASSIGN Clause

The ASSIGN clause associates a file with an external medium. The assignment name has the following format for printer and disk files:

```
Device-Type-Name
```

Device-Type:

Device-Type	Use
PRINTER	printer files
DISK	disk files

Name: 1- to 8-character field specifying the external name by which the file is known to the system. This is the name that appears in the NAME field on the OCL FILE statement.

IBM Extension

ASSIGN Clause (DATABASE Files)

Device Type:	DATABASE	DISK files with IBM extensions

Note: The DATABASE device type is a superset of the DISK device type which allows additional IBM extensions such as noncontiguous key processing and READ FIRST, LAST or PRIOR.

ASSIGN Clause (Transaction files): Using the ASSIGN clause, you associate the TRANSACTION file with devices through the use of the assignment name.

Format:

```
Device-Type-Filename1[-Format-Type1], Filename2
[-Format-Type2]
```

Device-Type:

Device Type	Use
WORKSTATION	communications and display station files

Filename: The name of the SFGR load member containing screen formats or, the IDDU format file containing communications formats. **Filename** is not required if only system-defined special formats are used.

Format-Type:

Format-Type	Use
	filename is an SFGR load member
S	filename is an SFGR load member
C	filename is an IDDU format file

Assignment-name-3 is treated as a comment.

The value for each field, 'filename' and 'Device-Type', is the same as the values above.

Note: Only one IDDU format file containing communication formats and one SFGR load member containing screen formats can be specified in an ASSIGN clause.

End of IBM Extension

RESERVE Clause

This clause must specify a value of 1 or 2; at least 1 buffer is required for a file. If you omit this clause, 1 buffer is reserved. If you specify a value greater than 2, 2 buffers are reserved.

SAME RECORD AREA Clause

You can specify no more than 15 SAME RECORD AREA clauses in a COBOL program.

SAME AREA or SAME SORT-MERGE Clause

You can specify more than 15 SAME AREA clauses in a COBOL program.

OBJECT-COMPUTER MEMORY SIZE Clause

This clause must specify an integer from 1 through 65,536.

KEY Clause

The RELATIVE KEY data name can have a maximum length of 7 bytes; the RECORD KEY data name can have a maximum length of 120 bytes in total.

DATA DIVISION CONSIDERATIONS

BLOCK CONTAINS Clause

The maximum block size is 9999 characters.

RECORD CONTAINS Clause

The maximum record length is 4096 bytes.

LINAGE Clause

The maximum size of the logical page is 32,767 lines. The logical page size is the sum of the number of lines in the body, top margin, and bottom margin of the page.

OCCURS Clause

The literals in the OCCURS clause must have a value of 1 through 32,767.

Item Size

If no other restrictions apply, the maximum item size is 32,767.

Index and Subscript Literals

An index or subscript literal must have a value of 1 through 32,767.

PROCEDURE DIVISION CONSIDERATIONS

CALL Statement

Your program can call no more than 100 subroutines. Called subroutines can also reside in the overlay area of the program being run. All subroutines that you want to overlay must have a category number greater than 7 when link-edited. For a further description of overlay, refer to *Link Editing with Overlay* in Chapter 4.

Using Option: A maximum of 15 operands can be specified for the USING option.

COMPUTE Statement

The maximum size of each operand is 18 decimal digits. Division by 0 always results in a size error condition.

GO TO DEPENDING ON Statement

You can specify a maximum of 99 branch points in a GO TO DEPENDING ON statement.

INSPECT Statement

You can specify a maximum of 15 comparison operands (TALLYING/REPLACING) in an INSPECT statement.

SORT/MERGE Statement

You can specify a maximum of 12 KEYS and 8 input files in any SORT or MERGE statement.

STOP Statement

When you specify STOP literal and the literal is nonnumeric, the literal is limited to 120 characters.

UNSTRING Statement

You can specify a maximum of 15 delimiters in an UNSTRING statement, and each delimiter must be an alphanumeric data item.

TRANSACTION FILE CONSIDERATIONS

If your display station is attached to a TRANSACTION file, and a SYSLIST procedure was run or an OCL statement was performed that changed the SYSLIST device to CRT, unpredictable results can occur when low-volume data is sent to the display station by the COBOL program (through low-volume input or output statements, such as DISPLAY, EXHIBIT, or ACCEPT).

Display station contention and unpredictable results may also occur when using SYSLOG in a procedure prior to executing a program operating with one or more TRANSACTION files.

Representation of Hexadecimal Values

To represent a hexadecimal character in a COBOL program, you must be familiar with the internal representation of numbers.

For example, the DUP KEY of the display terminal is a hexadecimal 1C which has to be converted to a form that a COBOL program can use. First of all, a variable must be defined in the following way:

```
A           PICTURE 9  COMP-4.
```

The use of COMP-4 allows the storing of hexadecimal values. Second, the hexadecimal value '1C' must be changed to its corresponding decimal value. The hexadecimal-decimal conversion tables in the *IBM System/36 Functions Reference Manual* SC21-9436 provide the corresponding decimal value of 28.

This would mean that the following statement would cause '001C' to be stored:

```
01     B           PICTURE 9  COMP-4  VALUE IS 28.
```

Next, a REDEFINES clause eliminates errors that would occur from moving from one data type to another.

```
01     B           REDEFINES  A.
    05    FILLER      PICTURE X.
    05    DUP-KEY     PICTURE X.
```

At this point, DUP-KEY is equal to hexadecimal '1C' and can be used for data comparison in the procedure division of the program.

Special Purpose Subroutines

Appendix B. Special Purpose Subroutines

System/36 COBOL provides the following subroutines that allow you to use special features of the system:

- CBMICR, CBMICO, CBEMCR, and CBEMCO read document information, using the 1255 Magnetic Character Reader (MCR)
- CBFTOD provides the time of day in system units (8.192 milliseconds)
- CBSTOP interrogates the system shutdown status.

1255 Magnetic Character Reader (MCR) Interface Subroutines

The four special COBOL 1255 Magnetic Character Reader (MCR) interface subroutines let you access document information read by the 1255 MCR. The CBMICR and CBMICO subroutines provide a function equivalent to that found in 1255 MCR subroutine SUBR08. The CBEMCR and CBEMCO subroutines provide a function equivalent to that found in 1255 MCR subroutine SUBR25.

The subroutines provide two ways to process document information:

1. CBMICR and CBEMICO use system and stacker specifications to describe the job to be done by the 1255.
2. CBEMCR and CBEMCO use a device control language (DCL) program to describe the job to be done by the 1255 MCR. The subroutine's parameter list is the data management interface between the subroutine and the DCL program. The parameter list takes the place of the system and stacker specifications used in the SUBR08 COBOL program. The DCL program is a separate assembler-like program that runs in the attachment I/O controller for the 1255 MCR.

The subroutines provide both an open and a read function. A call to an *open* subroutine (CBMICO or CBEMCO) is required before you can read records from the 1255 MCR. When a call to a *read* subroutine (CBMICR or CBEMCR) returns an end-of-file condition, no more records can be read until a second *open* call has been executed.

Formats of the subroutine calls are as follows:

- CALL 'CBEMCO' USING data-name-1
- CALL 'CBMICO' USING data-name-1
- CALL 'CBEMCR' USING data-name-2
- CALL 'CBMICR' USING data-name-2.

For subroutines CBEMCO and CBMICO, data-name-1 must refer to a structure having the following format:

```
01  Data-name-1.
    05  Data-name-a   PICTURE 9.
    05  Data-name-b   PICTURE 9(4) USAGE COMP-4 VALUE integer-1.
    05  Data-name-c   PICTURE 9(4) USAGE COMP-4 VALUE integer-2.
    05  Data-name-d   PICTURE 9(3) USAGE COMP-4 VALUE integer-3.
    05  Data-name-e   PICTURE X(integer-2).
    05  Data-name-f.
    05  Data-name-g   PICTURE XX    VALUE HIGH-VALUE.
```

where:

- **Data-name-a** is the return code generated after each read operation. Return code values and meanings are:

  0 - Successful completion

  1 - End-of-file condition

  3 - Permanent error.

- **Data-name-b** is the length of the system and stacker specifications, or SUBR25 parameter list array (contained in data-name-f). Integer-1 must be equal to or be a multiple of the number 80.

- **Data-name-c** is the length of the input buffer (contained in data-name-e). Integer-2 must be eight bytes larger than the desired buffer size, to allow for System/36 boundary alignment. The buffer must be large enough to accommodate at least ten records.

- **Data-name-d** is the length of the input record. When the data structure is used with CBMICO, the length must be 55 bytes and is provided only for proper spacing of data.

- **Data-name-e** is the input buffer. It must include eight additional bytes for boundary alignment. The size must agree with the value coded in data-name-c.

- **Data-name-f** is the system and stacker specifications or SUBR25 parameter list array. The size of the array must match the value coded in data-name-b. For a discussion of the array contents, see the manual, *Using and Programming the 1255 Magnetic Character Reader*. Use the SUBR08 system and stacker specification format for CBMICO, and use the SUBR25 parameter list format for CBEMCO.

- **Data-name-g** is the delimiter used by the *open* subroutine to check the accuracy of the structure. If your program contains a variable number of stacker specifications, be sure to move this field to the position following the last specification and place the proper value in data-name-b before calling the *open* subroutine.

For subroutines CBEMCR and CBMICR, **data-name-2** is the logical record area into which the *read* subroutine places an input record each time the subroutine is called. The length must be at least 55 bytes for CBMICR and it must not be less than the value in data-name-d for CBEMCR. Figure B-1 shows the format of the standard 55-character input record for CBMICR. The input record format for CBEMCR is user-defined. Refer to the manual, *Using and Programming the 1255 Magnetic Character Reader,* for a detailed explanation of how to use the 1255 MCR. The manual contains in-depth explanations of the following:

- Subroutines SUBR08 and SUBR25

- System and stacker specifications

- The SUBR25 Parameter List and Device Control Language (DCL) program

- The input record format.

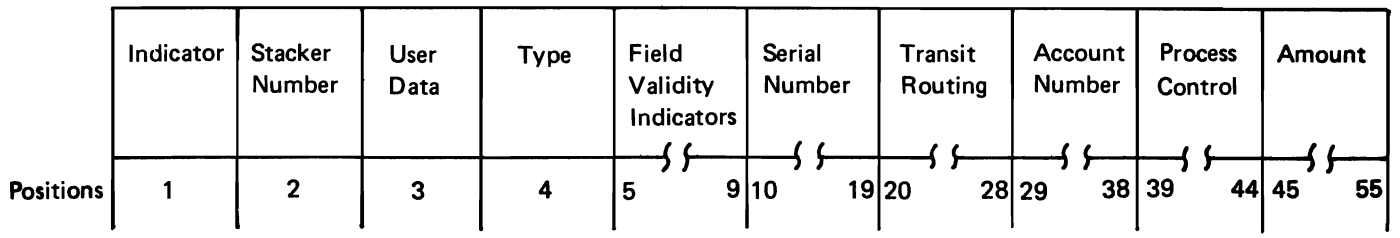


Figure B-1. Format of the input record

Enhanced Timer Subroutine

The CBFTOD subroutine allows you to get better timing resolution than with the ACCEPT TIME statement. With CBFTOD, you can obtain the time of day either in system units of 8.192 milliseconds, or in the normal "HHMMSS" format. CBFTOD also provides the system date.

This subroutine is called by the COBOL statement:

```
CALL 'CBFTOD' USING identifier-1, identifier-2 [identifier-3]
```

where:

- **Identifier-1** is the date, defined as PICTURE 9(6).
- **Identifier-2** is the time of day. Define identifier-2 as PICTURE 9(9) COMP-4 when getting the time in system units of 8.192 milliseconds. Otherwise, define identifier-2 as PICTURE 9(6) for the standard "HHMMSS" format.
- **Identifier-3**, when specified, indicates that the time of day is to be provided in 8.192-millisecond units. If you specify identifier-3, it *must* have a value of 1, and must be defined as PICTURE 9. If you do *not* specify identifier-3, the time of day is provided in the "HHMMSS" format.

If CBFTOD detects an error in the parameter list, the subroutine sets to ZERO the date returned in identifier-1. After calling this subroutine, you should check the date field to make sure a zero value was *not* returned.

The following errors can be detected by the CBFTOD subroutine:

- More than three parameters were passed.
- Identifier-3 was specified but was not equal to 1.

Shutdown Status Test Subroutine

The shutdown status test subroutine, CBSTOP, is a special System/36 feature. The CBSTOP subroutine is used to determine whether the system operator has requested system shutdown. This subroutine is called by the COBOL statement:

```
CALL 'CBSTOP' USING identifier
```

where identifier is a one-character numeric item defined in the Working-Storage or Linkage section with explicit or implicit usage of DISPLAY in the calling program. Upon return from CBSTOP, the identifier will contain one of the following values:

0 - Shutdown has not been requested

1 - Shutdown has been requested.

Language Summary and Comparison

Appendix C. Language Summary and Comparison

Assumptions for System/36 COBOL Language

1. The low-intermediate FIPS level of ANS 1974 COBOL is supported, except for restrictions noted under the *Level of Language Support* in Chapter 1. This level requires the following processing modules: 1NUC, 1TBL, 1SEQ, 1REL, 1SEG, 1LIB, 1DEB, 1IPC.

   Summary of the Four Levels of FIPS COBOL

LOW	L/I	H/I	HIGH	
1	1	2	2	NUC-Nucleus
1	1	2	2	TBL-Table Handling
1	1	2	2	SEQ-Sequential I/O
-	1	2	2	REL-Relative I/O
-	-	-	2	INX-Indexed I/O
-	-	1	2	SRT-Sort-Merge
-	-	-	-	RPW-Report Writer
-	1	1	2	SEG-Segmentation
-	1	1	2	LIB-Library
-	1	2	2	DEB-Debug
-	1	2	2	IPC-Inter-Program Communication
1	-	2	2	COM-Communication

   Legend:

   L/I = Low-intermediate
   H/I = High-intermediate
   - = Not included in the referenced level
   1 = The processing module must be implemented at ANS level 1
   2 = The processing module must be implemented at ANS level 2

2. A large selection of elements from higher-level ANS modules are provided, as well as existing and new IBM extensions. System/3 COBOL, System/34 COBOL, and System/36 COBOL support requirements are used to complete this selection. Elements supported appear under the column S/34 and S/36 in this appendix.

3. As required by FIPS, a mechanism is provided for flagging elements that are not in a given FIPS level.

4. Compiler options are those provided by System/3 COBOL and System/34 COBOL, as well as options for the following:

- Cross-reference listing
- Syntax-check only compile (no code generation)
- Identification of statements that do not adhere to FIPS-levels.

Summary of System/36 COBOL Language

Figures C-1 and C-2 explain the headings and the codes used in the summaries of System/36 COBOL processing modules on the following pages.

Column Headings	Explanation
ANS 1	1974 ANS COBOL standard, level 1 of those modules considered for inclusion in System/36 COBOL
ANS 2	1974 ANS COBOL standard, level 2 of those modules considered for inclusion in System/36 COBOL
S/3	System/3 COBOL and System/34 PRPQ COBOL language (ANSI 1968 Standard)
S/34	System/34 COBOL language
S/36	System/36 COBOL language

Figure C-1. Column Headings for Summaries of COBOL Processing Modules

Code	Explanation
X	Element is allowed (additional notes may apply).
-	Element is not allowed.
a,b,c, ...	Indicated parenthetical note follows.
*	An asterisk beside any entry in this column indicates that the element applies only for System/34. System/36 will treat it as comments.
1	Allowed, but treated as comments.
2	Allowed, but with restrictions.
3	System/3 compiler gives a diagnostic, but gives proper result.
4	System/3 compiler supports the same function, but via different syntax.
5	ANS 1974 standard indicates this is partly implementer defined, or it is dependent on specific hardware components.
6	Allowed, but IBM-defined limits exist (in accordance with code 5).
7	System/3 compiler diagnoses this as an error.
8	System/3 compiler does not allow this to be omitted; if it is omitted, System/3 compiler gives a diagnostic but recovers according to 1974 ANS COBOL rules.
9	Supported by System/36 only.

Figure C-2. Explanation of Codes for Summaries of COBOL Processing Modules

Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	S/36	Elements
					LANGUAGE CONCEPTS
					Character Set
X	X	X	X	X	• Characters used for words: 0 through 9 and A through Z - (hyphen)
					• Characters used in punctuation:
X	X	X	X	X	Space(), equal sign(=), and quote(")
–	X	X	X	X	comma and semicolon
–	–	X	X	X	apostrophe instead of quote (**IBM Extension**)
					• Characters used in editing:
X	X	X	X	X	B + - . , Z * $
X	X	X	X	X	0 CR DB
X	X	–	X	X	/
–	X	X	X	X	• Characters used in arithmetic operations: + - * / **
–	X	X	X	X	• Characters used in relation conditions: = > <
					Separators
X	X	X	X	X	• Quote("), period(.), and space()
–	X	X	X	X	• Semicolon and comma
					Character strings
					• COBOL words
X	X	X	X	X	– Words up to 30 characters are supported
					– User-defined words

Figure C-3 (Part 1 of 13). Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	S/36	Elements
X	X	X	X	X	• Data name
–	X	X	X	X	– Data name need not begin with an alpha character
X	X	X	X	X	– Level number
X	X	X	X	X	– Mnemonic name
X	X	X	X	X	– Paragraph name
X	X	X	X	X	– Program name
X	X	X	X	X	– Routine name
X	X	X	X	X	– Section name
–	X	X	X	X	– Condition name
					– System names
X	X	X	X	X	• Computer name
X	X	X	X	X	• Implementer name
X	X	X	X	X	• Language name
					– Reserved words
X	X	X	X	X	• Key words
X	X	X	X	X	• Optional words
					• Figurative constants:
X	X	X	X	X	ZERO, SPACE
–	X	X	X	X	ZEROS, ZEROES, SPACES
X	X	X	X	X	HIGH-VALUE, LOW-VALUE, QUOTE
–	X	X	X	X	HIGH-VALUES, LOW-VALUES, QUOTES, ALL literal

Figure C-3 (Part 2 of 13). Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	X	X	X	X	• Special-character words: Arithmetic operators and relational operators
					• Connectives
–	X	X	X	X	– Qualifier connectives: OF, IN
–	X	X	X	X	– Series connectives: , (separator comma) ; (separator semicolon)
–	X	X	X	X	– Logical connectives: AND, OR, AND NOT, OR NOT
–	–	X	–	–	• Special register: TALLY (**ANS 1968**)
					• Literals
X	X	X	X	X	– Numeric literals: 1 to 18 digits
X	X	X	X	X	– Nonnumeric literals: 1 to 120 characters
X	X	X	X	X	• PICTURE character strings
X	X	X	X	X	• Comment entries
					Qualification Rules
X	X	X	X	X	• Unqualified references to unique names
–	X	X	X	X	• Qualified references to nonunique names
–	X	X	X	X	– Data names, paragraph names, condition names
–	X	–	X	X	– Text names
					Reference Format
X	X	X	X	X	• Sequence number
					• Continuation of lines

Figure C-3 (Part 3 of 13). Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	S/36	Elements
X	X	X	X	X	– Nonnumeric literals
–	X	X	X	X	– Words and numeric literals
					• Comment lines
X	X	X	X	X	– Asterisk (*) comment line
X	X	X	X	X	– Stroke (/) comment line
					IDENTIFICATION DIVISION
X	X	X	X	X	• PROGRAM-ID paragraph
X	X	X	X	X	• AUTHOR paragraph
X	X	X	X	X	• INSTALLATION paragraph
X	X	X	X	X	• DATE-WRITTEN paragraph
–	X	–	X	X	• DATE-COMPILED paragraph
X	X	X	X	X	• SECURITY paragraph
–	–	X	–	–	• REMARKS paragraph (** ANS 1968 **)
					ENVIRONMENT DIVISION
					Configuration Section
X	X	X	X	X	• SOURCE-COMPUTER paragraph
X	X	X	X	X	• OBJECT-COMPUTER paragraph
X,5	X,5	X	X	X	– Computer name
X,5	X,5	X	X	X	– MEMORY SIZE clause
X	X	–	X	X	– PROGRAM COLLATING SEQUENCE clause
X	X	X	X	X	• SPECIAL-NAMES paragraph
X	X	–	X	X	– Alphabet-name clause
X	X	–	X	X	• STANDARD-1 option
X	X	–	X	X	• NATIVE option

Figure C-3 (Part 4 of 13). Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	S/36	Elements
X,5	X,5	–	X	X	• Implementer-name phrase
–	X	–	X	X	• Literal option
X	X	X	X	X	– CURRENCY SIGN clause
X	X	X	X	X	– DECIMAL-POINT clause
X,5	X,5	X	X	X	– Implementer-name IS mnemonic-name
–	–	X	X	X	• IBM allows C01, CSP
–	–	X	–	–	• IBM allows CONSOLE
–	–	–	X	X	• IBM allows REQUESTOR, SYSTEM-CONSOLE
–	–	X	X	X	• IBM allows UPSI-0 through UPSI-7
X	X	X	X	X	• ON STATUS IS condition-name
X	X	X	X	X	• OFF-STATUS IS condition-name
–	–	–	X	X	• IBM allows LOCAL-DATA, ATTRIBUTE-DATA, SYSTEM-SHUTDOWN
X	X	X	X	X	– Implementer-name series
					DATA DIVISION
					Working-Storage Section
X	X	X	X	X	• Data description entry
X	X	X	X	X	• BLANK WHEN ZERO clause
X	X	X	X	X	• Data-name or FILLER clause
X	X	X	X	X	• JUSTIFIED (or JUST) clause
X	X	X	X	X	• Level number
					– Valid (logical) values

Figure C-3 (Part 5 of 13). Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	S/36	Elements
X	X	X	X	X	• 01 through 10
–	X	X	X	X	• 11 through 49
–	X	–	X	X	• 66 (RENAMES)
X	X	X	X	X	• 77
–	X	X	X	X	• 88
					– Valid (physical) appearance
X	X	X	X	X	• Two digits supported (01, 02, ...)
–	X	X	X	X	• One-digit abbreviation supported (1, 2,..., 9)
X	X	X	X	X	• PICTURE (or PIC) clause
X	X	X	X	X	– Character string may contain 30 characters
X	X	X	X	X	– Data characters: X9
X	X	X	X	X	– Data character: A
X	X	X	X	X	– Operational symbol: S
X	X	X	X	X	– Operational symbol: V
X	X	X	X	X	– Operational symbol: P
					– Fixed insertion characters:
X	X	X	X	X	B + - . , $
X	X	X	X	X	O CR DB
X	X	–	X	X	/
X	X	X	X	X	– Replacement or floating characters $ + - Z *
X	X	X	X	X	– Currency sign substitution
X	X	X	X	X	– Decimal point substitution
X	X	X	X	X	• REDEFINES Clause
–	X	X	X	X	– May be nested

Figure C-3 (Part 6 of 13). Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	X	–	X	X	• RENAMES clause
X	X	X	X	X	• SIGN clause
X,5	X,5	X,1	X,1	X,1	• SYNCHRONIZED (or SYNC) clause
X	X	X	X	X	• USAGE clause
X	X	X	X	X	– DISPLAY
X,5	X,5	X	X	X	– COMPUTATIONAL (or COMP)
–	–	X	X	X	– COMPUTATIONAL-3 (or COMP-3) (**IBM/Codaysl**)
–	–	X	X	X	– COMPUTATIONAL-4 (or COMP-4) (**IBM/Codaysl**)
X	X	X	X	X	• VALUE clause
X	X	X	X	X	– Literal
–	X	–	X	X	– Literal series
–	X	–	X	X	– Literal-1 THRU literal-2
–	X	–	X	X	– Literal range series
					PROCEDURE DIVISION
–	X,5	X,a	X,a	X,a	Arithmetic Expressions (a = exponent identifier/literal must be positive integral value)
					Conditional Expressions
X	X	X	X	X	• Simple condition
X	X	X	X	X	– Relation condition
X	X	X	X	X	• Relational operators
X	X	X	X	X	– [NOT] GREATER THAN

Figure C-3 (Part 7 of 13). Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	X	X	X	X	– [NOT] >
X	X	X	X	X	– [NOT] LESS THAN
–	X	X	X	X	– [NOT] <
X	X	X	X	X	– [NOT] EQUAL TO
X	X	X	X	X	– [NOT] =
X	X	X	X	X	• Comparison of numeric operands
X	X	X	X	X	• Comparison of nonnumeric operands
–	X	X	X	X	– Operands of unequal size are permitted
X	X	X	X	X	– Class condition
–	X	X	X	X	– Condition-name condition
–	X	X	X	X	– Sign condition
X	X	X	X	X	– Switch-status condition
–	X	X	X	X	• Complex condition
–	X	X	X	X	– Logical Operators AND, OR, NOT
–	X	X	X	X	– Negated simple condition
–	X	X	X	X	– Combined and negated combined condition
–		–	X	X	– Abbreviated combined relation condition
X	X	X	X	X	Arithmetic Statements
X	X	X	X	X	• Arithmetic operands limited to 18 digits
–	X	–	X	X	• Multiple results in arithmetic statements
X	X	X	X	X	ACCEPT Statement
X,5	X	X,6	X,6	X,6	• At least one transfer of data supported

Figure C-3 (Part 8 of 13). Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	X	X	X	X	• FROM mnemonic-name phrase
–	–	X	–	–	• FROM CONSOLE phrase (**IBM Extension**)
–	X	X	X	X	• FROM DATE phrase
–	X	–	X	X	• FROM DAY phrase
–	X	–	X	X	• FROM TIME phrase
X	X	X	X	X	ADD Statement
X	X	X	X	X	• Identifier/literal series
X	X	X	X	X	• TO identifier
–	X	–	X	X	• TO identifier series
X	X	X	X	X	• GIVING identifier
–	X	–	X	X	• GIVING identifier series
X	X	X	X	X	• ROUNDED phrase
X	X	X	X	X	• SIZE ERROR phrase
–	X	–	X	X	• CORRESPONDING phrase
X	X	X	X	X	ALTER Statement
X	X	X	X	X	• Procedure name
–	X	X	X	X	• Procedure-name series
–	X	X	X	X	COMPUTE Statement
–	X,5	X	X	X	• Arithmetic expression
–	X	–	X	X	• Identifier series
–	X	X	X	X	• ROUNDED phrase
–	X	X	X	X	• SIZE ERROR phrase
X	X	X	X	X	DISPLAY Statement
X,5	X	X,6	X,6	X,6	• At least one transfer of data supported
–	X	–	X	X	• Multiple transfers of data supported

Figure C-3 (Part 9 of 13). Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	S/36	Elements
X	X	X	X	X	• Identifier/literal
X	X	X	X	X	• Identifier/literal series
–	X	X	X	X	• UPON mnemonic-name phrase
–	–	X	–	–	• UPON CONSOLE phrase (**IBM Extension**)
X	X	X	X	X	DIVIDE Statement
X	X	X	X	X	• INTO identifier
–	X	–	X	X	• INTO identifier series
X	X	X	X	X	• By identifier/literal
X	X	X	X	X	• GIVING identifier
–	X	–	X	X	• GIVING identifier series
–	X	–	X	X	• REMAINDER phrase
X	X	X	X	X	• ROUNDED phrase
X	X	X	X	X	• SIZE ERROR phrase
X	X	X	X	X	ENTER Statement
X	X	X	X	X	EXIT Statement
X	X	X	X	X	GO TO Statement
–	X	X	X	X	• Procedure name can be omitted
X	X	X	X	X	• DEPENDING ON phrase
X	X	X	X	X	IF Statement
X	X	X	X	X	• Imperative statement can contain multiple verbs
–	X	X,3	X	X	• Not limited to imperative statements
–	X	X,3	X	X	• Nested statements
X	X	X	X	X	• ELSE phrase
X	X	X	X	X	• NEXT SENTENCE phrase
–	–	X	–	–	EXAMINE Statement (**ANS 1968**)

Figure C-3 (Part 10 of 13). Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	–	X	–	–	• Only single-character literals
–	–	X	–	–	• TALLYING phrase
–	–	X	–	–	– ALL/LEADING/UNTIL FIRST option
X	X	–	X	X	INSPECT Statement
X	X	–	X	X	• Single-character identifiers or literals
–	X	–	X	X	• Multiple-character identifiers or literals
X	X	–	X	X	• TALLYING phrase
X	X	–	X	X	– BEFORE/AFTER INITIAL option
X	X	–	X	X	• REPLACING phrase
X	X	–	X	X	• TALLYING and REPLACING phrases
–	X	–	X	X	• TALLYING and REPLACING series
X	X	X	X	X	MOVE Statement
X	X	X	X	X	• TO identifier
X	X	X	X	X	• Identifier series
–	X	–	X	X	• CORRESPONDING phrase
X	X	X	X	X	MULTIPLY Statement
X	X	X	X	X	• BY identifier
–	X	–	X	X	• BY identifier series
X	X	X	X	X	• GIVING identifier
–	X	–	X	X	• GIVING identifier series
X	X	X	X	X	• ROUNDED phrase
X	X	X	X	X	• SIZE ERROR phrase
–	–	X	–	–	NOTE Statement (**ANS 1968**)

Figure C-3 (Part 11 of 13). Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	S/36	Elements
X	X	X	X	X	PERFORM Statement
X	X	X	X	X	• Procedure name
X	X	X	X	X	• THRU phrase
X	X	X	X	X	• TIMES phrase
–	X	X	X	X	• UNTIL phrase
–	X	X,2	X	X	• VARYING phrase
X	X	X	X	X	STOP Statement
X	X	X	X	X	• Literal
X	X	X	X	X	• RUN
–	X	–	X	X	STRING Statement
–	X	–	X	X	• Identifier/literal series
–	X	–	X	X	• DELIMITED BY phrase
–	X	–	X	X	• POINTER phrase
–	X	–	X	X	• ON OVERFLOW phrase
X	X	X	X	X	SUBTRACT Statement
X	X	X	X	X	• Identifier/literal series
X	X	X	X	X	• FROM identifier
–	X	–	X	X	• FROM identifier series
X	X	X	X	X	• GIVING identifier
–	X	–	X	X	• GIVING identifier series
X	X	X	X	X	• ROUNDED phrase
X	X	X	X	X	• SIZE ERROR phrase
–	X	–	X	X	• CORRESPONDING phrase
–	X	–	X	X	UNSTRUNG Statement
–	X	–	X	X	• DELIMITED BY phrase
–	X	–	X	X	• INTO series

Figure C-3 (Part 12 of 13). Summary of Elements in the Nucleus

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	X	–	X	X	– DELIMITER phrase
–	X	–	X	X	– COUNT phrase
–	X	–	X	X	• POINTER Phrase
–	X	–	X	X	• TALLYING phrase
–	X	–	X	X	• ON OVERFLOW phrase

Figure C-3 (Part 13 of 13). Summary of Elements in the Nucleus

Summary of Elements in the Table Handling Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
					LANGUAGE CONCEPTS
					User-Defined Words
X	X	X	X	X	• Index name
					Subscripting
X	X	X	X	X	• 1 level supported
X	X	X	X	X	• 2 or 3 level supported
					Indexing
X	X	X	X	X	• 1, 2, or 3 levels supported
					DATA DIVISION
X	X	X	X	X	OCCURS clause
X	X	X	X	X	• Integer TIMES
–	X	–	X	X	• Integer-1 TO integer-2 DEPENDING on data-name
–	X	–	X	X	• ASCENDING/DESCENDING data-name
–	X	–	X	X	• Data-name series
–	X	–	X	X	• ASCENDING/DESCENDING series
X	X	X	X	X	• INDEXED BY index-name series
X,5	X,5	X	X	X	USAGE IS INDEX Clause
					PROCEDURE DIVISION
–	X	–	X	X	SEARCH Statement
–	X	–	X	X	• VARYING phase
–	X	–	X	X	• AT END phase
–	X	–	X	X	• WHEN phase

Figure C-4 (Part 1 of 2). Summary of Elements in the Table Handling Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	X	–	X	X	• WHEN phase series
–	X	–	X	X	• ALL phase
–	X	–	X	X	• WHEN phase
–	X	–	X	X	• AT END phase
X	X	X	X	X	SET Statement
X	X	X	X	X	• Index-name/identifier series
X	X	X	X	X	• Index name
X	X	X	X	X	• UP BY identifier/integer
X	X	X	X	X	• DOWN BY identifier/integer
X	X	X	X	X	• Index-name series

Figure C-4 (Part 2 of 2). Summary of Elements in the Table Handling Module

Summary of Elements in the Sequential I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
					LANGUAGE CONCEPTS
					User-Defined Words
X	X	X	X	X	• File name
X	X	X	X	X	• Record name
X	X	–	X	X	I-O Status
					Special Register
–	X	X,2	X	X	• LINAGE-COUNTER
					ENVIRONMENT DIVISION
					INPUT-OUTPUT SECTION
X	X	X	X	X	• FILE-CONTROL paragraph
X	X	X	X	X	• File control entry
X	X	X	X	X	– SELECT clause
–	X	–	X	X	• OPTIONAL phrase
X	X	X	X	X	– ASSIGN clause
–	–	X	–	–	• FOR MULTIPLE REEL/UNIT phrase (**ANS 1968**)
X	X	X,4	X	X	– ORGANIZATION IS SEQUENTIAL clause
X	X	X	X	X	– ACCESS MODE IS SEQUENTIAL clause
X	X	–	X	X	– FILE STATUS clause
–	X	–	X	X	– RESERVE integer AREA(S) clause
–	–	X	–	–	– RESERVE NO/integer ALTERNATE AREA(S) clause (**ANS 1968**)

Figure C-5 (Part 1 of 5). Summary of Elements in the Sequential I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	–	X,1	–	–	– PROCESSING MODE clause (**ANS 1968**)
–	–	X,1	–	–	• FILE-LIMIT clause (**ANS 1968**)
X	X	X	X	X	• I-O-CONTROL paragraph
X	X	X,2	X	X,1	– RERUN clause
X	X	X	X	X,1	– SAME AREA clause
X	X	X	X	X,1	– SAME AREA series
–	X	–	X	X	– SAME RECORD AREA clause
–	X	–	X	X	– SAME RECORD AREA series
–	X,5	–	X,1	X,1	– MULTIPLE FILE TAPE clause
					DATA DIVISION
					FILE SECTION
X	X	X	X	X	• File description entry
X	X	X	X	X	• Record description entry
X	X	X	X	X	• BLOCK CONTAINS clause
X	X	X	X	X	– Integer RECORDS/CHARACTERS
–	X	–	X	X	– Integer-1 TO integer-2 RECORDS/CHARACTERS
X,5	X,5	–	X,1	X,1	• CODE-SET clause
X	X	X,1	X,1	X,1	• DATA RECORDS clause
X	X	X,1	X,1	X,1	– Data name
X	X	X,1	X,1	X,1	– Data-name series
X	X	X	X	X	• LABEL RECORDS clause
X	X	X	X	X	– STANDARD
X	X	X,7	X	X	– OMITTED

Figure C-5 (Part 2 of 5). Summary of Elements in the Sequential I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	X	X,2	X	X	• LINAGE clause
–	X	X,2	X	X	– FOOTING phrase
–	X	–	X	X	– TOP phrase
–	X	–	X	X	– BOTTOM phrase
X	X	X	X	X	• RECORD CONTAINS clause
X	X	X	X	X	– Integer-1 TO integer-2 CHARACTERS
X,5	X,5	X,1	X,1	X,1	• VALUE OF clause
X,5	X,5	X,1	X,1	X,1	– Implementer-name IS literal
X	X	–	X,1	X,1	– Implementer-name IS literal series
–	X	–	X,1	X,1	– Implementer-name IS data-name
–	X	–	X,1	X,1	– Implementer-name IS data-name series
					PROCEDURE DIVISION
X	X	X	X	X	CLOSE statement
X	X	X	X	X	• Single file name
–	X	X	X	X	• File-name series
X,5	X,5	X	X,1	X,1	• REEL/UNIT
–	X,5	X	X	X	– WITH LOCK phrase
–	X,5	–	X,1	X,1	– WITH NO REWIND phrase
–	X,5	–	–	–	– FOR REMOVAL phrase
X	X	X	X	X	OPEN Statement
X	X	X	X	X	• Single file name
–	X	X	X	X	• File-name series
X	X	X	X	X	• INPUT phrase

Figure C-5 (Part 3 of 5). Summary of Elements in the Sequential I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	X,5	–	–	–	– REVERSED phrase
–	X,5	–	X,1	X,1	– WITH NO REWIND phrase
X	X	X	X	X	• OUTPUT phrase
–	X,5	–	X,1	X,1	– NO REWIND phrase
X,5	X,5	X	X	X	• I-O phrase
–	X,5	–	X	X	• EXTEND phrase
–	X	–	X	X	• INPUT,OUTPUT,I-O,EXTEND series
X	X	X	X	X	READ Statement
X	X	X	X	X	• INTO identifier
X	X	X,8	X	X	• AT END phrase
X,5	X,5	X,4	X	X	REWRITE Statement
X	X	X	X	X	• From identifier
X	X	–	X	X	USE Statement
X	X	–	X	X	• EXCEPTION/ERROR PROCEDURE clause
X	X	–	X	X	– ON file name
X	X	–	X	X	– ON INPUT
X	X	–	X	X	– ON OUTPUT
X	X	–	X	X	– ON I-O
–	X	–	X	X	– ON EXTEND
–	X	–	X	X	– ON file-name series
X	X	X	X	X	WRITE Statement
X	X	X	X	X	• FROM identifier
X	X,5	X	X	X	• BEFORE/AFTER ADVANCING phrase integer
X,5	X,5	X	X	X	– Integer
–	X,5	X	X	X	– Identifier

Figure C-5 (Part 4 of 5). Summary of Elements in the Sequential I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
X,5	X,5	X	X	X	– LINE(S) option
X,5	X,5	X	X	X	– PAGE
–	X,5	X	X	X	– Mnemonic name
–	X,5	X	X	X	• AT END-OF-PAGE/EOP phrase
–	–	X	–	–	• INVALID KEY phrase (**ANS 1968**)

Figure C-5 (Part 5 of 5). Summary of Elements in the Sequential I-O Module

Summary of Elements in the Relative I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
					LANGUAGE CONCEPTS
					User-Defined Words
X	X	X	X	X	• File name
X	X	X	X	X	• Record name
X	X	–	X	X	I-O Status
					ENVIRONMENT DIVISION
					INPUT-OUTPUT SECTION
X	X	X	X	X	• FILE-CONTROL paragraph
X	X	X	X	X	• File control entry
X	X	X	X	X	– SELECT clause
X	X	X	X	X	– ASSIGN clause
X	X	X,4	X	X	– ORGANIZATION IS RELATIVE clause
X	X	X	X	X	– ACCESS MODE clause
X	X	–	X	X	• SEQUENTIAL
X	X	–	X	X	– RELATIVE KEY phrase
X	X	X	X	X	• RANDOM
X	X	–	X	X	– RELATIVE KEY phrase
–	–	X	–	–	– ACTUAL KEY phrase (**ANS 1968**)
–	X	–	X	X	• DYNAMIC
–	X	–	X	X	– RELATIVE KEY phrase
X	X	–	X	X	– FILE STATUS clause

Figure C-6 (Part 1 of 4). Summary of Elements in the Relative I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	X	–	X	X	– RESERVE integer AREA(S) clause
–	–	X	–	–	– RESERVE NO/integer ALTERNATE AREA(S) clause (**ANS 1968**)
–	–	X,1	–	–	• FILE-LIMIT clause (**ANS 1968**)
–	–	X,1	–	–	• PROCESSING MODE clause (**ANS 1968**)
X	X	X	X	X	• I-O CONTROL paragraph
X	X	X	X	X,1	– RERUN clause
X	X	X	X	X,1	– SAME AREA clause
X	X	X	X	X,1	– SAME AREA series
–	X	–	X	X	– SAME RECORD AREA clause
–	X	–	X	X	– SAME RECORD AREA series
					DATA DIVISION
					FILE SECTION
X	X	X	X	X	• File description entry
X	X	X	X	X	• Record description entry
X	X	X	X	X	• BLOCK CONTAINS clause
X	X	X	X	X	– Integer RECORDS/CHARACTERS
–	X	–	X	X	– Integer-1 TO integer-2 RECORDS/CHARACTERS
X	X	X,1	X,1	X,1	• DATA RECORDS clause
X	X	X,1	X,1	X,1	– Data name
X	X	X,1	X,1	X,1	– Data-name series
X	X	X	X	X	• LABEL RECORDS clause
X	X	X	X	X	– STANDARD

Figure C-6 (Part 2 of 4). Summary of Elements in the Relative I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
X	X	X,7	X	X	– OMITTED
X	X	X	X	X	• RECORD CONTAINS clause
X	X	X	X	X	– Integer-1 TO integer-2 CHARACTERS
X,5	X,5	X,1	X,1	X,1	• VALUE OF clause
X,5	X,5	X,1	X,1	X,1	– Implementer-name IS literal
X,5	X,5	–	X,1	X,1	– Implementer-name IS literal series
–	X,5	–	X,1	X,1	– Implementer-name IS data-name
–	X,5	–	X,1	X,1	– Implementer-name IS data-name series
					PROCEDURE DIVISION
X	X	X	X	X	CLOSE Statement
X	X	X	X	X	• Single file name
X	X	X	X	X	• File-name series
X,5	X,5	X	X	X	• WITH LOCK phrase
X	X	X	X	X	DELETE Statement
X	X	–	X	X	• INVALID KEY phrase
X	X	X	X	X	OPEN Statement
X	X	X	X	X	• Single file name
X	X	X	X	X	• File-name series
X	X	X	X	X	• INPUT phrase
X	X	X	X	X	• OUTPUT phrase
X,5	X,5	X	X	X	• I-O phrase
X	X	X	X	X	• INPUT, OUTPUT, and I-O series
X	X	X	X	X	READ statement
X	X	X	X	X	• INTO identifier

Figure C-6 (Part 3 of 4). Summary of Elements in the Relative I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	X	–	X	X	• NEXT phrase
X	X	X,8	X	X	• AT END phrase
X	X	X,8	X	X	• INVALID KEY phrase
X,5	X,5	X	X	X	REWRITE Statement
X	X	X	X	X	• FROM identifier
X	X	X	X	X	• INVALID KEY phrase
–	–	X,1	–	–	SEEK Statement (**ANS 1968**)
–	X	–	X	X	START Statement
–	X	–	X	X	• KEY IS phrase
–	X	–	X	X	• INVALID KEY phrase
X	X	–	X	X	USE Statement
X	X	–	X	X	• EXCEPTION/ERROR PROCEDURE phrase
X	X	–	X	X	– ON file-name
X	X	–	X	X	– ON INPUT
X	X	–	X	X	– ON OUTPUT
X	X	–	X	X	– ON I-O
–	X	–	X	X	– ON file-name series
X	X	X	X	X	WRITE Statement
X	X	X	X	X	• FROM identifier
X	X	X,8	X	X	• INVALID KEY phrase

Figure C-6 (Part 4 of 4). Summary of Elements in the Relative I-O Module

Summary of Elements in the Indexed I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
					LANGUAGE CONCEPTS
					User-Defined Words
X	X	X	X	X	• File name
X	X	X	X	X	• Record name
X	X	–	X	X	I-O Status
					ENVIRONMENT DIVISION
					INPUT-OUTPUT SECTION
X	X	X	X	X	• FILE-CONTROL paragraph
X	X	X	X	X	• File control entry
X	X	X	X	X	– SELECT clause
X	X	X	X	X	– ASSIGN clause
X	X	X	X,4	X,4	– ORGANIZATION IS INDEXED clause
X	X	X	X	X	– ACCESS MODE clause
X	X	X	X	X	• SEQUENTIAL
X	X	X	X	X	• RANDOM
–	X	–	X	X	• DYNAMIC
X	X	X	X	X	– RECORD KEY clause
–	–	–	X	X	• WITH DUPLICATES phrase (**IBM Extension**)
–	–	–	–	X	• Noncontiguous key (**IBM Extension**)
–	X	–	–	–	– ALTERNATE RECORD KEY clause
–	X	–	–	–	• WITH DUPLICATES phrase

Figure C-7 (Part 1 of 5). Summary of Elements in the Indexed I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	–	X	–	–	– NOMINAL KEY clause (**IBM Extension**)
X	X	–	X	X	– FILE STATUS clause
–	X	X	X	X	– RESERVE integer AREA(S) clause
–	–	X	–	–	– RESERVE NO/integer ALTERNATE AREA(S) clause (**ANS 1968**)
–	–	X,1	–	–	– FILE-LIMIT clause (**ANS 1968**)
–	–	X,1	–	–	– PROCESSING MODE IS clause (**ANS 1968**)
X	X	X	X	X	• I-O-CONTROL paragraph
X	X	X	X	X	– RERUN clause
X	X	X	X	X	– SAME AREA clause
X	X	X	X	X	– SAME AREA series
–	X	–	X	X	– SAME RECORD AREA clause
–	X	–	X	X	– SAME RECORD AREA series
–	–	X	X	X	– APPLY CORE-INDEX series (**IBM Extension**)
					DATA DIVISION
					FILE SECTION
X	X	X	X	X	• File description entry
X	X	X	X	X	• Record description entry
X	X	X	X	X	• BLOCK CONTAINS clause
X	X	X	X	X	– Integer RECORDS/CHARACTERS
–	X	–	X	X	– Integer-1 TO integer-2 RECORDS/CHARACTERS
X	X	X,1	X,1	X,1	• DATA RECORDS clause

Figure C-7 (Part 2 of 5). Summary of Elements in the Indexed I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
X	X	X,1	X,1	X,1	– Data name
X	X	X,1	X,1	X,1	– Data-name series
X	X	X	X	X	• LABEL RECORDS clause
X	X	X	X	X	– STANDARD
X	X	X,7	X	X	– OMITTED
X	X	X	X	X	• RECORD CONTAINS clause
X	X	X	X	X	– Integer-1 TO integer-2 CHARACTERS
X,5	X,5	X,1	X,1	X,1	• VALUE OF clause
X,5	X,5	X,1	X,1	X,1	– Implementer-name IS literal
X,5	X,5	–	X,1	X,1	– Implementer-name IS literal series
–	X,5	–	X,1	X,1	– Implementer-name IS data-name
–	X,5	–	X,1	X,1	– Implementer-name IS data-name series
					PROCEDURE DIVISION
X	X	X	X	X	CLOSE Statement
X	X	X	X	X	• Single file name
X	X	X	X	X	• File-name series
X,5	X,5	X	X	X	• WITH LOCK phrase
X	X	–	X	X	DELETE Statement
X	X	–	X	X	• INVALID KEY phrase
X	X	X	X	X	OPEN Statement
X	X	X	X	X	• Single file name
X	X	X	X	X	• File-name series
X	X	X	X	X	• INPUT phrase
X	X	X	X	X	• OUTPUT phrase

Figure C-7 (Part 3 of 5). Summary of Elements in the Indexed I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
X,5	X,5	X	X	X	• I-O phrase
X	X	X	X	X	• INPUT, OUTPUT, and I-O series
X	X	X	X	X	READ Statement
X	X	X	X	X	• INTO identifier
–	X	–	–	–	• KEY IS phrase
–	–	–	–	X	• FIRST phrase (**IBM Extension**)
–	–	–	–	X	• LAST phrase (**IBM Extension**)
–	–	–	–	X	• PRIOR phrase (**IBM Extension**)
–	X	–	X	X	• NEXT phrase
X	X	X,8	X	X	• AT END phrase
X	X	X,8	X	X	• INVALID KEY phrase
X,5	X,5	X	X	X	REWRITE Statement
X	X	X	X	X	• FROM identifier
X	X	X,8	X	X	• INVALID KEY phrase
–	X	X	X	X	START Statement
–	X	X,4	X	X	• KEY IS phrase
–	–	–	–	X	• Noncontiguous key (**IBM Extension**)
–	X	X,8	X	X	• INVALID KEY phrase
X	X	–	X	X	USE Statement
X	X	–	X	X	• EXCEPTION/ERROR PROCEDURE phrase
X	X	–	X	X	– ON file name
X	X	–	X	X	– ON INPUT
X	X	–	X	X	– ON OUTPUT
X	X	–	X	X	– ON I-O

Figure C-7 (Part 4 of 5). Summary of Elements in the Indexed I-O Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
–	X	–	X	X	– ON file-name series
X	X	X	X	X	WRITE Statement
X	X	X	X	X	• FROM identifier
X	X	X	X	X	• INVALID KEY phrase

Figure C-7 (Part 5 of 5). Summary of Elements in the Indexed I-O Module

Summary of Elements in the Sort-Merge Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
					LANGUAGE CONCEPTS
					User-Defined Words
X	X	–	X	X	• File name
					ENVIRONMENT DIVISION
					INPUT-OUTPUT SECTION
X	X	–	X	X	• FILE-CONTROL paragraph
X	X	–	X	X	• File control entry
X	X	–	X	X	– SELECT clause
X	X	–	X	X	– ASSIGN clause
–	X	–	X	X	• I-O CONTROL paragraph
–	X	–	X	X	– SAME RECORD AREA clause
–	X	–	X	X	– SAME RECORD AREA series
–	X	–	X	X	– SAME SORT/SORT-MERGE AREA clause
–	X	–	X	X	– SAME SORT/SORT-MERGE AREA series
					DATA DIVISION
					FILE SECTION
X	X	–	X	X	• File description entry
X	X	–	X	X	• Record description entry
X	X	–	X	X	• DATA RECORDS clause
X	X	–	X	X	– Data name
X	X	–	X	X	– Data-name series
X	X	–	X	X	• RECORD CONTAINS clause

Figure C-8 (Part 1 of 3). Summary of Elements in the Sort-Merge Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
X	X	–	X	X	– Integer-1 TO integer-2 CHARACTERS
					PROCEDURE DIVISION
–	X	–	X	X	MERGE Statement
–	X	–	X	X	• KEY data name
–	X	–	X	X	– Data-name series
–	X	–	X	X	• ASCENDING series
–	X	–	X	X	• DESCENDING series
–	X	–	X	X	• Mixed ASCENDING/DESCENDING
–	X	–	X	X	• COLLATING SEQUENCE phrase
–	X	–			• USING phrase
–	X	–	X	X	• OUTPUT PROCEDURE phrase
–	X	–	X	X	• GIVING phrase
X	X	–	X	X	RELEASE Statement
X	X	–	X	X	• FROM phrase
X	X	–	X	X	RETURN Statement
X	X	–	X	X	• INTO phrase
X	X	–	X	X	• AT END phrase
X	X	–	X	X	SORT Statement
X	X	–	X	X	• Program may contain one SORT statement
–	X	–	X	X	• Program may contain multiple SORT statements
X	X	–	X	X	• KEY data name
X	X	–	X	X	– Data-name series
X	X	–	X	X	• ASCENDING series
X	X	–	X	X	• DESCENDING series

Figure C-8 (Part 2 of 3). Summary of Elements in the Sort-Merge Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
X	X	–	X	X	• Mixed ASCENDING/DESCENDING
–	X	–	X	X	• COLLATING SEQUENCE phrase
X	X	–	X	X	• INPUT PROCEDURE phrase
X	X	–	X	X	• USING phrase
X	X	–	X	X	• OUTPUT PROCEDURE phrase
X	X	–	X	X	• GIVING phrase

Figure C-8 (Part 3 of 3). Summary of Elements in the Sort-Merge Module

Summary of Elements in the Debug Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
					LANGUAGE CONCEPTS
					Special Registers
X	X	–	X	X	• DEBUG-ITEM
					ENVIRONMENT DIVISION
					CONFIGURATION SECTION
					• SOURCE-COMPUTER paragraph
X	X	–	X	X	– WITH DEBUGGING MODE clause
					PROCEDURE DIVISION
X	X	–	X	X	USE FOR DEBUGGING Statement
X	X	–	X	X	• Procedure name
X	X	–	X	X	• Procedure-name series
X	X	–	X	X	• ALL PROCEDURES
–	X	–	–	–	• ALL REFERENCES OF identifier series
–	X	–	–	–	• File-name series
–	X	–	–	–	• Cd-name series
–	–	X	X	X	EXHIBIT Statement (**IBM Extension**)
–	–	X	X	X	• NAMED/CHANGED NAMED option
–	–	X	X	X	• Identifier series
–	–	X	X	X	READY TRACE Statement (**IBM Extension**)
–	–	X	X	X	RESET TRACE Statement (**IBM Extension**)

Figure C-9 (Part 1 of 2). Summary of Elements in the Debug Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
					ALL DIVISIONS
X	X	–	X	X	Debugging Lines (permitted after the OBJECT-COMPUTER paragraph)

Figure C-9 (Part 2 of 2). Summary of Elements in the Debug Module

Summary of Elements in the Inter-Program Communication Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
					DATA DIVISION
X	X	X	X	X	Linkage Section
					PROCEDURE DIVISION
					Procedure Division Header
X	X	X	X	X	• USING phrase
X	X	X	X	X	CALL Statement
X	X	X	X	X	• Literal
–	X	–	–	–	• identifier
X	X	–	–	–	• ALL PROCEDURES
X	X	X	X	X	• USING phrase
–	X	–	–	–	• ON OVERFLOW phrase
–	X	–	–	–	CANCEL Statement
X	X	X	X	X	EXIT PROGRAM Statement

Figure C-10. Summary of Elements in the Inter-Program Communication Module

Summary of Elements in the Segmentation Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
					LANGUAGE CONCEPTS
					User-Defined Words
X	X	X	X	X	• Segment number
					ENVIRONMENT DIVISION
					CONFIGURATION SECTION
					• OBJECT-COMPUTER paragraph
–	X	–	X,1	X,1	– SEGMENT-LIMIT clause
					PROCEDURE DIVISION
X	X	X	X	X	Segment Numbers
X	X	X	X	X	• Fixed segment number 0-49
X	X	X	X	X	• Independent segment number 50-99
–	X	–	X	X	• Sections with the same number need not be contiguous

Figure C-11. Summary of Elements in the Segmentation Module

Summary of Elements in the Library Module

ANS 1	ANS 2	S/3	S/34	S/36	Elements
					LANGUAGE CONCEPTS
					User-Defined Words
X,5	X,5	X	X	X	• Text name
–	X,5	–	X	X	• Library name
					ALL DIVISIONS
X	X	X	X	X	COPY Statement
X	X	X	X	X	• Text name
–	X	–	X	X	• OF/IN library-name
–	X	–	X	X	• REPLACING phrase

Figure C-12. Summary of Elements in the Library Module

File Processing Summary and Status Key Values

Appendix D. File Processing Summary and Status Key Values

Figure D-1 lists the required and optional entries for various types of supported file structures. Following are the codes used:

Code	Explanation
.	Not applicable
C	Optional entry; treated as comments only
I	Optional for a file opened for input or input-output
R	Required
X	Required; syntax-checked, but treated as documentation
N	Optional for an input file; treated as comments
O	Optional

Device Type	Printer	Disk Seq	Disk Rel Seq	Disk Rel Random	Disk Rel Dynamic	Disk Idx Seq	Disk Idx Random	Disk Idx Dynamic	Work Station
ENVIRONMENT DIVISION									
RERUN . . . RECORDS	C	C	C	C	C	C	C	C	C
SAME	O	O	O	O	O	O	O	O	O
AREA	C	C	C	C	C	C	C	C	C
RECORD AREA	O	O	.	O	O	O	O	O	O
SORT AREA	O	O	O	O	O	O	O	O	O
SORT MERGE AREA	O	O	O	O	O	O	O	O	O
SELECT	R	R	R	R	R	R	R	R	R
ASSIGN	R	R	R	R	R	R	R	R	R
OPTIONAL	.	I	.	.	.	.	.	.	.
ORGANIZATION	O	O	R	R	R	R	R	R	R
SEQUENTIAL	O	O	.	.	.	.	.	.	.
RELATIVE	.	.	R	R	R	.	.	.	.
INDEXED	.	.	.	.	.	R	R	R	.
TRANSACTION	.	.	.	.	.	.	.	.	R
ACCESS	O	O	O	R	R	O	R	R	O
SEQUENTIAL	O	O	O	.	.	O	.	.	O
RANDOM	.	.	.	R	.	.	R	.	.
DYNAMIC	.	.	.	.	R	.	.	R	.
RESERVE	C	O	O	C	C	O	C	C	.
RELATIVE KEY	.	.	O	R	R	.	.	.	.
RECORD KEY	.	.	.	.	.	R	R	R	.
DUPLICATES	.	.	.	.	.	O	O	O	.
FILE STATUS	O	O	O	O	O	O	O	O	O
CONTROL-AREA	.	.	.	.	.	.	.	.	O

Figure D-1 (Part 1 of 3). File Processing

Device Type	Printer	Disk Seq	Disk Rel Seq	Disk Rel Random	Disk Rel Dynamic	Disk Idx Seq	Disk Idx Random	Disk Idx Dynamic	Work Station
DATA DIVISION									
LABEL RECORDS	X	X	X	X	X	X	X	X	X
STANDARD	.	R	R	R	R	R	R	R	O
OMITTED	R	.	.	.	.	.	.	.	O
VALUE OF	C	C	C	C	C	C	C	C	C
BLOCK CONTAINS	O	O	O	O	O	O	O	O	.
RECORD CONTAINS	O	O	O	O	O	O	O	O	O
DATA RECORDS	O	O	O	O	O	O	O	O	O
CODE-SET	C	C	C	C	C	C	C	C	.
LINAGE	O	.	.	.	.	.	.	.	.
PROCEDURE DIVISION									
OPEN	R	R	R	R	R	R	R	R	R
INPUT	.	O	O	O	O	O	O	O	.
OUTPUT	R	O	O	O	O	O	O	O	.
I-O	.	O	O	O	O	O	O	O	R
NO REWIND	C	C	.	.	.	.	.	.	.
REVERSED	.	N	.	.	.	.	.	.	.
EXTEND	.	O	.	.	.	.	.	.	.
CLOSE	R	R	R	R	R	R	R	R	R
REEL/UNIT	C	C	C	C	C	C	C	C	C
REMOVAL	C	C	C	C	C	C	C	C	C
NO REWIND	C	C	C	C	C	C	C	C	C
NO REWIND	C	C	C	C	C	C	C	C	C
WITH LOCK	O	O	O	O	O	O	O	O	O
READ	.	I	I	I	I	I	I	I	I
NEXT	.	.	I	.	I	I	.	I	.
FIRST	.	.	.	.	.	.	.	I *	
LAST	.	.	.	.	.	.	.	I *	
PRIOR	.	.	.	.	.	.	.	I *	
INTO	.	I	I	I	I	I	I	I	I
AT END	.	I	I	.	I	I	.	I	I
INVALID KEY	.	.	.	I	I	.	I	I	.

* Allowed only for DATABASE device type

Figure D-1 (Part 2 of 3). File Processing

Device Type	Printer	Disk Seq	Disk Rel Seq	Disk Rel Random	Disk Rel Dynamic	Disk Idx Seq	Disk Idx Random	Disk Idx Dynamic	Work Station
WRITE	O	O	O	O	O	O	O	O	O
FROM	O	O	O	O	O	O	O	O	O
INVALID KEY	.	.	O	O	O	O	O	O	.
ADVANCING	O	.	.	.	.	.	.	.	.
AT END-OF-PAGE	O	.	.	.	.	.	.	.	.
FORMAT IS	.	.	.	.	.	.	.	.	R
INDICATOR	.	.	.	.	.	.	.	.	O
START	.	.	O	.	O	O	.	O	.
KEY	.	.	O	.	O	O	.	O	.
INVALID KEY	.	.	O	.	O	O	.	O	.
REWRITE	.	O	O	O	O	O	O	O	.
FROM	.	O	O	O	O	O	O	O	.
INVALID KEY	.	.	O	O	O	O	O	O	.
DELETE	.	.	O	O	O	O	O	O	.
INVALID KEY	.	.	.	O	O	.	O	O	.
USE	O	O	O	O	O	O	O	O	O
EXCEPTION/ERROR	O	O	O	O	O	O	O	O	O
FOR DEBUGGING	O	O	O	O	O	O	O	O	O

Figure D-1 (Part 3 of 3). File Processing

Status Key 1	Status Key 2	Meaning
0		Successful completion
	0	No further information
	1	Initial READ from a REQUESTOR (IBM Extension)
1	0	At end of file (no outstanding invites)
2		Invalid key
	1	Sequence error[1]
	2	Duplicate key when duplicates are not allowed
	3	No record found
	4	Boundary violation--indexed or relative file
3		Permanent error
	0	No further information[2]
	4	Boundary violation--sequential file
9[3]		Other errors (IBM Extension)
	0	Invalid update, add, or output operation
	2	Logic error (I/O to unopened file, file locked, already OPEN, already CLOSED, or invalid operation)[4]
	4	No current record pointer for I/O request
	5	Invalid or incomplete file information[5]
	9	Undefined
	A	STOP requested by system operator
	C	Acquire operation failed; display station not in standby mode

Figure D-2 (Part 1 of 2). Status Key Values and Meanings

Status Key 1	Status Key 2	Meaning
	D	Terminal operator released display station with INQUIRY key
	E	Program released its requestor; I/O rejected
	F	Acquire operation failed; either operator signed on is unauthorized or program is unauthorized to use resources
	G	Input data rejected; buffer too small
	H	Acquire operation failed, resource is unavailable or currently owned by another program
	I	Write operation failed; input data already received by data management
	K	IDDU format not found or cannot be retrieved
	N	Temporary error (error during communications session)

Figure D-2 (Part 2 of 2). Status Key Values and Meanings

---

1 This can also occur when the primary key was invalidly changed when updating a key for a multiple index file.

2 This can occur when a remote file is used and one of the following conditions exist when the I/O operator is requested:

- The remote system reported a "permanent error" condition.
- A communication failure occurs.
- For the input operation, the record number is larger than the maximum value that can be handled in the requesting system.

3 Program termination will not always occur.

4 This can occur when the operation performed on an open file is not possible; for example,

- Reading and output file.
- Writing to an input file.
- Updating a file that is not opened as I/O.

5 This can occur when either of the following conditions exist when the file is opened:

- WITH DUPLICATES is specified on the RECORD KEY clause, and duplicate keys are not allowed for the file.
- WITH DUPLICATES is not specified on the RECORD KEY clause, and duplicate keys are allowed for the file.

EBCDIC and ASCII Collating Sequences

Appendix E. EBCDIC and ASCII Collating Sequences

This appendix contains the ascending collating sequences for the following character sets:

- EBCDIC (Extended Binary Coded Decimal Interchange Code)
- ASCII (American National Standard Code for Information Interchange).

The tables show:

- Decimal positions within the sequence
- The binary representation
- The symbol
- The meaning for each character
- The matching decimal position within the other sequence.

Note: When you are using the literal option of the alphabet-name clause, you must add 1 to the number shown in this appendix to specify the corresponding character. (The numbers in this appendix run from 0 through 255; the numbers in the literal option run from 1 through 256.)

EBCDIC Collating Sequence

Collating Sequence	Binary Representation	Symbol	Meaning	ASCII Number	
0	00000000			0	
.					
64	01000000	SP	Space	32	
.					
74	01001010	¢	Cent sign	91	*
75	01001011	.	Period, decimal point	46	
76	01001100	<	Less-than sign	60	
77	01001101	(	Left parenthesis	40	
78	01001110	+	Plus sign	43	
79	01001111	|	Vertical bar, logical OR	33	*
80	01010000	&	Ampersand	38	
.					
.					
90	01011010	!	Exclamation point	93	*
91	01011011	$	Dollar sign	36	
92	01011100	*	Asterisk	42	
93	01011101	)	Right parenthesis	41	
94	01011110	;	Semicolon	59	
95	01011111	¬	Logical NOT	94	
96	01100000	-	Minus, hyphen	45	
97	01100001	/	Slash	47	
.					
.					
106	01101010	|	Broken vertical bar	124	*
107	01101011	,	Comma	44	
108	01101100	%	Percent sign	37	
109	01101101	_	Underscore	95	
110	01101110	>	Greater-than sign	62	
111	01101111	?	Question mark	63	
.					
.					

Note: The symbols with an asterisk (*) in the rightmost column are not matched with the same symbol in the other collating sequence.

Collating Sequence	Binary Representation	Symbol	Meaning	ASCII Number
121	01111001	\	Grave accent	96
122	01111010	:	Colon	58
123	01111011	#	Number sign	35
124	01111100	@	At sign	64
125	01111101	'	Apostrophe, prime	39
126	01111110	=	Equal sign	61
127	01111111	"	Quotation mark	34
.				
129	10000001	a		97
130	10000010	b		98
131	10000011	c		99
132	10000100	d		100
133	10000101	e		101
134	10000110	f		102
135	10000111	g		103
136	10001000	h		104
137	10001001	i		105
.				
.				
145	10010001	j		106
146	10010010	k		107
147	10010011	l		108
148	10010100	m		109
149	10010101	n		110
150	10010110	o		111
151	10010111	p		112
152	10011000	q		113
153	10011001	r		114
.				
161	10100001	~	Tilde	126
162	10100010	s		115
163	10100011	t		116
164	10100100	u		117
165	10100101	v		118
166	10100110	w		119
167	10100111	x		120
168	10101000	y		121
169	10101001	z		122
.				
.				

Note: The symbols with an asterisk (*) in the rightmost column are not matched with the same symbol in the other collating sequence.

Collating Sequence	Binary Representation	Symbol	Meaning	ASCII Number
192	11000000	{	Left brace	123
193	11000001	A		65
194	11000010	B		66
195	11000011	C		67
196	11000100	D		68
197	11000101	E		69
198	11000110	F		70
199	11000111	G		71
200	11001000	H		72
201	11001001	I		73
.				
.				
208	11010000	}	Right brace	125
209	11010001	J		74
210	11010010	K		75
211	11010011	L		76
212	11010100	M		77
213	11010101	N		78
214	11010110	O		79
215	11010111	P		80
216	11011000	Q		81
217	11011001	R		82
.				
224	11100000	\	Reverse slant	92
.				
226	11100010	S		83
227	11100011	T		84
228	11100100	U		85
229	11100101	V		86
230	11100110	W		87
231	11100111	X		88
232	11101000	Y		89
233	11101001	Z		90
.				
.				
240	11110000	0		48
241	11110001	1		49
242	11110010	2		50
243	11110011	3		51
244	11110100	4		52
245	11110101	5		53
246	11110110	6		54
247	11110111	7		55
248	11111000	8		56
249	11111001	9		57
.				
.				
.				
255				

Note: The symbols with an asterisk (*) in the rightmost column are not matched with the same symbol in the other collating sequence.

ASCII Collating Sequence

Collating Sequence	Binary Representation	Symbol	Meaning	EBCDIC Number	
0	00000000		Null	0	
.					
.					
.					
32	00100000	SP	Space	64	
33	00100001	!	Exclamation point	79	*
34	00100010	"	Quotation mark	127	
35	00100011	#	Number sign	123	
36	00100100	$	Dollar sign	91	
37	00100101	%	Percent sign	108	
38	00100110	&	Ampersand	80	
39	00100111	'	Apostrophe, prime	125	
40	00101000	(	Left parenthesis	77	
41	00101001	)	Right parenthesis	93	
42	00101010	*	Asterisk	92	
43	00101011	+	Plus sign	78	
44	00101100	,	Comma	107	
45	00101101	-	Hyphen, minus	96	
46	00101110	.	Period, decimal point	75	
47	00101111	/	Slash	97	
48	00110000	0		240	
49	00110001	1		241	
50	00110010	2		242	
51	00110011	3		243	
52	00110100	4		244	
53	00110101	5		245	
54	00110110	6		246	
55	00110111	7		247	
56	00111000	8		248	
57	00111001	9		249	
58	00111010	:	Colon	122	
59	00111011	;	Semicolon	94	
60	00111100	<	Less-than sign	76	
61	00111101	=	Equal sign	126	
62	00111110	>	Greater-than sign	110	
63	00111111	?	Question mark	111	
64	01000000	@	At sign	124	

Note: The symbols with an asterisk (*) in the rightmost column are not matched with the same symbol in the other collating sequence.

Collating Sequence	Binary Representation	Symbol	Meaning	EBCDIC Number	
65	01000001	A		193	
66	01000010	B		194	
67	01000011	C		195	
68	01000100	D		196	
69	01000101	E		197	
70	01000110	F		198	
71	01000111	G		199	
72	01001000	H		200	
73	01001001	I		201	
74	01001010	J		209	
75	01001011	K		210	
76	01001100	L		211	
77	01001101	M		212	
78	01001110	N		213	
79	01001111	O		214	
80	01010000	P		215	
81	01010001	Q		216	
82	01010010	R		217	
83	01010011	S		226	
84	01010100	T		227	
85	01010101	U		228	
86	01010110	V		229	
87	01010111	W		230	
88	01011000	X		231	
89	01011001	Y		232	
90	01011010	Z		233	
91	01011011	[	Left bracket	74	*
92	01011100	\	Reverse slant	224	
93	01011101	]	Right bracket	90	*
94	01011110	^	Circumflex	95	
		¬	Logical NOT		
95	01011111	_	Underscore	109	
96	01100000	`	Grave accent	121	

Note: The symbols with an asterisk (*) in the rightmost column are not matched with the same symbol in the other collating sequence.

Collating Sequence	Binary Representation	Symbol	Meaning	EBCDIC Number	
97	01100001	a		129	
98	01100010	b		130	
99	01100011	c		131	
100	01100100	d		132	
101	01100101	e		133	
102	01100110	f		134	
103	01100111	g		135	
104	01101000	h		136	
105	01101001	i		137	
106	01101010	j		145	
107	01101011	k		146	
108	01101100	l		147	
109	01101101	m		148	
110	01101110	n		149	
111	01101111	o		150	
112	01110000	p		151	
113	01110001	q		152	
114	01110010	r		153	
115	01110011	s		162	
116	01110100	t		163	
117	01110101	u		164	
118	01110110	v		165	
119	01110111	w		166	
120	01111000	x		167	
121	01111001	y		168	
122	01111010	z		169	
123	01111011	{	Left brace	192	
124	01111100	|	Vertical line	106	*
125	01111101	}	Right brace	208	
126	01111110	~	Tilde	161	

Note: The symbols with an asterisk (*) in the rightmost column are not matched with the same symbol in the other collating sequence.

IBM American National Standard COBOL Reserved Words

Appendix F. IBM American National Standard COBOL Reserved Words

No word in the following two lists should appear as a programmer-defined name.

Reserved Words Used by the System/36 COBOL Compiler

Words preceded by an asterisk (*) are not included in the American National Standard COBOL, X3.23-1974, reserved word list.

ACCEPT
ACCESS
*ACQUIRE
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
ALSO
ALTER
ALTERNATE
AND
*APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
*ATTRIBUTE-DATA
AUTHOR

BEFORE
BLANK
BLOCK
BOTTOM
BY

CALL
*CHANGED
CHARACTER
CHARACTERS
CLOSE
CODE-SET
COLLATING
COMMA
COMP
*COMP-3
*COMP-4
COMPUTATIONAL
*COMPUTATIONAL-3
*COMPUTATIONAL-4
COMPUTE
CONFIGURATION
CONTAINS
*CONTROL-AREA
COPY
CORR
CORRESPONDING
COUNT
*CORE-INDEX
*CSP
CURRENCY
*C01

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DISPLAY
DIVIDE
DIVISION
DOWN
*DROP
DUPLICATES
DYNAMIC

ELSE
END
END-OF-PAGE
ENTER
ENVIRONMENT
EOP
EQUAL
ERROR
EVERY
EXCEPTION
*EXHIBIT
EXIT
EXTEND

FD
FILE
FILE-CONTROL
FILLER
FIRST
FOOTING
FOR
*FORMAT
FROM

GIVING
GO
GREATER

HIGH-VALUE
HIGH-VALUES

IDENTIFICATION
IF
I-O
I-O-CONTROL
IN
INDEX
INDEXED
*INDIC
*INDICATOR
*INDICATORS
INITIAL
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS

JUST
JUSTIFIED

KEY

LABEL
LAST
LEADING
LEFT
LESS
LINAGE
LINAGE-COUNTER
LINE
LINES
LINKAGE
*LOCAL-DATA
LOCK

LOW-VALUE
LOW-VALUES

MEMORY
MERGE
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY

*NAMED
NATIVE
NEGATIVE
NEXT
NO
NOT
NUMERIC

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION
OUTPUT
OVERFLOW

PAGE
PERFORM
PIC
PICTURE
PLUS
POINTER
POSITIVE
*PRIOR
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID

QUOTE
QUOTES

RANDOM
READ
RECORD
RECORDS
REDEFINES
REEL
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
*REQUESTOR
RERUN
RESERVE
RESET
RETURN
REVERSED
REWIND
REWRITE
RIGHT
*ROLLING
ROUNDED
RUN

SAME
SD
SEARCH
SECTION
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
START
*STARTING
STATUS
STOP
STRING
SUBTRACT
SYNC
SYNCHRONIZED
*SYSTEM-CONSOLE
*SYSTEM-SHUTDOWN

TALLYING
TERMINAL
THAN
*THEN
THROUGH
THRU
TIME
TIMES
TO
TOP
*TRACE
TRAILING
*TRANSACTION
*TRUE

UNIT
UNSTRING
UNTIL
UP
UPON
*UPSI-0
*UPSI-1
*UPSI-2
*UPSI-3
*UPSI-4
*UPSI-5
*UPSI-6
*UPSI-7
USAGE

USE
USING

VALUE
VALUES
VARYING

WHEN
WITH
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

Reserved Words Not Used by the System/36 COBOL Compiler

The reserved words in the following list are not used by the System/36 COBOL compiler and should not be used if compatibility with other American National Standard COBOL compilers and CODASYL COBOL is desired.

CANCEL
CD
CF
CH
CLOCK-UNITS
COBOL
CODE
COLUMN
COMMUNICATION
CONTROL
CONTROLS

DE
DESTINATION
DETAIL
DISABLE

EGI
EMI
ENABLE
ESI

FINAL

GENERATE
GROUP

HEADING

INDICATE
INITIATE

LENGTH
LIMIT
LIMITS
LINE-COUNTER

MESSAGE

NUMBER

PAGE-COUNTER
PF
PH
POSITION
PRINTING

QUEUE

RD
RECEIVE
REFERENCES
REPORT
REPORTING
REPORTS
RF
RH

SEND
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUM
SUPPRESS
SYMBOLIC

TABLE
TAPE
TERMINATE
TEXT
TYPE

Glossary

abbreviated combined relational condition. A combined condition that omits a common subject, or a common subject and common relational operator, from a consecutive sequence of relation conditions. For example, (A and B) or (A and C) can be abbreviated A and (B or C).

access mode. A method used to read a specific logical record from, or to write a specific logical record into a file assigned to an input/output device. Access can be sequential (records are referred to one after another in the order in which they appear in the file), it can be random (the individual record can be referred to in a nonsequential manner), or it can be dynamic (records can be accessed sequentially or randomly, depending on the form of the specific input/output request).

actual decimal point. The representation, using the decimal point character (. or ,), of the decimal point position in a data item. The actual decimal point appears in printed reports and requires a position in storage. Contrast with *assumed decimal point.*

advanced program-to-program communication (APPC). SSP-ICF communications support that allows System/36 to communicate with other systems having the same support. APPC is the way that System/36 puts the IBM SNA LU-6.2 protocol into effect.

alphabet name. A user-defined word, in the SPECIAL-NAMES paragraph, that names a character set or collating sequence.

alphabetic character. A character that is one of the 26 uppercase characters of the alphabet, or a space.

alphanumeric character. Any character in the computer's character set.

alphanumeric edited character. An alphanumeric data item whose PICTURE character string contains at least one B, O, or 1.

alternative index. An index that is built after an indexed file or an alternative indexed file is created and that provides a different order for reading or writing records in the file. Contrast with *primary index.*

American National Standard Code for Information Interchange (ASCII). The code developed by for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

American National Standards Institute (ANSI). An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

ANSI. See *American National Standards Institute.*

APPC. See *advanced program-to-program communication.*

arithmetic expression. An arithmetic expression can be an identifier for a numeric elementary item, a numeric literal, such identifiers and literals separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

arithmetic operator. A symbol (1 character or a 2-character set) that indicates the arithmetic operation to be performed. Arithmetic operators include: + (addition), - (subtraction), * (multiplication), / (division), ** (exponentiation).

ascending key. The values by which data is ordered from the lowest value to the highest value of the key according to the rules for comparing data items.

ascending key sequence. The arrangement of data in order from the lowest value of the key field to the highest value of the key field. Contrast with *descending key sequence.*

ASCII. See *American National Standard Code for Information Interchange.*

assignment name. A word that associates a file name with an external device.

assumed decimal point. A logical decimal point position that is used to align a value properly for calculation; the assumed decimal point does not occupy a storage

position in a data item. Contrast with *actual decimal point.*

AT END condition. A condition that occurs when the following statements are run: a READ statement for a sequentially accessed file; a RETURN statement if a logical record does not exist for the associated sort or merge file; a SEARCH statement if the search operation ends without satisfying the condition specified in any of the associated WHEN phrases.

binary item. A numeric data item that is represented within the computer in binary digits (that is, as a number in the base 2). If the number is signed, the sign is the leftmost bit in each item.

block. A group of records that is recorded or processed as a unit. Same as *physical record.*

Boolean data. A category of data items that are limited to a value of one or zero.

Boolean literal. See *literal.*

buffer. An area of storage, temporarily reserved for performing input or output, into which data is read or from which data is written.

called program. A program that is the object of a CALL statement combined at object time with the calling program to produce a run unit.

calling program. A program that executes a CALL to another program.

character. One of a set of indivisible symbols that can be arranged in sequence to express information.

character set. All the valid COBOL characters.

character string. A sequence of characters that form a COBOL word, a literal, a PICTURE character string, or a comment entry.

class condition. A condition that states that the content of an item is all alphabetic or all numeric.

clause. An ordered set of consecutive COBOL character strings whose purpose is to specify an attribute of an entry.

COBOL (common business-oriented language). A high-level programming language, similar to English, that is used primarily for commercial data processing.

COBOL character set. The following 51 characters:

Character	Meaning
0, 1, ..., 9	Digit
A, B, ..., Z	Letter
	Space (blank)
+	Plus sign
-	Minus sign (hyphen)
*	Asterisk
/	Stroke (virgule, slash)
=	Equal sign
$	Currency sign
,	Comma (decimal point)
;	Semicolon
.	Period (decimal point)
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
>	Greater than symbol
<	Less than symbol

collating sequence. The sequence in which characters are ordered within the computer for sorting, combining, or comparing.

column. A character position within a print line. The columns are numbered from 1, by 1, starting at the leftmost character position of the print line and extending to the rightmost position of the print line.

combined condition. A condition that is the result of connecting two or more conditions with the AND or the OR logical operator.

comment. An annotation in the Identification Division or Procedure Division of a COBOL source program. A comment is ignored by the compiler. As an IBM extension, comments may be included at any point in a COBOL source program.

comment entry. An entry in the Identification Division that is not translated by the compiler.

comment line. A source program line that is not translated by the compiler. The comment line can be used to document the program. A special form of the comment line can be used to cause page ejection before the comment line is printed.

compilation time. The time during which a source program is translated from a high-level language to a machine language program.

compiler. A program that translates instructions written in a high-level programming language into machine language.

compiler directing statement. A statement that controls what the compiler does rather than what the user program does.

complex condition. A condition in which one or more logical operators (AND, OR or NOT) act upon one or more conditions. Complex conditions include negated

simple conditions, combined conditions, and negated combined conditions. See *conditional expression and simple condition.*

compound condition. A statement that tests two or more relational expressions. The result can be true or false.

computer name. A system name that identifies the computer upon which the program is to be compiled or run.

condition. An expression in a program for which a truth value can be determined at run time. Conditions include the simple conditions (relation condition, class condition, condition-name condition, switch-status condition, sign condition) and the complex conditions (negated simple conditions, combined conditions, negated combined conditions).

condition name. A name assigned to a specific value, set of values, or range of values, within the complete set of values that a conditional variable can possess. Or, it is the name assigned to a status of an IBM-defined switch.

condition-name condition. A condition that states that the value of a conditional variable is one of the set of values assigned to a condition-name associated with the conditional variable.

conditional expression. A simple condition or a complex condition specified in an IF, a PERFORM, or a SEARCH statement. See *complex condition* and *simple condition*

conditional statement. A statement that causes a condition to be evaluated to a value of either true or false and that controls program flow depending on this value.

conditional variable. A data item, one or more values of which has a condition-name assigned to it.

CONFIGURATION SECTION. A section of the Environment Division of the COBOL program. It describes the overall specifications of computers.

connective. A work or a punctuation character that associates a data name, paragraph name, condition name, or text name with its qualifier, links two or more operands in a series, or forms a conditional expression.

constant. A data item with a value that does not change. Contrast with *variable.*

control storage. Storage in the computer that contains the programs used to control input and output operations and the use of main storage. Contrast with *main storage.*

currency sign. The character $.

currency symbol. The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present, the currency sign is used. See *currency sign.*

current record. The record that is currently available to the program.

current record pointer. An internal mechanism that is used in sequential retrieval of the next record.

data clause. A clause that appears in a data description entry in the Data Division and that provides information describing a particular attribute of a data item.

data description entry. An entry in the Data Division that describes the characteristics of a data item.

data dictionary. A folder that contains field, format, and file definitions.

DATA DIVISION. One of the four main component parts of a COBOL program. The Data Division describes the files to be used in the program and the records contained within the files. It also describes any internal Working-Storage records that will be needed.

data hierarchy. The relationship between a group item or record and the group data items and elementary data items that make it up.

data item. A character or a set of consecutive characters (excluding literals in either case) defined as a unit of data by the COBOL program.

data name. A user-defined word that names a data item. When used in the general formats, *data name* represents a word that cannot be subscripted, indexed, nor qualified unless specifically permitted by the rules of that format. (See *identifier*) An index name is not a data name.

DDM. See *Distributed Data Management*

debugging line. A COBOL statement run only when the WITH DEBUGGING MODE clause is specified. Debugging lines can help determine the cause of an error.

debugging section. A Declaratives section that receives control when an identifier, a file name, or a procedure name is encountered in the Procedure Division.

declaratives. A set of one or more special-purpose sections, written at the beginning of the Procedure Division, that can be used for input/output error checking or debugging.

Division, that can be used for input/output error checking or debugging.

declarative sentence. A compiler-directing statement that specifies when a debugging procedure or an exception/error procedure is to be run.

delimiter. A character or a sequence of consecutive characters that marks the end of a unit of data and is not part of that unit of data.

descending key. The values by which data is ordered from the highest value to the lowest value of the key, in accordance with the rules for comparing data items.

descending key sequence. The arrangement of data in order from the highest value of the key field to the lowest value of the key field. Contrast with *ascending key sequence.*

Development Support Utility (DSU). An editor that can be invoked through either the DSU procedure or the corresponding Help screen.

digit. Any of the numerals from 0 through 9.

direct file. See *relative file.*

display format. Data that defines (or describes) a display.

display screen. The part of the display station on which information is displayed.

display station. A device that includes a keyboard from which an operator can send information to the system and a display screen on which an operator can see the information sent or receive information from the system.

Distributed Data Management (DDM). A feature of the System Support Program Product that allows an application program to work on files that reside on a remote system.

division. One of the four major parts in a COBOL program: Identification, Environment, Data, or Procedure.

division header. The reserved words and punctuation that indicate the beginning of one of the four divisions of a COBOL program.

DSU. See *Development Support Utility*

dynamic access. An access mode in which records can be read from or written to a file in a nonsequential order (see *random access*) and read from a file in a sequential order (see *sequential access*) during the scope of the same OPEN statement.

EBCDIC. See *extended binary-coded decimal interchange code.*

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC set.

editing character. A single character or a fixed 2-character combination used to format output.

elementary item. A data item that is described as not being logically subdivided.

entry. Any descriptive set of consecutive clauses ended by a period and written in the Identification, Environment, or Data Division of a COBOL source program.

ENVIRONMENT DIVISION. One of the four main component parts of a COBOL program. The Environment Division describes the computers upon which the source program is compiled and those on which the object program is executed; it also provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored.

exponent. A number, indicating to which power another number (the base) is to be raised.

EXTEND mode. An open mode in which records are added to the end of a sequential file.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 eight-bit characters.

external decimal item. See *zoned decimal item.*

figurative constant. A reserved word that represents a numeric or character value, or a string of repeated values. The word can be used instead of a literal to represent the value.

FILE-CONTROL. The name and header of an Environment Division paragraph in which the data files for a given source program are named and assigned to specific input/output devices.

file description entry. An entry in the File Section of the Data Division that contains information about the identification, the physical structure, and the record name of a file.

file name. The name used by a program to identify a file.

file organization. The permanent file structure established at the time a file is created.

FILE SECTION. A section of the Data Division, that contains descriptions of all externally stored data (or files) used in a program. Such information is given in one or more file description entries.

function-name. A name, defined by IBM, that identifies system logical units, system-supplied information, printer control characters, or program switches.

group item. A named set of consecutive elementary or group items.

hierarchy. A set of entries that includes all subordinate entries to the next equal- or higher-level number.

IDDU. See *Interactive Data Definition Utility.*

IDDU communications format. A feature of the Interactive Communications Feature, that allows data to be sent or received between your program and a remote program.

IDDU format file. A file that contains data definitions.

IDENTIFICATION DIVISION. One of the four main component parts of a COBOL program. The Identification Division identifies the source program and the object program and, in addition, may include such documentation as the author's name, the installation where written, and the date written.

identifier. A data name that is unique or is made unique by the correct combination of qualifiers, subscripts, or indexes.

ideographic. Pertaining to two-byte characters consisting of pictograms, symbolic characters, and other types of symbols.

imperative statement. A statement that specifies that an action is always to be taken. An imperative statement can consist of a sequence of imperative statements.

implementor name. A system name that identifies the external medium of a COBOL file and the name by which it is known to the system.

independent data item. A data item in the Working-Storage Section that has no relationship with other data items.

index. A computer storage position or register, the contents of which identify a particular element in a set of elements.

index data item. A data item in which the contents of an index can be saved.

index name. A user-defined word that names an index.

indexed data name. A data name followed by one or more index names enclosed in parentheses, which is used to reference an element or a set of elements in a table.

indexed file. A file in which the key and the position of each record is recorded in a separate portion of the file called an index. Contrast with *direct file* and *sequential file.*

indicator. An internal switch that communicates a condition between parts of a program or procedure.

input file. A file that is opened in the input mode.

input mode. An open mode in which records can be read from the file.

input-output file. A file that is opened in the I-O mode.

INPUT-OUTPUT SECTION. In the Environment Division, the section that names the files and external media needed by an object program. It also provides information required for the transmission and handling of data when an object program is run.

integer. A numeric data item or literal that does not include any character positions to the right of the decimal point. When the term integer appears in formats, integer must be an unsigned numeric literal and must be nonzero unless the rules for that format explicitly state otherwise.

Interactive Communications Feature (SSP-ICF). A feature of the System Support Program Product that allows a program to interactively communicate with another program or system.

Interactive Data Definition Utility (IDDU). The part of the System Support Program Product that you can use to define the characteristics of data and the contents of files.

INVALID KEY condition. A run-time condition in which the value of a key for an indexed or relative file incorrectly refers to the file.

I/O CONTROL. The name and the header for an Environment Division paragraph in which object program requirements for specific input/output techniques are specified. These techniques include rerun checkpoints, the sharing of same areas by several data files, and the use of a storage-resident cylinder index.

I/O mode. An open mode in which records can be read from, written to, or deleted from a file.

Kanji. The ideographic character set used by the Japanese to represent their native language.

Katakana. A native Japanese character set that is used primarily to write foreign words phonetically.

key. One or more characters used to identify the record and establish the record's order within an indexed file or a direct (relative) file.

key word. A reserved word that is required by the syntax of a COBOL statement or entry.

language name. A system name that specifies a particular programming language.

level indicator. Two alphabetic characters (FD or SD) that identify the type of file description entry.

level number. A numeric character (1 through 9) or a 2-character set (01 through 49, 66, 77, 88) that begins a data description entry and establishes its level in a data hierarchy. Level numbers 66, 77 and 88 identify special properties of a data description entry.

library name. A user-defined word that names a library.

LINKAGE SECTION. A section of the Data Division that describes data made available from another program.

literal. A symbol or a quantity in a source program that is itself data, rather than a reference to data.

logical operator. A reserved word that defines the logical connection between conditions or negates a condition: OR (logical connective--either or both), AND (logical connective--both), and NOT (logical negation).

logical order. The order in which records are sequentially read from a file. For sequential and relative files, the logical order corresponds to the physical order of the records in the file. For indexed files, the logical order is based on the order of the keys in the index portion of the file.

logical record. The most inclusive data item. The level number for a logical record is 01.

main program. The highest-level program involved in a run unit.

main storage. The part of the processing unit where programs are run. Contrast with *control storage.*

memory resident overlays. An option that makes a program request that its overlays remain in main storage.

merge file. The temporary file that contains all the records to be combined by a MERGE statement.

mnemonic name. A user-defined word associated with a function name in the Environment Division.

mode. A method of operation

MRO. See *memory resident overlays.*

MRT program. See *multiple requester terminal program.*

multiple requester terminal (MRT) program. A program that can process requests from more than one display station or SSP-ICF session at the same time using a single copy of the program. Contrast with *single requester terminal (SRT) program.*

name. A word that defines a COBOL operand. A name is composed of not more than 30 characters.

native character set. The default character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

native collating sequence. The default collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

negated combined condition. The NOT logical operator immediately followed by a combined condition in parentheses.

negated condition. A condition that is made opposite (either true or false), by the NOT logical operator.

negated simple condition. The NOT logical operator immediately followed by a simple condition.

nest. To incorporate a structure or structures of some kind into a structure of the same kind. For example, to nest one loop (the nested loop) within another loop (the nesting loop); to nest one subroutine (the nested subroutine) within another subroutine (the nesting subroutine).

Network Resource Directory (NRD). An area on disk that lists the files on remote systems that can be accessed using Distributed Data Management (DDM).

next executable statement. The statement to which control is transferred after the current statement runs.

noncontiguous item. A data item in the Working-Storage Section of the Data Division that bears no relationship with other data items.

noncontiguous key. A key that can be made up of three record fields. These fields may be separated by non-key fields.

nonnumeric items. A data item that is alphanumeric, alphabetic, or Boolean.

nonnumeric literal. See *literal.*

NRD. See *Network Resource Directory*

numeric. Pertaining to any of the digits 0 through 9.

numeric edited item. A numeric item whose PICTURE character string contains valid editing characters.

numeric item. An item whose contents must be numeric. If signed, the item can also contain a representation of an operation sign.

numeric literal. See *literal.*

OBJECT-COMPUTER. The name of an Environment Division paragraph in which the computer upon which the object program will be run is described.

object program. A set of instructions in machine-runnable form. The object program is produced by a compiler from a source program.

object-time subroutine. An internal system subroutine that the compiler sets up links to when link-editing. Object-time subroutines are stored in #COBLIB.

open mode. The state of a file after performing an OPEN statement for that file and before performing a CLOSE statement for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O, or EXTEND.

operand. The object of a verb or an operator; that is, an operand is the data or equipment governed or directed by a verb or operator.

optional word. A reserved word included in a statement format that is intended to make that statement easier to read.

output file. A file that is opened in either the output mode or the extend mode.

output mode. An open mode in which records can be written to the file.

OUTPUT PROCEDURE. A procedure that provides special processing of records when they are returned from the sort or merge function.

overflow condition. A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

overlay. A program segment that is loaded into main storage and replaces all or part of a previously loaded program segment.

overlay structure. A graphic representation showing the relationship of segments of an overlay program and how the segments are arranged to use the same main storage area at different times.

packed decimal format. A format in which each byte (except the rightmost byte) within a field represents two numeric digits. The rightmost byte contains one digit and the sign. For example, the decimal value +123 is represented as 0001 0010 0011 1111. Contrast with *zoned decimal format.*

pad. To fill unused positions in a field with dummy data, usually zeros or blanks.

paragraph. In the Procedure Division, a paragraph name followed by a period and a space and by zero, one, or more sentences. In the Identification and Environment divisions, a paragraph header followed by zero, one, or more entries.

paragraph header. A reserved word, followed by a period and a space, that indicates the beginning of a paragraph in the identification and Environment Divisions.

paragraph name. A user-defined word that identifies and begins a paragraph in the Procedure Division.

parameter. A variable or a literal that is used to pass data values between calling and called programs.

phrase. An ordered set of one or more consecutive COBOL character strings that forms part of a clause or a Procedure Division statement.

physical record. A unit of data that is moved into or out of the computer.

procedure. One or more successive paragraphs or sections within the Procedure Division, which directs the computer to perform some action or series of actions.

PROCEDURE DIVISION. One of the four main component parts of a COBOL program. The Procedure Division contains instructions for solving a problem. The Procedure Division may contain imperative statements, conditional statements, paragraphs, procedures and sections.

procedure name. A paragraph name or a section name in the Procedure Division.

program name. A user-defined word that identifies a COBOL source program.

pseudo-text. A sequence of character strings or separators bounded by, but not including, pseudo-text delimiters. Pseudo-text is used in the COPY REPLACING statement for replacing text strings.

pseudo-text delimiter. Two contiguous equal signs (= =) used to delimit pseudo-text.

qualified name. A name that has been made unique by the addition of one or more qualifiers.

qualifier. A name used to uniquely identify another name.

random access. An access mode in which records can be read from, written to, or removed from a file in any order.

record. A collection of fields that is treated as a unit.

record area. A storage area reserved for processing the record described in a record description entry in the File Section.

record description entry. The total set of data description entries associated with a particular record.

record name. A data name for a record described in a record description entry.

relation character. One of the characters that expresses a relationship between two operands: = (equal to), > (greater than), < (less than).

relation condition. A condition that relates two arithmetic expressions and/or data items.

relational operator. A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used to express a relation condition.

relative file. Same as *direct file*.

reserved word. A word used in a source program to describe an action to be taken by the program or by the compiler, respectively. It must not appear in the program as a user-defined name or a system name.

routine. A set of statements in a program that causes the system to perform an operation or a series of related operations.

run. To cause a program, utility, or other machine function to be performed.

run unit. A set of one or more object programs that function as a unit at execution time to provide a problem solution.

section. A section header followed by a set of zero, one, or more paragraphs or entries, called a section body. Each section consists of the section header and the related section body.

section header. A combination of words, followed by a period and a space, that indicates the beginning of a section in the Environment, Data, or Procedure Division.

section name. A user-defined word that names a section in the Procedure Division.

sector. An area on a disk track or a diskette track reserved to record information.

sentence. A sequence of one or more statements; the last statement ends with a period followed by a space.

separator. A punctuation character used to delimit character-strings.

sequential access. An access mode in which records are read from, written to, or removed from a file based on the logical order of the records in the file.

sequential file. A file in which records occur in the order in which they were entered. Contrast with direct file and indexed file.

sequential processing. The processing of records in the order in which they exist in a file.

serial search. A search in which the members of a set are consecutively examined, beginning with the first member and ending with the last member.

SEU. See *source entry utility*.

shift-in control character. A character (hexadecimal 'OF') that indicates the end of a string of ideographic characters. Contrast with shift-out control character.

shift-out control character. A character (hexacecimal 'OE') that indicates the start of a string of ideographic characters. Contrast with shift-in control character.

sign condition. A condition that states that the value of a data item is less than, equal to, or greater than zero.

simple condition. Any single condition chosen from the set: relation condition; class condition; condition-name condition; switch-status condition; sign condition. See *complex condition* and *conditional expression*.

single requester terminal (SRT) program. A program that can process requests from only one display station or SSP-ICF session from each copy of the program. Contrast with *multiple requester terminal program*.

SOURCE-COMPUTER. The name of an Environment Division paragraph describing the computer upon which the source program will be compiled.

source entry utility (SEU). The part of the Utilities Program Product used by the operator to enter and update source and procedure members.

source program. A set of instructions that are written in a programming language and that must be translated to machine language before the program can be run.

special character. A COBOL character that is neither numeric nor alphabetic. Special characters in COBOL include the space (), and the period (.), as well as the following: + - * / = $, " ( ) ; < >.

special registers. Compiler-generated data items used to store information produced by specific COBOL features (for example, the DEBUG-ITEM special register).

special-character word. A reserved word that is an arithmetic operator or a relation character.

SPECIAL-NAMES. The name of an Environment Division paragraph and the paragraph itself in which names supplied by IBM are related to mnemonic names specified by the programmer. In addition, this paragraph can be used to exchange the functions of the comma and the period or to specify a substitution character for the currency sign in the PICTURE string.

SRT program. See *single requester terminal program.*

SSP. See *System Support Program Product.*

SSP-ICF. See *Interactive Communications Feature.*

standard data format. The format in which data is described as to how it appears when it is printed rather than how the data is stored in the computer.

statement. A syntactically valid combination of words and symbols, beginning with a verb, that is written in the Procedure Division.

subprogram. A called program.

subscript. An integer or variable whose value refers to a particular element in a table or an array.

subscripted data name. A data name that has been made unique through the use of a subscript.

switch-status condition. A condition that states that a switch is currently on or off.

SYSIN. An SSP routine to support input from the system device.

SYSLOG. AN SSP routine to support input and output (logoff messages) to a device.

SYSTEM-CONSOLE. A COBOL function name associated with the operator's keyboard/display.

system-defined special format. A feature of the Interactive Communications Feature that begins with two dollar signs ($$), and provides SSP-ICF functions for COBOL users.

system name. An IBM-defined name that has a predefined meaning to the COBOL compiler. System names include computer names, language names, device names, and function names.

System Support Program Product (SSP). A group of licensed programs that manage the running of other programs and the operation of associated devices, such as the display station and printer. The SSP also contains utility programs that perform common tasks, such as copying information from diskette to disk.

S/I. See *shift-in control character.*

S/O. See *shift-out control character.*

table. A set of logically consecutive data items that are defined in the Data Division by means of the OCCURS clause.

test condition. A statement that, when taken as a whole, may be either true or false, depending on the circumstances existing at the time the expression is evaluated.

text name. A user-defined word that identifies library text.

text word. Any character string or separator, except the space, in copied COBOL source or in pseudo-text.

TRANSACTION file. An input/output file used to communicate with display stations and SSP-ICF sessions.

unary operator. A plus sign (+) or a minus sign (-), which precedes a variable or a left parenthesis in an arithmetic expression and which has the effect of multiplying the expression by +1 or -1, respectively.

UPSI switch. See *user program status indicator switch.*

user-defined word. A word, required by a clause or a statement that must be supplied by the user in a clause or statement.

user program status indicator (UPSI) switch. One of a set of eight switches that can be set by and passed between application programs and procedures.

user-defined word. A word, required by a clause or a statement that must be supplied by the user in a clause or statement.

user program status indicator (UPSI) switch. One of a set of eight switches that can be set by and passed between application programs and procedures.

variable. A name used to represent a data item whose value can change while the program is running. Contrast with *constant*.

verb. A COBOL reserved word that expresses an action to be taken by a COBOL compiler or an object program.

word. A character string of not more the 30 characters that forms a user-defined word, a system name, or a reserved word.

work station. A device that lets people transmit information to or receive information from a computer; for example, a display station or printer.

WORKING-STORAGE SECTION. A section name (and the section itself) in the Data Division. The section describes records and noncontiguous data items that are not part of external files but are developed and processed internally. It also defines data items whose values are assigned in the source program.

zoned decimal format. A format for representing numbers in which the digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the rightmost byte; bits 0 through 3 of all other bytes contain 1s (hex F). For example, in zoned decimal format, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. Contrast with *packed decimal format*.

zoned decimal item. A numeric data item that is represented internally in zoned decimal format.

Index

B

C

D

E

F

G

H

I

M

N

O

P

Q

R

S

T

U

V

W

Z

Numerics

What Is Your Opinion of This Manual?

Your comments can help us produce better manuals. Please take a few minutes to evaluate this manual as soon as you become familiar with it. Circle Y (Yes) or N (No) for each question that applies. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

FINDING INFORMATION

Y N Is the table of contents helpful?
What would make it more helpful?

Y N Is the index complete?
List specific terms that are missing.

Y N Are the chapter titles and other headings meaningful?
What would make them more meaningful?

Y N Is information organized appropriately?
What would improve the organization?

Y N Does the manual refer you to the appropriate places for more information?
List specific references that are wrong or missing.

UNDERSTANDING INFORMATION

Y N Is the purpose of this manual clear?
What would make it clearer?

Y N Is the information explained clearly?
Which topics are unclear?

Y N Are the examples clear?
Which examples are unclear?

Y N Are examples provided where they are needed?
Where should examples be added or deleted?

Y N Are terms defined clearly?
Which terms are unclear?

Y N Are terms used consistently?
Which terms are inconsistent?

Y N Are too many abbreviations and acronyms used?
Which ones are not understandable?

Y N Are the illustrations clear?
Which ones are unclear?

USING INFORMATION

Y N Does the information apply to your situation?
Which topics do not apply?

Y N Is the information accurate?
What information is inaccurate?

Y N Is the information complete?
What information is missing?

Y N Is only necessary information included?
What information is unnecessary?

Y N Are the examples useful models?
What would make them more useful?

Y N Is the format of the manual (shape, size, color) effective?
What would make the format more effective?

OTHER COMMENTS

Use the space below for any other opinions about this manual or about the entire set of manuals for this system.

YOUR BACKGROUND

What is your job title?

What is your primary job responsibility?

How many years have you used computers?

Which programming languages do you use?

How many times per month do you use this manual?

Your name
Company name
Street address
City, State, ZIP

No postage necessary if mailed in the U.S.A.

Cut Along Line

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N. Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Development Laboratory
Information Development, Department 532
Rochester, Minnesota 55901

Fold and tape | Please do not staple | Fold and tape

IBM

International Business Machines Corporation

IBM System/36: Programming with COBOL File No. S36-24 Printed in U.S.A. SC21-9007-3

READER'S COMMENT FORM

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your nearest IBM branch office. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

☐ If your comment does not need a reply (for example, pointing out a typing error) check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

☐ If you would like a reply, check this box. Be sure to print your name and address below.

Page number(s): Comment(s):

Please contact your nearest IBM branch office to request additional publications.

Name _____

Company or Organization _____

Address _____

_____

City State Zip Code

No postage necessary if mailed in the U.S.A.

Cut Along Line

Fold and tape Please do not staple Fold and tape

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N. Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Development Laboratory
Information Development, Department 532
Rochester, Minnesota 55901

Fold and tape Please do not staple Fold and tape

IBM

International Business Machines Corporation

IBM System/36: Programming with COBOL File No. S36-24 Printed in U.S.A. SC21-9007-3

International Business Machines Corporation

Programming with COBOL

Contents

File Number
S36-24

Order Number
SC21-9007-3

Part Number
59X3992

Printed in U.S.A.