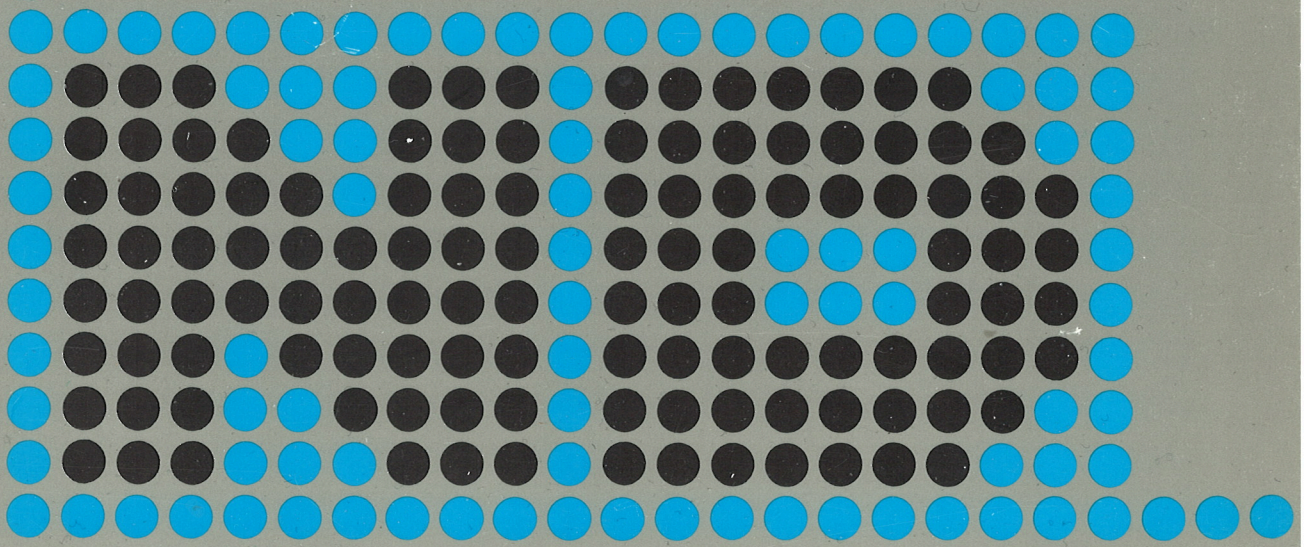
Norsk Data

ND COBOL
Reference Manual

ND-60.144.02
Revision B



ND COBOL
Reference Manual

ND-60.144.02
Revision B

NOTICE

The information in this document is subject to change without notice. Norsk Data A.S assumes no responsibility for any errors that may appear in this document. Norsk Data A.S assumes no responsibility for the use or reliability of its software on equipment that is not furnished or supported by Norsk Data A.S.

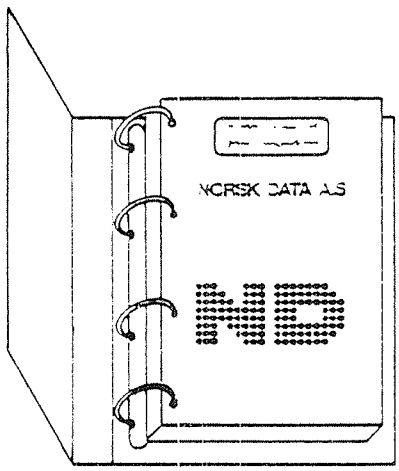
The information described in this document is protected by copyright. It may not be photocopied, reproduced or translated without the prior consent of Norsk Data A.S.

Copyright (C) 1984 by Norsk Data A.S

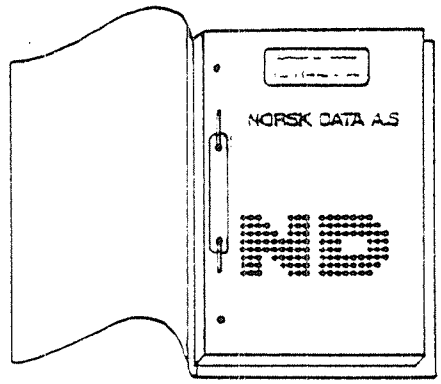
This manual is in loose leaf form for ease of updating. Old pages may be removed and new pages easily inserted if the manual is revised.

The loose leaf form also allows you to place the manual in a ring binder (A) for greater protection and convenience of use. Ring binders with 4 rings corresponding to the holes in the manual may be ordered in two widths, 30 mm and 40 mm. Use the order form below.

The manual may also be placed in a plastic cover (B). This cover is more suitable for manuals of less than 100 pages than for large manuals. Plastic covers may also be ordered below.



A Ring Binder



B Plastic Cover

Please send your order to the local ND office or (in Norway) to:

Documentation Department
Norsk Data A.S
P.O. Box 4, Lindeberg gård
Oslo 10

ORDER FORM

I would like to order

..... Ring Binders, 30 mm, at nkr 20,- per binder

..... Ring Binders, 40 mm, at nkr 25,- per binder

..... Plastic Covers at nkr 10,- per cover

Name .....
Company .....
Address .....
.....
City .....

PRINTING RECORD

Printing	Notes
01/81	Version 01
03/82	Revision A:
	The following pages have been revised:
	x, xi, xiii,
	1–1, 2–10, 4–1, 4–2, 4–11, 5–30, 6–10, 6–12, 6–22, 6–25, 6–25a, 6–27, 6–80,
	6–90, A–11, A–12
	Index pages 1 through 12.
	The following pages have been added:
	H–1, H–2.
08/82	Version 02
07/83	Revision A:
	The following pages have been revised:
	v, vi, xv,
	1–4, 2–8, 5–5, 5–9, 5–22, 5–27, 5–31, 5-32, 5–34, 6–23, 6–24, 6–25, 6–26,
	6–27, 6–28, 6–29, 6–30, 6–31, 6–32, 6–33, 6–34, 6–35, 6–36, 6–44, 6–44a,
	6–44b, 6–44c, 6–44d, 6–44e, 6–53, 6–55, 6–56, 6–70, 6–101, 6–102, 6–103,
	6–104, 6–105, 6–107, 6–110, 9–4, 9–5,
	A–1, A–2, A–3, A–4, A–5, A–6, A–7, A–8, A–9, A–10, A–11, A–12, A–13, A–14,
	A–15, A–16, A–17, A–18, A–19, A–20, A–21, A–22, A–23, A–24, A–25, A–26,
	A–27, A–28, A–29, A–30, A–31, D–1, D–2, E–1, F–1, F–2, F–3, H–2,
	–1–, –2–, –3–, –4–, –5–, –6–, –7–, –8–, –9–, –10–, –11–, –12–
06/84	Revision B:
	The following pages have been revised:
	iv, v, vii, xv
	1-2, 1-3, 1-5, 6-28, 6-33, 6-34, 6-35, 6-36, 6-37 to 6-44g, 6-54
	A-1, A-1a, A-1b, A-9, A-12, A-15, A-16, D-1,
	The following pages have been added:
	J-1, J-2, J-3, J-4, J-5, J-6

ND COBOL Reference Manual
Publ.No ND-60.144.02 Rev. B
June 1984

NORSK DATA A.S
P.O.Box 25, Bogerud
0621 Oslo 6, Norway

Manuals can be updated in two ways, new versions and revisions. New versions consist of a complete new manual which replaces the old manual. New versions incorporate all revisions since the previous version. Revisions consist of one or more single pages to be merged into the manual by the user, each revised page being listed on the new printing record sent out with the revision. The old printing record should be replaced by the new one.

New versions and revisions are announced in the ND Bulletin and can be ordered as described below.

The reader's comments form at the back of this manual can be used both to report errors in the manual and to give an evaluation of the manual. Both detailed and general comments are welcome.

These forms, together with all types of inquiry and requests for documentation should be sent to the local ND office or (in Norway) to:

Documentation Department
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

PREFACE

THE PRODUCT

COBOL (COmmon Business Oriented Language) is a programming language, based on English, which was developed for use in commercial data processing. The original COBOL specification resulted from the work of the CODASYL (Conference on Data Systems Languages) committee in the U. S. A. in 1959. ND COBOL is based on American National Standard X3.23 — 1974. ND COBOL is COBOL for both ND-100 and the ND-500. Differences, where they occur, are described in the text.

This manual describes ND COBOL, ND-10176, version F (ND-100) and ND COBOL, ND-10177, version F (ND-500).

THE READER

The manual is intended for the programmer using ND COBOL who requires a detailed and formal explanation of the product as well as an account of the features and facilities available to the user.

PREREQUISITE KNOWLEDGE

A basic knowledge of data processing techniques is necessary for the reader and some familiarity with COBOL would be helpful. The reader should also have some knowledge of the SINTRAN III/VS operating system.

HOW TO USE THE MANUAL

The description is given in the order in which the Divisions and Sections appear in the written programs.

The manual is intended for reference purposes and is organized as follows:

Part I of the manual describes ND COBOL in general terms and gives specific rules for the writing of COBOL source programs. There is a chapter for each COBOL division. Part II contains an account of each 'other feature' or special topic requiring a section of its own. Supplemental information is given in Appendixes at the end.

CHANGES FROM PREVIOUS VERSION

Version F features Screen Handling, Multi-user relative and indexed I-O, the DO statement, extensions to the IF statement, IMPORT and EXPORT clauses. Section 1.2.1 lists the extensions to the standard in more detail.

ACKNOWLEDGEMENT

Any organization interested in reproducing the COBOL standard and specifications in whole or in part, using ideas from this document as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce the following acknowledgement paragraphs in their entirety as part of the preface to any such publication. (Any organization using a short passage from this document, such as in a bookreview, is requested to mention "COBOL" in acknowledgement of the source, but need not quote the acknowledgement):

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributer or by the CODASYL Programming Language Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

The authors and copyright holders of the copyrighted material used herein:

> FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the UNIVAC® I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27 A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

FORMAT NOTATION

Basic formats are prescribed in the manual for the elements of the COBOL language. The notation described here is used to define unambiguously for the programmer how the clauses and statements of COBOL should be written.

RESERVED WORDS

COBOL has a specified list of words for use in source programs which have preassigned meanings and cannot appear in programs as user-defined words or system names. A complete list of the reserved words can be found in Appendix D.

Reserved words may be divided into two categories:

Key Words

These are required by the syntax of the format. They are always in upper case and underlined.

Optional Words

As their name implies, they may be included or omitted without changing the syntax of the program. They appear in upper case but are not underlined.

Words printed in lower case letters represent information to be supplied by the programmer. All such words are defined within this manual.

The general format is also defined by the use of the following symbols:

Braces ({ }) These enclose vertically stacked items and indicate that one of the enclosed items must appear.

Brackets ([ ]) Square brackets are used to show that the enclosed item is optional, depending on the requirements of the program.

The *Ellipsis* (...) These dots specify that the immediately preceding unit may occur any number of times in succession at the user's option.

The *arithmetic* and *logical operators* (+, −, >, <, =) When they appear in formats they are required items even though they are not underlined.

Any other punctuation or special characters which appear in general formats indicate the actual occurrence of these characters and are required by the syntax.

TABLE OF CONTENTS

+ + +

PART I STANDARD COBOL

PART II OTHER FEATURES

1 INTRODUCTION

1.1 ND COBOL

ND COBOL is a standard high level language implemented as a conventional compiler and run-time library system operating under SINTRAN III/VS operating system.

ND COBOL is based upon American National Standard X3-23-1974. Elements of the COBOL language are allocated to 12 different functional processing "modules".

Each module of the COBOL Standard has two "levels" — level 1 represents a subset of the full set of capabilities and features contained in level 2.

In order for a given system to be called COBOL, it must provide at least level 1 of the Nucleus, Table Handling and Sequential I-O modules.

The following summary specifies the contents of ND COBOL with respect to the Standard:

Module	Features Available in ND COBOL
Nucleus	All of level 1 and level 2 except: level 66 the RENAMES clause the switch-status condition the ENTER statement. Additional features are: USAGE is COMPUTATIONAL —1 COMPUTATIONAL —2 COMPUTATIONAL —3 ACCEPT FROM CPU-TIME.
Sequential I-O	All of level 1 and level 2 except the RERUN the LINAGE and CODE SET clauses with the addition of the RECORDING MODE clause.

Indexed I-O and Relative I-O	All of level 1 and level 2 except the RERUN and the SAME RECORD AREA clauses with the addition of: the RECORDING MODE clause.
Table Handling	All of level 1 and level 2.
Sort/Merge	All of level 1 and level 2 except: the SAME AREA clause.
Inter-Program Communication	All of level 1 and level 2 except: the CANCEL statement.
Debugging	Conditional compilation: lines with 'D in column 7' are bypassed unless WITH DEBUGGING MODE.

1.2 OPERATIONAL REQUIREMENTS

The ND COBOL compiler occupies 128 K words of the program area.

The compiler may execute as a reentrant subsystem under the SINTRAN III/VS operating system, when only the necessary 1 K pages are brought into the memory as needed. In this way, several active users may share a common code.

A system scratch file for the active terminal will be used to store compiler information.

The source program is accepted in any media supported by the ND File System, and may be entered and modified using an interactive editor. Once entered, source files are stored on disk, floppy diskette or magnetic tape and can be compiled by using simple compiler commands.

On the ND-10 a special microprogram is required.

The result of a compilation is:

A) A source listing including compiler assigned line numbers, source file name, object file name, date and time.

B) In the event of any source program errors (or warnings) diagnostic messages will appear following the source listing. These messages have the format:

   — Line number (5 digits)
   — English message text
   — (Optional) Further relevant data

C) An object program in library relocatable form (BRF on the ND-100 or NRF on the ND-500) can be used by the ND Relocating Loader for the ND-100 or the ND-500 Linkage Loader for the ND-500, to prepare the object program in a form which is ready for execution.

1.2.1 Extensions

For version E, new functions have been introduced as follows:

The BLANK statement.
The ACCEPT-ERROR statement.
The ACCEPT and DISPLAY statements with Screen-Handling options.
The OPEN statement with the MULTI-USER MODE, IMMEDIATE-WRITE and MANUAL UNLOCK options.
The READ statement with LOCK.
The UNLOCK statement.
The DO statement.
The IF statement with the THEN, ELSE-IF and END-IF clauses.
The IMPORT and EXPORT clauses for inter-program communication.

1.2.2 Known Restrictions

For the time being the following restrictions are applicable:

A) A 77/01 item must not be greater than 32767 bytes.

1.3 HOW TO USE THE SYSTEM

A complete example of the compilation, loading and execution of a simple program with some of the features of ND COBOL is shown in Appendix J.

1.3.1 How to Compile a COBOL Program

The COBOL compiler may be recovered from the operating system by entering:

```
@COBOL                  (ND-100)
or
@ND COBOL-500           (ND-500)
```

When the compiler has printed * (asterisk) on the terminal, it is ready to accept commands from the user.

All commands may be abbreviated as in SINTRAN III/VS. A^C and Q^C (control A, control Q) may be applied to command input.

The command to compile is:

```
COMPILE  source file  list file  object file
```

The source file is your symbolic program containing COBOL statements. A listing of the program is written on the list file while the object program in binary relocatable format is written onto the object file.

The files must be specified by their names and these names must be delimited by at least one space or comma. The default source file type is :SYMB. The list file type is :SYMB and the object file type is :BRF on the ND-100 or :NRF on the ND-500. (Scratch file 100 cannot be used as the object file.)

If the source input file is not a disk file, a line containing *END (from column 1) must close the source file.

Example:

```
@COBOL
*COMP SOURCE LINE-PRIN "OBJ"
```

Note that in this example the object file (OBJ) is specified inside quotes since it is to be a new file.

If no diagnostics appear, the compiler has accepted all the statements as syntactially and semantically correct. The object version may now be loaded by the ND Relocating Loader. (ND-100) or the ND-500 Linkage-Loader for the ND-500.

Sample Compilation

Let us assume that you have produced a program using an editor and it is stored under the name EX-001 with type SYMB (suitable as input to ND COBOL) as follows:

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 09.45.16   DATE: 30.09.80
    SOURCE FILE: X-001
    OBJECT FILE: "X-001"

      1        ********************************************************************
      2        *    THIS IS A SAMPLE EXAMPLE THAT CAN SERVE TO FAMILIARIZE        *
      3        *    US WITH THE RESULT OF A COBOL COMPILATION.                    *
      4        *                                                                  *
      5        *    THE PROGRAM COUNTS THE NUMBER OF RECORDS ON THE FILE          *
      6        *    "ABC:DATA".                                                   *
      7        ********************************************************************
      8         IDENTIFICATION DIVISION.
      9         PROGRAM-ID.            X-001.
     10         AUTHOR.                NORSK DATA A/S
     11                                NORWAY.
     12         DATE-WRITTEN.          OCTOBER 1980.
     13
     14         ENVIRONMENT DIVISION.
     15         CONFIGURATION SECTION.
     16         SOURCE-COMPUTER.       NORD-100.
     17         OBJECT-COMPUTER.       NORD-100.
     18         SPECIAL-NAMES.         DECIMAL-POINT IS COMMA.
     19         INPUT-OUTPUT SECTION.
     20         FILE-CONTROL.
     21             SELECT L-FILE ASSIGN TO "ABC:DATA".
     22
     23         DATA DIVISION.
     24         FILE SECTION.
     25         FD  L-FILE
     26             BLOCK CONTAINS 1 RECORDS
     27             RECORD CONTAINS 10000 CHARACTERS.
     28         01  L-RECORD           PIC X(10000).
     29         WORKING-STORAGE SECTION.
     30         01  NUMBER-OF-RECORDS PIC 9(10) VALUE 0.
     31
     32         PROCEDURE DIVISION.
     33         1000.
     34             OPEN INPUT L-FILE.
     35         2000.
     36             READ L-FILE        AT END GO TO 9000.
     37             ADD 1              TO NUMBER-OF-RECORDS.
     38             GO TO 2000.
     39         9000.
     40             DISPLAY "NUMBER OF RECORDS IN THE FILE IS "
     41                      NUMBER-OF-RECORDS.
     42             CLOSE L-FILE.
     43             STOP-RUN.

     43 E - SYNTAX ERROR (RESUMPTION AT NEXT PARAGRAPH/VERB): STOP-RUN

***       1 ERROR MESSAGE ***

*EXIT

@
```

Figure 1.1.

Note the following in the compilation listing:

1. The page heading contains date, time, source file name (EX-001) and object file name.

2. The source line (first 80 positions only) is listed along with compiler assigned line numbers.

3. Diagnostic or warning messages, if any, appear after the source program listing.

The error in the example:

> 43 E — SYNTAX ERROR (RESUMPTION AT NEXT PARAGRAPH/VERB): STOP-RUN

produces the relevant line number (43) together with explanatory text and the element itself which caused the error.

After successful compilation, the next step will be to link-edit using the Relocating Loader. Finally, the resultant program module will be executed. These operations are now discussed.

1.3.2 How to Load and Execute a COBOL Program

On the ND-100, the ND Relocating Loader may be recovered from the operating system by entering:

@NRL

When the loader has displayed an asterisk (*) on the terminal, it is ready to accept commands from the user.

Your program(s) may be loaded into a program-file instead of into main memory if you use the command:

*PROG-FILE (file name)

The PROG-FILE must be the first command given after the loader recovery.

Loader input is obtained from one or more files/library files. The loader is initiated by the command:

*LOAD file name [file name] ...

Each of the files specified will be loaded until end-of-file is detected, then control is returned to the user at the terminal by the prompt * (asterisk) and the loader is then ready to accept another command.

To obtain the entry point addresses of the loaded program, use the command:

*ENTRIES-DEFINED (file name)

The octal addresses which appear on this map denote the last reference address.

There should be no undefined entry points remaining. If your program is loaded into main memory it may be started by the command:

*RUN

When the program has been executed, control is transferred to the operating system and @ (at) is displayed.

Note that the RUN command can only be used in one-bank mode and if no PROG-FILE or IMAGE-FILE is specified.

If you wish to leave the loader and enter the operating system you may simply enter:

*EXIT

You may restart the loader by using the system command:

@CONTINUE.

Message:

If the message:

LOADER TABLE OVERFLOW

is given it means that there is no more room for entries. The table length may be expanded through the command:

*SIZE number of entries (octal)

For additional documentation relating to use of NRL, refer to its complete documentation ND-60.066.

On the ND-500, the ND LINKAGE LOADER (known as the NLL) may be called by the command:

@ND-500 LINKAGE-LOADER

when the NLL types NLL:, it is ready to accept commands from the user.

The NLL will create a program ready for the execution. On the ND-500 a program is termed a *domain.* A domain is named before anything is loaded to it by the command:

NLL:SET-DOMAIN <domain-name>

Then a segment is opened by the command:

NLL:OPEN-SEGMENT <segment-name>[,<attributes>]

And now programs and library files can be loaded into the segment by the command:

NLL:LOAD-SEGMENT <file>[,<file>]...

To obtain the entry point addresses of the loaded program, use the command:

NLL:LIST-ENTRIES-DEFINED

and undefined entries by:

NLL:LIST-ENTRIES-UNDEFINED

If the user wishes to leave the NLL he can type the command:

NLL:EXIT

and he will return to SINTRAN.

For additional documentation relating to the use of the NLL, the user should refer to its complete documentation in the manual: ND-500 LOADER/MONITOR, ND-60.136.03.

1.3.3 Compiling, Loading and Execution

Using the same source program and commands as in the previous example, the following commands can be used:

(all lines start in column 1)

ON THE ND-100

```
@COBOL
COMP PROGRAM, TERM, PROGRAM
EXIT
@NRL
PROG-FILE PROGRAM
LOAD PROGRAM, COBOL-1 BANK
EXIT
@PROGRAM
```

ON THE ND-500

```
@ND COBOL
COMPILE PROGRAM, TERM, PROGRAM
EXIT
@ND LINKAGE LOADER
SET-DOMAIN PROGRAM
OPEN-SEGMENT PROGRAM
LOAD-SEGMENT PROGRAM, COBOL-LIB
EXIT
@ND PROGRAM
```

2 LANGUAGE CONCEPTS

2.1 STRUCTURE OF COBOL

Every COBOL program is divided into four divisions. Each must be placed in its proper sequence and begin with a division header.

2.1.1 The COBOL Divisions

The four divisions of a COBOL source program and their functions are:

2.1.1.1 Identification Division

This names the program and, optionally, documents the compilation date, etc.

2.1.1.2 Environment Division

This describes the computer(s) and equipment to be used by the program. It also includes a description of the relationship between the files containing data and the input-output devices.

2.1.1.3 Data Division

This defines the names and characteristics of all the data to be processed by the program.

2.1.1.4 Procedure Division

This consists of executable statements that direct the processing of data at execution time.

2.1.2 Structure within the Divisions — Clauses and Statements

A *clause* specifies the attributes of an *entry* which, containing a series of clauses ending with a period, can appear in each division except the procedure division.

A *statement*, written in the procedure division, specifies an action to be taken by the object program. A series of statements, ending with a period, is defined as a *sentence*.

Every clause or statement in the program may be further subdivided into units called phrases or options. A *phrase* is an ordered set of one or more COBOL character strings forming a part of a clause or statement. An *option* is a phrase in which the programmer can choose between alternative wordings, according to the meanings he wishes the phrase to possess.

Clauses, entries, statements and sentences may be combined into *paragraphs* and *sections* which each define a larger part of the problem program. A section may itself contain paragraphs.

2.2 STRUCTURE OF THE LANGUAGE

2.2.1 COBOL Character Set

The most basic and indivisible unit of the language is the character. The set of characters used to form COBOL character strings and separators is given below.

The complete COBOL character set consists of the 52 following characters:

Character:	Meaning:
0, 1, ..., 9	digit
A, B, ..., Z	letter
	space (blank)
+	plus sign
—	minus sign (hyphen)
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	dollar sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
"	quotation mark (double)
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol
'	apostrophe (single quotation mark)

Note that a reference to 'characters' throughout this manual wil be to a subset of the above list, i.e., the list not including 'separators" (defined in Section 2.2.3.4).

2.2.2 Character-Strings

A character-string is a character or sequence of contiguous characters which form a COBOL word, a literal, a PICTURE character-string or a comment entry. A character-string is delimited by separators.

2.2.3 COBOL Words

A COBOL word can be a user defined word, a system word or a reserved word. Its maximum length is 30 characters. System words and reserved words are defined as follows.

2.2.3.1 User Defined Words

These are COBOL words supplied by the programmer. Characters valid in a user defined word are:

A through Z
0 through 9
– (hyphen)

The hyphen may not be the first or last character. A list of the sets of user defined words together with their formation rules is given below.

User Defined Word Set:	Characteristics:
condition name data name record name file name index name mnemonic name	Must contain at least one alphabetic character. Within each set the name must be unique. (It can be made unique by qualification if the format rules for the set permit.)
library name program name routine name	The above rules apply.
paragraph name section name	May be all numeric, otherwise rules in paragraph 1 apply.

The function of each user defined word in any clause or statement will be found under the description for that clause or statement.

The function of each *system name* (Norsk Data defined names for communication with the operating system) will be found in the Glossary.

2.2.3.2 Reserved Words

Reserved words may be divided into the following categories:

1. Key words
2. Optional words
3. Connectives
4. Special registers
5. Figurative constants
6. Special character words

A *reserved word* is a COBOL word having a fixed meaning and it must not be used as a user defined word or system name. A list is given in Appendix D.

KEY WORDS

A *key word* is required when the format in which it appears is used in a source program. Within each format, such words are upper case and underlined.

Key words are of three types:

1. Verbs such as ADD, READ and MOVE.

2. Required words, which appear in statement and entry formats.

3. Words which have a specific functional meaning such as NEGATIVE, SECTION, etc.

OPTIONAL WORDS

Within each format, uppercase words that are not underlined are called *optional words* and may appear at the user's option. The presence or absence of an optional word does not alter the semantics of the COBOL program in which it appears.

CONNECTIVES

These are:

1. Qualifier connectives that are used to associate a data name, a condition name, a text name or a paragraph name with its qualifier: OF, IN.

2. Series connectives that link two or more consecutive operations : (separator comma) or ; (separator semicolon).

3. Logical connectives that are used in the formation of conditions: AND, OR.

SPECIAL REGISTERS

Each compiler generated storage area whose primary function is to store information produced by one of the specific COBOL features, is a special register.

Examples:

DATE, DAY, TIME (see ACCEPT statement in the Procedure Division).

FIGURATIVE CONSTANTS

Certain reserved words are used to name and reference certain constant values which will be generated by the compiler when these words are used. Known as *figurative constants* they must not be bounded by quotation marks. Singular and plural forms may be used interchangeably.

The reserved words and the *figurative constant values* they generate are listed below.

ZERO, ZEROS ZEROES	Represents the value '0' or one or more of the characters '0', depending on context.
SPACE, SPACES	Represents one or more of the character space from the computer's character set.
HIGH-VALUE, HIGH-VALUES	Represents one or more of the characters that has the highest ordinal position in the program collating sequence.
LOW-VALUE, LOW-VALUES	Represents one or more of the characters that has the lowest ordinal position in the program collating sequence.
QUOTE, QUOTES	Represents one or more of the characters '"'. The word QUOTE or QUOTES cannot be used in place of a quotation mark in a source program to bound a nonnumeric literal. Thus, QUOTE ABD QUOTE is incorrect as a way of stating the nonnumeric literal "ABD".
ALL literal	Represents one or more of the string of characters comprising the literal. The literal must be either a nonnumeric literal of one character length or a figurative constant other than ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readability only.

When a figurative constant represents a string of one or more characters, the length of the string is determined by the compiler from context according to the following rules:

1. When a figurative constant is associated with another data item (e.g., is moved to or compared with another item) the string of characters composing the figurative constant is repeated character by character on the right until the size of the resultant string in characters is equal to that of the associated data item. This is done prior to and independent of any application of a JUSTIFIED clause associated with the data item.

2. When a figurative constant is not associated with another data item, as when the figurative constant appears in a DISPLAY, STRING, STOP or UNSTRING statement, the length of the string is one character.

A figurative constant may be used wherever a literal appears in the format, except that whenever the literal is restricted to having only numeric characters in it, the only figurative constant permitted is ZERO (ZEROS, ZEROES).

Each reserved word which is used to reference a figurative constant value is a distinct character string with the exception of the construction 'ALL literal' which is composed of two distinct character strings.

SPECIAL CHARACTER WORDS

These are the arithmetic operators (+ – / * or **) or the relational characters (< > =). They are described under arithmetic expressions and conditional expressions in the Procedure Division.

2.2.3.3 Literals

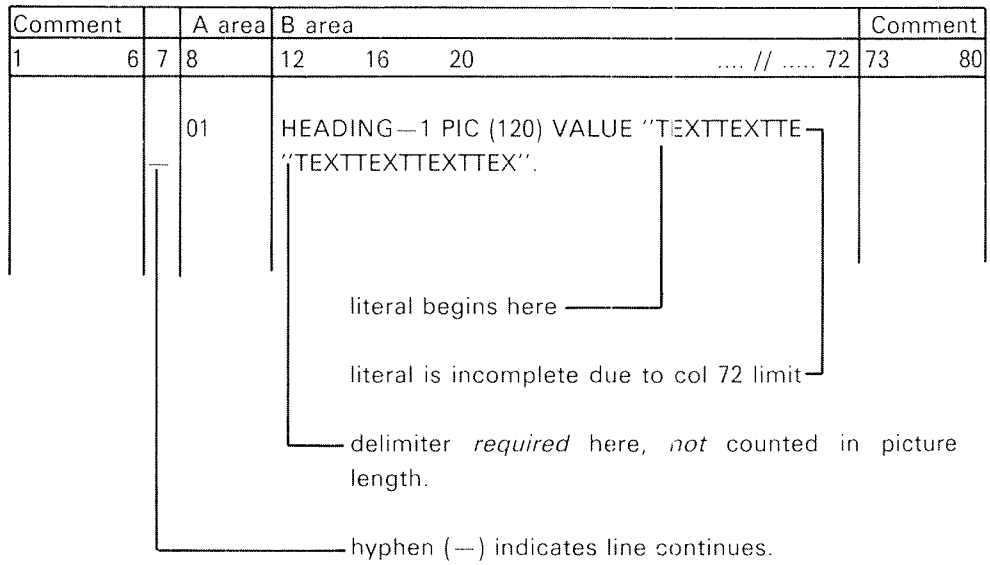
A *literal* is a character string with a value specified either by the ordered set of characters by which it is composed or by a figurative constant. There are two types of literals: nonnumeric and numeric.

A *nonnumeric* literal is a character string bounded by quotation marks containing any allowable character from the ASCII character set. Its maximum length is 150.

Any punctuation characters included within a character string are part of its value.

A matching pair of either single or double quotes is allowed to bound the character string forming a nonnumeric literal. If the character string is bounded by single quotes then each embedded quotation mark must be represented by a pair of single quotes. If, however, the bounds are double quotes then each embedded quotation mark must be represented by a pair of double quotes.

Nonnumeric literals — a coding example:



A *numeric* literal is a character string whose characters are selected from the digits '0' through '9', the plus sign, the minus sign and/or the decimal point. The rules for formation of numeric literals are as follows:

1. A literal must contain at least one digit.

2. One through 18 digits are allowed.

3. A literal must not contain more than one sign. The sign must always be in the leftmost position.

4. It must not contain more than one decimal point. This must not be in the rightmost position.

2.2.3.4 Separators

A separator is a character or two contiguous characters formed according to the following rules:

1. The punctuation character space is a separator. Anywhere a space is used as a separator or as part of a separator, more than one space may be used. All spaces immediately following the separators comma, semicolon or period are considered part of that separator and are not considered to be the separator space.

2. Except when the comma is used in a PICTURE character-string, the punctuation characters comma and semicolon, immediately followed by a space, are separators that may be used anywhere the separator space is used. They may be used to improve program readability.

3. The punctuation character period, when followed by a space is a separator. It must be used only to indicate the end of a sentence, or as shown in formats.

4. The punctuation characters right and left parenthesis are separators. Parentheses may appear only in balanced pairs of left and right parentheses delimiting subscripts, reference modifiers, arithmetic expressions, boolean expressions, or conditions.

5. The punctuation character quotation mark is a separator. An opening quotation mark must be immediately preceded by a space or left parenthesis; a closing quotation mark, both when paired with an opening quotation mark, must be immediately followed by one of the separators space, comma, semicolon, period, or right parenthesis.

6. The spearator space may optionally immediately precede all separators except:

   a. The separator closing quotation mark. In this case, a preceding space is considered as part of the nonnumeric literal and not as a separator.

7. The separator space may optionally immediately follow any separator except the opening quotation mark. In this case, a following space is considered as part of the nonnumeric literal and not as a separator.

Any punctuation character which appears as part of the specification of a PICTURE character-string or numeric literal is not considered as a punctuation character, but rather as a symbol used in the specification of that PICTURE character-string or numeric literal. PICTURE character-strings are delimited only by the separators space, comma, semicolon, or period.

The rules established for the formation of separators do not apply to the characters which comprise the contents of nonnumeric literals or comment lines.

2.2.4 COBOL Format

COBOL programs must be written in a standard format based on an 80 character line. The output listing of the source program is printed in the same format.

The illustration in this section shows the layout of a coding sheet. The following code rules include a description of th fields within it.

Continuation area (column 7)

Here one indicates the continuation of words and numeric literals from the previous to the current line. The symbol used is a hyphen.

If there is no hyphen the preceding line is assumed to be followed by a space.

If there is a hyphen in the continuation area, then the first nonblank character of this line immediately follows the last nonblank character of the preceding line without an intervening space.

If there is a nonnumeric literal in the line to be continued which does not have a closing quotation mark, then all spaces up to and including column 72 are considered to be part of this literal. The continuation line must contain a hyphen in its continuation area and the first nonblank character must now be a quotation mark. (See the coding example of a nonnumeric literal in the previous section on "literals".)

Area A and area B

These occupy columns 8 through 11 and 12 through 72 respectively. The elements that may begin in area A and the placement of elements that can follow them are given in the following chart.

Sequence Rules for Elements in Areas A and B

Elements in Area A	Followed by:	Elements placed in:
Division header	(Procedure Division only) USING	Area B (same or next line)
	section header paragraph header DECLARATIVES	Area A (next line)
Section header	USE statement	Area B
	paragraph header paragraph name (either to follow USE, if specified)	Area A (next line)
paragraph header or paragraph name	Environment division entry Procedure division sentence	Area B (same or next line)
level indicator level number	data name	Area B (same line)
DECLARATIVES	Declaratives section name	Area A (next line)
END DECLARATIVES	section header	Area A (next line)

Comment Lines

A comment line is any line with an * (asterisk) or / (slash) in column 7. It may appear on any line following the one containing the identification division header. A comment may be written in areas A or B and contain any characters from the ANSCII character set.

The * denotes that the comment is to be printed in the output listing immediately following the last preceding line. The / denotes that the current page of the output listing is to be ejected and that the comment will appear on the first line of the next page.

Coding Sheet Layout

Standard COBOL coding sheets are rarely used when programming on the ND system, as most programmers will "code" via the ND EDITORS. However, the following layout should be useful as the coding sheet fields are referred to in the text.

Comment		A area	B area	Comment
1 6	7	8	12 16 20 .... // 72	73 80

3 THE IDENTIFICATION DIVISION

The identification division must be included in every source program. This division names the source programs and the object program.

A *source program* is the initial COBOL program. An *object program* is the output from the compilation.

In addition, the user may include in this division information such as the date the program was written, etc.

Format:

IDENTIFICATION DIVISION.
PROGRAM-ID. program name.
[AUTHOR. [comment entry] ...]
[INSTALLATION. [comment entry] ...]
[DATE-WRITTEN. [comment entry] ...]
[DATE-COMPILED. [comment entry] ...]
[SECURITY. [comment entry] ...]
[REMARKS [comment entry] ...]

The Identification Division must begin with the reserved words IDENTIFICATION DIVISION followed by a period.

The PROGRAM-ID paragraph gives the name by which a program is identified and it must be the first paragraph in the Identification Division. The other paragraphs are optional.

Use of the Date-Compiled paragraph does not produce the compilation date on that line. The date of compilation always appears on the first page of the listing, whether or not this paragraph is present.

All comment-entries serve only as documentation, the syntax of the program is unaffected by them.

4 THE ENVIRONMENT DIVISION

The Environment Division contains a description of the computer on which the source program is compiled together with the functions that are dependent on its physical characteristics. The presence of the Environment Division is optional.

General Format:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION
[SOURCE-COMPUTER. computer name [WITH DEBUGGING MODE] . ]
[OBJECT-COMPUTER. computer name]
    [.SEGMENT-LIMIT IS segment number]
[SPECIAL-NAMES. [.CURRENCY SIGN IS literal]
    [.DECIMAL-POINT IS COMMA] . ]
[INPUT-OUTPUT SECTION.
FILE-CONTROL. file control entry [file control entry] ...
[I-O-CONTROL. input-output control entry]]
```

4.1 CONFIGURATION SECTION

4.1.1 Source Computer Paragraph

Format:

```
SOURCE-COMPUTER.

[ ND-10    ]
[ ND-100   ]  [WITH DEBUGGING MODE].
[ ND-100CE ]
[ ND-500   ]
```

The WITH DEBUGGING MODE clause indicates that all debugging lines are to be compiled. If it is not specified, debugging lines will be compiled as if they were comment lines.

A *debugging line* is any line in a source program with a "D" coded in column 7 (the continuation area).

Each line must be written so that a syntactically correct program results when the debugging lines are compiled into the program. Debugging lines may be continued but each continuation line must contain a "D" in column 7.

Debugging lines may be specified only after the SOURCE-COMPUTER paragraph.

4.1.2 Object Computer Paragraph

Format:

OBJECT-COMPUTER.

[ND-10 | ND-100 | ND-100CE | ND-500]

[,SEGMENT-LIMIT IS segment number].

The SEGMENT-LIMIT clause is treated by the compiler as comments only.

4.1.3 Special-Names Paragraph

The SPECIAL NAMES paragraph provides a substitute character for the currency symbol and specifies whether the functions of the decimal point and comma are to be exchanged in PICTURE clauses and numeric literals. For the format see the beginning of this chapter.

4.1.3.1 Currency Is Clause

The literal which appears in the CURRENCY SIGN IS literal clause is used in the PICTURE clause to represent the currency symbol. The literal is limited to a single character and must not be one of the following characters:

1. digits 0 through 9
2. alphabetical characters A, B, C, D, L, P, R, S, V, X. Z or the space
3. special characters '*', '+', '—', ',', '.', ';', '(',')', '"', '/', '='.

If this clause is not present, only the currency sign is used in the PICTURE clause.

4.1.3.2 Decimal-Point Is Comma Clause

When specified means that the function of the comma and period are exchanged in the character string of the PICTURE clause and in numeric literals.

4.2 INPUT-OUTPUT SECTION

The input-output section names files and provides specifications for other file related information. Its general format is shown at the beginning of this chapter.

4.2.1 File Processing — Language Concepts

The way in which COBOL files in a program are processed depends on how the data is organized on a file and how this data is to be accessed.

4.2.1.1 Data Organization

This refers to the permanent logical structure of the file and is defined as one of three types.

1. *Sequential Organization*

   With this organization, each record in the file except the first has a unique predecessor record, and each record except the last has a unique successor record. These predecessor/successor relationships are established by the order of the WRITE statements when the file is created. Once established, these relationships do not change, however it is possible to add records to the end of the file. The records may be fixed or variable length.

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 09.11.35   DATE: 22.10.80
    SOURCE FILE: (TD)GENSEQ

      1        IDENTIFICATION DIVISION.
      2        PROGRAM-ID.
      3              GENSEQ.
      4       ***************************************************************
      5       *      CREATES SQ-FILE AND LISTS.
      6       ***************************************************************
      7        ENVIRONMENT DIVISION.
      8        INPUT-OUTPUT SECTION.
      9        FILE-CONTROL.
     10              SELECT SQ-FILE ASSIGN "COB1:DATA" ,
     11                             ORGANIZATION IS SEQUENTIAL,
     12                             ACCESS IS SEQUENTIAL.
     13       DATA DIVISION.
     14       FILE SECTION.
     15       FD    SQ-FILE
     16               LABEL RECORDS STANDARD
     17               DATA RECORDS M-REC.
     18       01  M-REC.
     19             02  FILLER PIC X(10).
     20             02  SEQNUM PIC 9(5).
     21             02  FILLER PIC X(5).
     22             02  FILLER PIC X(40).
     23        WORKING-STORAGE SECTION.
     24       01    RANDNO          COMP,  VALUE ZERO.
     25       01    MAXRAND         COMP,  VALUE 1000.
     26       01    NORECS          PIC 9(4).
     27       01    RECCNT          COMP, VALUE 0.
     28
     29       PROCEDURE DIVISION.
     30       INIT-01.
     31             OPEN OUTPUT SQ-FILE.
     32             DISPLAY 'CREATE  RECORDS ?'.
     33             PERFORM GET-NORECS.
     34
     35             PERFORM CRE-SQ-FILE NORECS TIMES.
     36       *                                  BUILD THE INPUT FILE
     37             CLOSE SQ-FILE.
     38             DISPLAY 'FILE SQ-FILE CREATED.', RECCNT, 'RECORDS.'.
     39             OPEN INPUT SQ-FILE.
     40       LIST-FILE-0.
     41             MOVE 0 TO RECCNT.
     42       LIST-FILE-1.
     43             READ SQ-FILE AT END  GO TO LIST-END.
     44             ADD 1 TO RECCNT.
     45             DISPLAY 'REC ', RECCNT, ' SEQNUM = ', SEQNUM.
     46             GO TO LIST-FILE-1.
     47       LIST-END.
     48             CLOSE SQ-FILE.
     49             DISPLAY "JOB FINISH".
     50             STOP RUN.
     51
     52       CRE-SQ-FILE.
     53             CALL 'RND' USING RANDNO, MAXRAND.
     54             MOVE ALL '*' TO M-REC.
     55             MOVE RANDNO TO SEQNUM.
     56             ADD 1 TO RECCNT.
     57             DISPLAY "UT REC =", RECCNT, " KEY =", SEQNUM.
     58             WRITE M-REC.
     59
     60       GET-NORECS.
     61             ACCEPT NORECS.
     62             IF NORECS NOT NUMERIC,
     63                             DISPLAY "** NOT NUMERIC DATA ",
     64                             GO TO GET-NORECS.

*** NO ERROR MESSAGES ***
```

2. *Indexed Organization*

A file with this organization is a mass storage file whose records, which may be fixed or variable, are accessed by means of a key. Each record can have one or more keys and each key is associated with a particular index held on that file. Each index provides a logical path to the data records according to the contents of a data item within each record as the *record key* for that index.

The RECORD KEY clause in the file control entry for each file names the *prime record key* for that file. When inserting, updating or deleting records in a file, each record must be identified solely by its prime record key. This value must, therefore, be unique and it must not be changed when updating the record.

The ALTERNATE RECORD KEY clause names an alternate record key for a file. (This value may be nonunique if the DUPLICATES phrase is specified for it.) These keys provide alternate access paths for record retrieval from the file.

3. *Relative Organization*

Relative file organization is permitted only on mass storage devices. The file may be thought of as a string of areas, each capable of holding a logical record. Each of these is identified by a relative record number which is used for storage and retrieval.

For example, the tenth record is the one addressed by relative record number 10 and is in the tenth record area, whether or not records have been written in the first through ninth record areas.

Records may be of fixed or variable length.

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 15.57.20   DATE: 21.10.80
    SOURCE FILE: ISAM-EX1

 1        IDENTIFICATION DIVISION.
 2        PROGRAM-ID.
 3              GEN-ISAM-1.
 4       ***************************************************************
 5       *      ISAM (INDEX SEQUENTIAL ACCESS METHOD ).
 6       *THE RECORD IS THE OUTPUT TO AN ISAM FILE USING THE *UNIQUE*
 7       *(IE: NO DUPLICATES ) DATA FOUND IN FIELD ISAM-KEY AS *KEY* VALUE.
 8       *
 9       *  BEFORE THIS JOB CAN BE RUN THE FOLLOWING *MUST* BE SO :
10       *      A) FILE "ISAM-EX:DATA" MUST EXIST.
11       *      B) FILE "ISAM-EX:ISAM" MUST NOT EXIST OR IF EXISTING
12       *                                CONTAIN *NO DATA !!*
13       ***************************************************************
14        ENVIRONMENT DIVISION.
15        INPUT-OUTPUT SECTION.
16        FILE-CONTROL.
17              SELECT ISAM-FILE ASSIGN TO "ISAM-EX:DATA",
18                             ORGANIZATION IS INDEXED,
19                             ACCESS MODE IS DYNAMIC,
20                             RECORD KEY IS  ISAM-KEY,
21                             FILE STATUS IS ISAMSTATUS.
22        DATA  DIVISION.
23        FILE SECTION.
24
25        FD ISAM-FILE
26              RECORD CONTAINS 46 CHARACTERS,
27              DATA RECORD IS ISAM-REC.
28        01 ISAM-REC.
29              02 ISAM-KEY     PIC X(6).
30       *          :.....................MUST BE IN RECORD AREA !
31              02 ISAM-TEXT     PIC X(40).
32
33        WORKING-STORAGE SECTION.
34        01 ISAMSTATUS          PIC XX.
35       *                                RETURN STATUS FROM ISAM.
36       ***************************************************************
37        PROCEDURE DIVISION.
38        A001.
39              OPEN I-O ISAM-FILE.
40        A002.
41              DISPLAY "ENTER KEY  (MAX 6 CHAR ) :",
42              ACCEPT ISAM-KEY.
43              IF ISAM-KEY = SPACES GO TO LIST.
44       *                                SPACES INPUT , END DIALOG
45              DISPLAY "ENTER TEXT (MAX 40 CHAR) :".
46              ACCEPT ISAM-TEXT.
47       *                                READ RECORDS FROM TERMINAL
48              WRITE ISAM-REC , INVALID KEY,
49                  DISPLAY "ISAM FILE ERROR :", ISAMSTATUS, ":".
50              GO TO A002.
51       *                                OUTPUT RECORD AND ASK AGAIN
52        LIST.
53              DISPLAY "ENTER ACCESS  KEY :".
54              ACCEPT ISAM-KEY.
55              IF ISAM-KEY  = SPACES  GO TO FINI.
56              READ ISAM-FILE RECORD KEY IS ISAM-KEY INVALID KEY,
57                             DISPLAY "** RECORD NOT FOUND !",
58                             GO TO LIST.
59              DISPLAY "REC: ", ISAM-KEY, ": ", ISAM-REC.
60              GO TO LIST.
61        FINI.
62              CLOSE ISAM-FILE.
63              DISPLAY "JOB END".
64              STOP RUN.

*** NO ERROR MESSAGES ***
```

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 09.06.39   DATE: 22.10.80
    SOURCE FILE: REL-EX

 1        IDENTIFICATION DIVISION.
 2        PROGRAM-ID.
 3              GENRELATIVE.
 4       ****************************************************************
 5       *     SHOWS THE USAGE OF A RELATIVE FILE :
 6       * THE FILE *MUST* EXIST BEFORE THE RUN BUT MAY BE EMPTY, EACH
 7       * RECORD IS LOCATED DIRECTLY BY ITS RELATIVE (TO 1) POSITION IN
 8       * THE FILE BY ITS *NUMERIC* KEY VALUE.
 9       ****************************************************************
10        ENVIRONMENT DIVISION.
11        INPUT-OUTPUT SECTION.
12        FILE-CONTROL.
13              SELECT RELFILE ASSIGN "RELATIVE-EX:DATA" ,
14                        ORGANIZATION IS RELATIVE,
15                        ACCESS IS DYNAMIC,
16                        RELATIVE KEY IS REL-KEY,
17                        FILE STATUS IS REL-STATUS.
18        DATA DIVISION.
19        FILE SECTION.
20
21        FD  RELFILE
22                  LABEL RECORD IS OMITTED
23                  DATA RECORD IS  REL-RECORD
24                  BLOCK CONTAINS  10 RECORDS
25                  RECORD CONTAINS 60 CHARACTERS.
26        01  REL-RECORD       PIC X(60).
27       *                       RECORD CANNOT BE "QED" TYPE RECORD
28
29        WORKING-STORAGE SECTION.
30        01 REL-STATUS         PIC XX.
31        01 REL-KEY            PIC 999.
32       *           :.........CANNOT APPEAR IN RELFILE RECORD AREA,
33       *                       MAX POSSIBLE SIZE IS 999999, RESTRICTED
34       *                       TO 999 IN THIS PROGRAM.
35
36        PROCEDURE DIVISION.
37
38        A000.
39              OPEN I-O RELFILE.
40        A002.
41              DISPLAY "ENTER KEY (MAX 999 ) :".
42              PERFORM GET-KEY.
43              IF REL-KEY = ZEROES GO TO A003.
44              DISPLAY "ENTER TEXT ( MAX 60 CHAR ) :".
45              ACCEPT REL-RECORD.
46              WRITE REL-RECORD INVALID KEY,
47                              DISPLAY " ** RELFILE ERROR :", REL-STATUS.
48              GO TO A002.
49        A003.
50              DISPLAY "ENTER ACCESS KEY :".
51              PERFORM GET-KEY.
52              IF REL-KEY = ZEROS GO TO A999.
53
54              READ RELFILE RECORD INVALID KEY,
55                           DISPLAY " ** RECORD NOT FOUND !", REL-STATUS,
56                           GO TO A003.
57              DISPLAY "REC :", REL-KEY, ":", REL-RECORD.
58              GO TO A003.
59        A999.
60              CLOSE RELFILE.
61              DISPLAY "JOB END".
62              STOP RUN.
63        GET-KEY.
64              ACCEPT REL-KEY.
65              IF REL-KEY NOT NUMERIC ,
66                              DISPLAY " ** KEY MUST BE NUMERIC ",
67                              GO TO GET-KEY.
68        GET-KEY-EXIT.
69              EXIT.

*** NO ERROR MESSAGES ***        ND-60.144.02
```

4.2.1.2 Access Modes

Three access modes are available in COBOL: *sequential, random,* and *dynamic.*

For *sequential organization* records can only be accessed in sequential access mode, i.e., in the order in which they were originally written on the file. A sequential mass storage file may be used for input and output at the same time. One file maintenance method made possible by this facility is to read a record, process it and, if it is updated, write it, modified, to its previous position.

For *indexed organization,* using the sequential access mode means that records are accessed in the ascending order of the record key values. (The order of retrieval of records within a set of records having duplicate key record values is the order in which the records were written into the set.)

Using the random access mode, records are accessed in a sequence determined by the programmer. A desired record is accessed by having its record key defined as a record key data item.

Using the dynamic access mode, the programmer may change at will, by means of appropriate coding, from sequential access to random access.

For *relative organization,* the file can be accessed either sequentially, dynamically or randomly. Sequential access provides the same results as if the file were organized sequentially. Records are accessed in ascending order of relative record number of records currently existing on the file.

Using random mode, the access sequence is controlled by the programmer. The desired record must have its relative record number placed in a relative key data item.

Such a file may be thought of as a serial string of areas, each capable of holding a logical record. Each of these areas is denominated by a relative record number. Records are stored and retrieved based on this number. For example, the tenth record is the one addressed by relative record number 10 and is the tenth record area, whether or not records have been written in the first through the ninth record areas.

In the dynamic access mode, the programmer may change at will from sequential access to random access using appropriate forms of input-output statements.

4.2.2 The File-Control Paragraph

The FILE-CONTROL paragraph associates each file with an external medium and allows specification of file organization, access mode, etc.

FILE-CONTROL.

[select-entry] . . .

Format 1: Sequential files

SELECT [OPTIONAL] file name

ASSIGN TO assignment-name-1

[;RESERVE integer [AREA / AREAS]]

[;ORGANIZATION IS SEQUENTIAL]

[;ACCESS MODE IS SEQUENTIAL]

[;FILE STATUS IS data-name-1].

Format 2: Indexed Files

SELECT file-name

ASSIGN TO assignment-name-1

[;RESERVE integer [AREA / AREAS]]

ORGANIZATION IS INDEXED

[;ACCESS MODE IS {SEQUENTIAL / RANDOM / DYNAMIC}]

;RECORD KEY IS data-name-2

[;ALTERNATE RECORD KEY IS data-name-3 [WITH DUPLICATES]]...

[;FILE STATUS IS data-name-4].

Format 3: Relative Files

```
SELECT file-name

    ASSIGN TO assignment-name-1

    ;RESERVE integer  AREA
                      AREAS

    ;ORGANIZATION IS RELATIVE

    [;ACCESS MODE IS { SEQUENTIAL } [,RELATIVE KEY IS data-name-5]
                     { RANDOM     }
                     { DYNAMIC    }  ,RELATIVE KEY IS data-name-5 ]

    [;FILE STATUS IS data-name-6].
```

Format 4: Sort/Merge

```
SELECT file-name ASSIGN TO assignment-name-1.
```

The SELECT clause must appear first in the file control entries but subsequent clauses may appear in any order.

Each file described in the Data Division must appear in one and only one entry in the file control paragraph.

The default access mode is sequential.

The file status data-name, (data-names-1, 4 and 6) must be defined in the Data Division as a two character, alphanumeric item which is not, however, defined in the file section.

All data-names may be qualified.

4.2.2.1 For Sequential Organization

When the ORGANIZATION IS SEQUENTIAL clause does not appear the existence of this clause is implied.

The OPTIONAL phrase may be specified for input or output files. Its specification is required for input or output files that are not necessarily present each time the object program is executed.

4.2.2.2 For Indexed Organization

Data-names 2 and 3 must be defined as alphanumeric in a record description entry for that file name. Neither can describe an item whose size is variable.

Data-name-3 cannot reference an item whose leftmost character position corresponds to the leftmost character position of an item referenced by data-name-2 or by any other data-name-3 associated with this file.

4.2.2.3 For Relative Organization

Data-name-5, which must be an unsigned integer, must not be described in a record description entry associated with that file.

If a relative file is referenced by a START statement then the RELATIVE KEY phrase must appear for that file.

4.2.2.4 General Rules:

1. The ASSIGN clause specifies the association of a file name with a storage medium.

2. The ORGANIZATION clause defines the logical structure of a file. This is established when the file is created and cannot be subsequently changed.

3. The RESERVE clause is treated as comments and appears for syntax reasons only.

4. When the FILE STATUS clause appears, the COBOL library system, after execution of every statement referencing the file, moves a value indicating the status of the execution into the data item referenced by this clause (see I-O Status under INPUT-OUTPUT statements in the Procedure Division description).

Records in the file are accessed in the sequence determined by the predecessor successor relationships established by the execution of WRITE statements in the file formation.

General Rules for Indexed Organization:

1. When the access mode is sequential, records in the file are accessed in the order of ascending record key values within a given key of reference. If the access mode is random then the value of the record key indicates the record to be accessed. When the access mode is dynamic the file may be accessed sequentially and/or randomly.

2. The RECORD KEY clause denotes the prime record key for the file and its values must be unique. The ALTERNATE RECORD KEY specifies an alternate record key for the file. Both record keys provide access paths to the records in the file.

For Relative Organization

1. When the access mode is sequential, records are accessed in the order of ascending relative record numbers of records existing on the file. If the access mode is random then the value of the RELATIVE KEY data item is used to locate a record. When the access mode is dynamic, records in the file can be accessed sequentially and/or randomly.

2. All records stored in a file are uniquely identified by relative record numbers. These specify the record's logical ordinary position as follows: the first logical record has a relative record number of one (1) and subsequent records have relative record numbers of 2, 3, 4, ...

4.2.3 The I-O Control Paragraph

(Sequential Files Only)

The I-O-CONTROL paragraph specifies the memory area to be shared by different files.

Format:

I-O-CONTROL.

[SAME AREA for file-name-1 {file-name-2} ...] ...

The I-O CONTROL paragraph is optional. More than one SAME clause may be included in a program however:

1. A file name must not appear in more than one SAME AREA clause.

2. The files referenced in the SAME AREA clause need not all have the same access.

The SAME AREA clause specifies that two or more files not representing sort files are to use the same memory area during processing. The area being shared includes all storage areas assigned to the specified files so that it is not valid to have more than one of the files open at the same time.

5 THE DATA DIVISION

5.1 DATA CONCEPTS

The Data Division describes the data that the object program is to accept as input, to manipulate, to create or to produce as output. Data to be processed falls into three categories:

1. That which is contained in files and enters or leaves the computer memory from specified areas. This data is *external* data.

2. That which is developed internally and placed into intermediate storage. This is known as *internal data.*

3. Constants defined by the user.

External data is contained in files. A *file* is a collection of records existing on an input or output device. When discussing records it is important to distinguish between the terms *physical record* and *logical record.* A physical record is a collection of data which is treated as an entity by the particular input or output device on which it is stored. A logical record is a collection of data having a logical relationship between its subdivisions. One logical relationship may extend accross physical records. Several may be contained within one physical record or it may be identical in size, i.e., contained completely on one physical unit of data. Unless otherwise described, the term record refers to a logical record when used in this manual.

The term *block* is associated with the use of records, usually to describe a unit of data consisting of one or more logical records. The term is synonomous with physical record.

5.2 STRUCTURE OF THE DATA DIVISION

The Data Division is divided into sections, each one having a specific logical function. The occurrence of indicidual sections is optional but they must appear in the order shown when written in the source program. It has the following format:

```
DATA DIVISION.

[ FILE SECTION.
    [ file-description-entry [record-description-entry] ....
      sort-file description-entry [record-description-entry] .... ]  ... ]

[ WORKING-STORAGE SECTION.
    [ 77-level description entry...
      record-description-entry ... ]  ... ]

[ LINKAGE SECTION.
    [ 77-level-description-entry...
      record-description-entry ... ]  ... ]
```

The file section contains a description of all externally stored data (FD) but not that which the program may develop internally. It also contains a description of each sort/merge file (SD) in the program.

The Working Storage section describes records which are developed and processed internally.

The Linkage Section describes data made available from another program (see the section on Interprogram Communication in the "Other Features" part of this manual).

5.3 FILE SECTION

This section must begin with the header FILE SECTION followed by a period. It contains file description entries and sort file description entries, each one followed by its associated record description. All clauses used in the *record description entry* of the File Section can be used in the Working-Storage section. The elements allowed in a record description are described later under "Data Description Entry" in the Working Storage section of the Data Division description (see also "The Concept of Levels" in that same section).

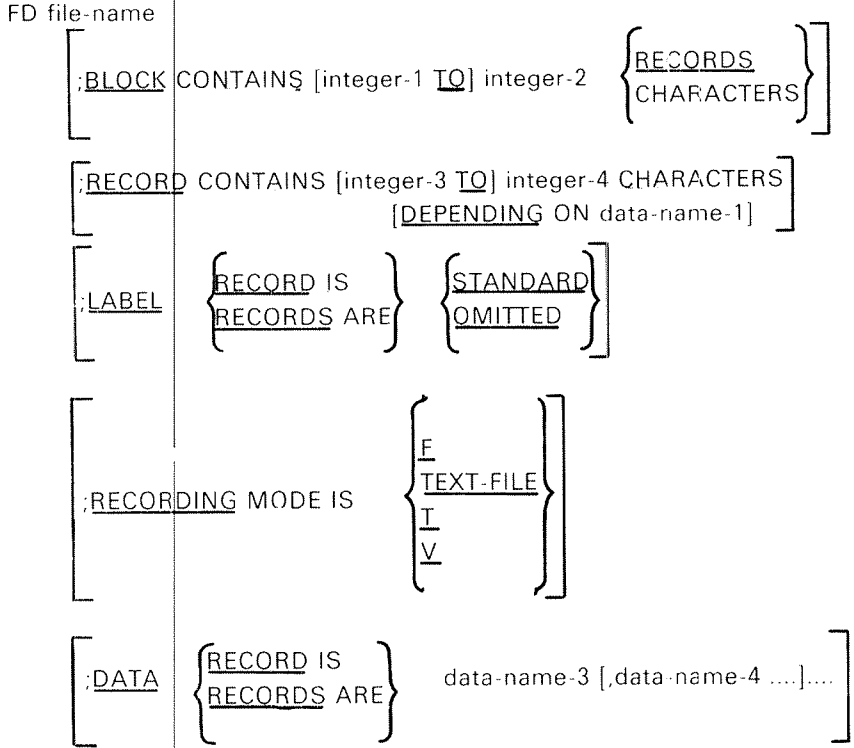
5.3.1 The File Description — Complete Entry Skeleton

The file description entry represents the highest level of organization in the File Section. It follows the File Section header and consists of a level indicator (FD), a file name and a series of independent clauses specifying the size of the physical and logical records, their structure and their record names on that file. The formats are:

Format 1: Indexed and Relative I-O.

```
FD file name

[;BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS    }]
                                          {CHARACTERS}

[;RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS
                  [DEPENDING ON data-name-1 ]                ]

[;LABEL {RECORD IS  }  {STANDARD}]
        {RECORDS ARE}  {OMITTED }

[;RECORDING MODE IS {F}]
                    {V}

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

[;VALUE OF FILE-ID IS integer-3]
```

Format 2: Sequential I-O



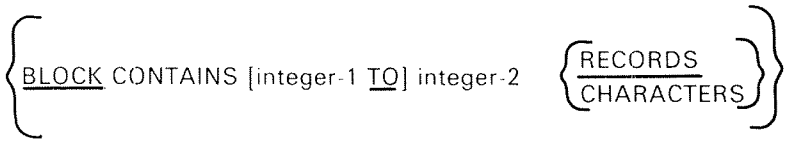
The level indicator FD identifies the beginning of a file description and must precede the file name. The clauses which follow are optional in many cases and they may appear in any order.

One or more record description entries must follow the file description entry.

5.3.1.1 The Block Contains Clause

The block contains clause specifies the size of a physical record.

Format:



General Rules:

1. If this clause is omitted, block size is set to 2048 characters.

2. The size of the physical record may be stated in terms of RECORDS, unless one of the following situations exists, in which case the RECORDS phrase must not be used:

   a) Where logical records may extend across physical records.
   b) The physical record contains padding (area not contained in a logical record).
   c) Logical records are grouped in such a manner that an inaccurate physical record size would be implied.

3. When the word CHARACTERS is specified, the physical record size is specified in terms of the number of character positions required to store the physical record, regardless of the types of characters used to represent the items within the physical record.

4. If only integer-2 is shown, it represents the exact size of the physical record. If integer-1 and integer-2 are both shown, they refer to the minimum and maximum size of the physical record, respectively.

5.3.1.2 The Data Records Clause

The DATA RECORDS clause serves only as documentation for the names of data records with their associated file.

Format:

```
DATA  {RECORD IS  }  data-name-1[,data-name-2 ....] ...
      {RECORDS ARE}
```

Data-name-1 and data-name-2 are the names of data records and must have 01 level number record descriptions, with the same names, associated with them.

General Rules:

1. The presence of more than one data name indicates that the file contains more than one type of data record. These records may be of differing sizes, different formats, etc. The order in which they are listed is not significant.

2. Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.

5.3.1.3 The Label Records Clause

The LABEL RECORDS clause is treated as comments.

Format:

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

5.3.1.4 The Record Contains Clause

The RECORD CONTAINS clause specifies the size of data records.

Format:

RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS
[DEPENDING ON data-name-1]

General Rules:

1. The size of each data record may be completely defined within the record description entry, however, if not, thefollowing notes apply:
   a) Integer-2 may not be used by itself unless all the data records in the file have the same size. In this case integer-2 represents the exact number of characters in the data record. If integer-1 and integer-2 are both shown, they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively.

   b) The size is specified in terms of the number of characters positions required to store the logical record, regardless of the types of characters used to represent the items within the logical record.

2. Data-name-1 must describe an elementary integer in the Working-Storage section. (Defined as COMPUTATIONAL, with no PICTURE clause specified.)

3. If data-name-1 is specified, the number of character positions in the record must be placed into the data item referenced by data-name-1 before any RELEASE, or WRITE statement is executed for the file and it must not be modif ed before any REWRITE statement.

4. If data-name-1 is specified, the execution of a DELETE, RELEASE, REWRITE, START or WRITE statement or the unsuccessful execution of a READ or RETURN statement does not alter the contents of the data item referenced by data-name-1.

5. During the execution of a RELEASE, REWRITE or WRITE statement, the number of character positions in the record is determined by the following conditions:

   a) If data-name-1 is specified, by the contents of the data item referenced by data-name-1.

   b) If data-name-1 is not specified, by the number of character positions in the record.

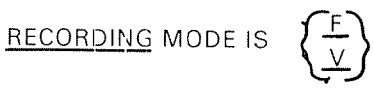
6. If data-name-1 is specified, after the successful execution of a READ or RETURN statement for the file, the contents of the data item referenced by data-name-1 will indicate the number of character positions in the record just read.

7. If the INTO phrase is specified in the READ or RETURN statement, the number of character positions in the current record that participate as the sending data item in the implicit MOVE statement is determined by the maximum size of the sendign field.

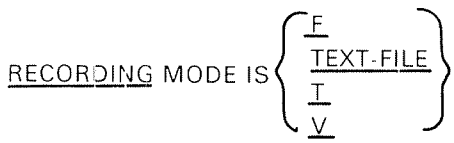
5.3.1.5 The Recording Mode Clause

The RECORDING mode clause specifies the record format used in the file.

Format 1: Indexed and Relative I-O.



Format 2: Sequential I-O.



F indicates that all records have exactly the same numbers of characters, that is, the number which is the length of the file's record area.

V means that the records in the file may have a varying number of characters, never less than 1 (one) and never more than the maximum size of the file's record area. With V format, two extra bytes of information are stored at the beginning of each record in the file. These bytes contain the length of the data portion of the record; they are never available to the COBOL program.

T (TEXT-FILE) means that the records of the file are in printable format and contain only ASCII characters. The records are separated by the characters line feed (12 octal) and carriage return (15 octal). This format is only valid for sequential files. T and TEXT-FILE are synonomous.

5.3.1.6 The VALUE OF FILE-ID IS Clause

The VALUE OF FILE-ID IS clause assigns an identifying value to the file.

Format:

VALUE OF FILE-ID IS integer-3

If an indexed or relative file is to be used in a subroutine, then it *must* be defined in both the main program and the subroutine with the *same* value of integer-3 in a VALUE OF FILE-ID IS clause. Integer-3 can take a value between 1 and 99 inclusively.

5.4 WORKING-STORAGE SECTION

The Working-Storage Section may describe data records which are not part of external files but are developed and processed internally. It must begin with the section WORKING-STORAGE SECTION followed by a period. It contains record description entries and data description entries for noncontiguous data items.

Data Description Entries

Noncontiguous items in Working-Storage that bear no hierarchical relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary items. Each is defined in a separate data desctiption entry with the special level number 77.

Record Description Entry

Data elements that bear a definite hierarchical relationship to one another must be grouped into records structured by level number.

5.4.1 Data Description

5.4.1.1 The Concept of Level

Because records must often be divided into logical subdivisions, the concept of level is inherent in the structure of a record. Fields which cannot be further subdivided are called *elementary items*. A record can be made up of elementary items or it can itself be an elementary item. If it is necessary to refer to a set of elementary items they can be combined as a *group item*. Note that an elementary item can belong to more than one group.

For example, an employers payroll file might contain a record for all employees at one location. Each employee name on the record could be represented as a group item while the subdivisions, or elementary items, might be age, salary, grade, tax code, etc.

Level Numbers

A system of level numbers form 1 to 49 is used to organize elementary and group items into records. Special level numbers 77 and 88 identify items used for special purposes, they do not structure a record and are:

77 For independent working storage or linkage section items which are not subdivisions of items or themselves subdivided.

88 For identification of a condition name associated with a particular value of a conditional variable (see the VALUE clause later in the Data Division section).

(Level 77 and 01 entries must have unique data names as they cannot be qualified. Subordinate data names, if qualifiable, need not be unique.)

Record Description Level Numbers

A level number must be assigned to each group or elementary item in a record. The level numbers used to structure records are:

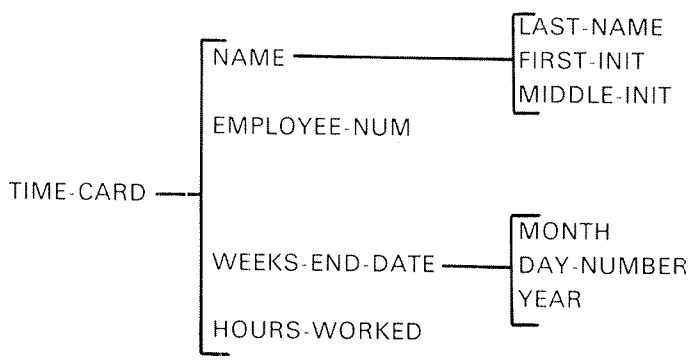
01 This specifies the record itself and is the most inclusive of the numbers. A level 01 entry may be either a group or elementary item.

02-49 These are given to group and elementary items within a record. Subordinate items are given 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 its own is encountered.

All elementary or group items immediately subordinate to one group item must be assigned level numbers higher than the level number of this group item.

For example, data may need to be structured as follows:



A corresponding record might appear in the form:

```
01        TIME-CARD.
  02      NAME.
    03    LAST-NAME                PICTURE X(18).
    03    FIRST-INIT               PICTURE X.
    03    MIDDLE-INIT              PICTURE X.
  02      EMPLOYEE-NUM             PICTURE 99999.
  02      WEEKS-END-DATE.
    05    MONTH                    PIC 99.
    05    DAY-NUMBER               PIC 99.
    05    YEAR                     PIC 99.
  02      HOURS-WORKED             PICTURE 99V9.
```

5.4.1.2 Classes and Categories of Data

There are five categories of data items which are grouped into three classes. The relationship between them is depicted below.

Level of Item:	Class:	Category:
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Alphabetic Numeric Edited Alphanumeric Edited Alphanumeric
Group	Alphanumeric	Alphabetic Numeric Numeric Edited Alphanumeric Edited Alphanumeric

Classes and Categories of Data

Note that for alphabetic and numeric the classes and categories are synonomous. The alphanumeric class includes the categories of alphanumeric edited, numeric edited and alphanumeric (without editing). Every elementary item, except for an index data item, belongs to one of the classes and to one of the categories.

Every group item belongs to the alphanumeric class (even if its subordinate items belong to other classes or categories).

Standard alignment rules for positioning data in an elementary item depend on the data category of the receiving item (i.e., the item into which the data is placed).

If the receiving tem is:

1. *Numeric*

   The data is aligned by decimal point and moved to the receiving character positions with zero fill or truncation on either end as required.

   If there is no assumed decimal point (an *assumed* decimal point is one that has logical meaning but does not exist as a character in the data) then the item is treated as if an assumed decimal point existed immediately after its rightmost character and is aligned as in the preceding rule.

2. *Numeric Edited*

   The data is aligned on the decimal point and (if necessary) truncated or padded with zeros at either end, except when editing causes replacement of leading zeros.

3. *Alphanumeric, Alphanumeric Edited, Alphabetic*

   The data is aligned at the leftmost character position and (if necessary) truncated or padded with spaces. If the JUSTIFIED clause is specified then this rule is modified as described in the description of this clause.

Signed Data. There are two classes of algebraic signs used in COBOL: *operational signs and editing signs.*

Operational signs are associated with signed numeric items to indicate their algebraic properties.

Editing signs, which are PICTURE symbols, are used with numeric edited items to indicate the sign of the item in edited output.

Data Reference

Every user specified name of an element in a COBOL program must be unique — either because no other name has a character string of the same value or because it can be made unique through qualification, indexing or subscripting.

Qualification

A name can be made unique if it exists within a hierachy of names such that it can be identified by specifying one or more higher leve names in this hierachy. This process is called *qualification* and the higher level names are called *qualifiers*.

Qualification is performed by following a user specified name by one or more phrases composed of a qualifier preceded by IN or OF. (IN and OF are logically equivalent.)

The Formats are:

Format 1:

Format 2:

Format 3:

file-name

Each qualifier must be of a successively higher level and be within the same hierachy as the name it qualifies.

The same name must not appear at 2 levels in a hierachy.

If a data name or condition name is assigned to more than one data item, it must be qualified each time it is referred to.

A paragraph name must not be duplicated within a section. When a paragraph name is qualified by a section name the word SECTION must not appear. A file name (used in the COPY statement) must name a SINTRAN file. Paragraph name need not be qualified when referred to within the section in which it appears. When it is being used as a qualifier, a data name cannot be subscripted.

If there is more than one combination of qualifiers that ensures uniqueness then any of these combinations can be used. *Note:* although enough qualification must be given to make the name unique, it may not be necessary to specify all the levels of the hierarchy.

No duplicate section names are allowed.

No data name can be the same as a section name or paragraph name.

Duplication of data names must not occur in those places where the data names cannot be made unique by qualifications.

Subscripting and Indexing

Subscripts and indexes are used for referencing an individual element within a table of elements that do not have individual data names. Subscripting and Indexing are explained in the section on Table Handling under "Other Features".

5.4.2 The Data Description — Complete Entry Skeleton

The format of the complete entry skeleton has been simplified for easier reading. The format of each clause is given with the individual descriptions.

5.4.2.1 Data Description Entry

A data description entry specifies the characteristics of a particular item of data.

Format 1:

level number { data name / FILLER } clause

[;BLANK WHEN ZERO clause]

[;JUSTIFIED clause]

[;PICTURE clause]

[;REDEFINES clause]

[;SIGN clause]

[;SYNCHRONIZED clause]

[;USAGE clause]

[;VALUE clause]

[;EXPORT clause]

Format 2:

88 condition name VALUE clause

Format 1 is used for record description entries and for level 77 entries.

General Rules:

1. The *level number* can be any number from 01 to 49 or 77. 01 to 09 can be written as 1 to 9.

2. The data name/FILLER (optional) entry must immediately follow the level number. Otherwise, the clauses may be written in any order.

3. The PICTURE clause must be specified for all elementary items except index data items and for computational, computational-1 and computational-2 items.

4. The BLANK WHEN ZERO, JUSTIFIED, PICTURE and SYNCHRONIZED clauses are valid only for elementary items.

5. Each entry must end with a period followed by a space and all clauses must be separated by a space or a comma or semicolon followed by a space.

Format 2 describes *condition names* which are user specified names that associate value(s) and/or range of values with a conditional variable. A *conditional variable* is a data item which can take one or more values is associated with a condition name.

General Rules:

1. Each condition name requires a separate entry with level number 88. Any entry beginning with this level number *is* a condition name.

2. A condition name can be associated with any data description entry containing a level number except:

   a) another condition name
   b) an index data item.

3. Each entry must end with a period followed by a space. Successive operands must be separated by a space or a semicolon or comma followed by a space.

5.4.2.2 The Blank When Zero Clause

The BLANK WHEN ZERO clause permits the blanking of an item when its value is zero.

Format:

BLANK WHEN ZERO

The BLANK WHEN ZERO clause can only be used for an elementary item whose PICTURE is numeric or numeric edited (see the PICTURE clause in this section). When it is used for an item whose PICTURE is numeric then the category of the item is considered to be numeric edited.

When the BLANK WHEN ZERO clause is used, the item will contain nothing but spaces if the value of the item is zero.

5.4.2.3 The Data Name/Filler Clause

A data name explicitly identifies the data being described. The key word FILLER, which may be omitted, specifies an item not explicitly referred to in a program.

Format:

In the File, Working-Storage and Linkage Sections, data name or FILLER must appear as the first word following the level number in each data description entry.

General Rules:

1. A data name identifies a data item used in the program, it may assume a number of different values during program execution.

2. The key word FILLER can name an elementary or group item in a record, under no circumstances can a FILLER item be referred to explicitly. However, it may be used as a conditional variable since such use does not require explicit reference to the item itself but only to its value.

5.4.2.4 The Justified Clause

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

Format:

$$\left\{\begin{matrix}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{matrix}\right\} \quad \text{RIGHT}$$

The JUSTIFIED clause can be specified only at the elementary item level. JUST is an abbreviation for JUSTIFIED and has the same meaning. It cannot be used with any data item which is numeric or for which editing is specified.

General Rules:

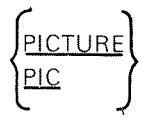
1. When a receiving data item is described with the JUSTIFIED clause and it is smaller than the sending item, the leftmost characters are truncated. If larger, the unused character positions at the left are filled with spaces.

2. When the JUSTIFIED clause is omitted then the standard rules for aligning data within an elementary item apply. (See Standard Alignment Rules under Classes of Data in this section.)

5.4.2.5 The Picture Clause

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

Format:



IS character-string

The PICTURE clause must be specified for every elementary item except an index data item, or computational, computational-1 and computational-2 items. It may be specified only at the elementary level. PIC is an abbreviated form of 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 elementary item. The maximum number of characters, i.e., symbols, allowed in the string is 30.

List of Symbols

The following list of symbols is used to represent the five categories of data that can be described in a PICTURE clause. (These are: alphabetic, number, alphanumeric, alphanumeric edited and numeric edited.) A brief description is given with each symbol. More detailed descriptions appear later.

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 will be inserted.

S — The letter S is used in a character-string to indicate the presence (but not the representation or, necessarily, the position) of an operational sign; it must be the leftmost character in the PICTURE. It is not counted in determining the size of the elementary item unless an associated SIGN clause specifies the SEPARATE CHARACTER phrase. (An operational sign indicates whether the value of the item is positive or negative.)

V — The V is used in a character position to indicate the location of an assumed decimal point and may appear only once in a character-string. It does not represent a character position and is therefore 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 which contains any allowable character from the computer's character set.

Z	Each Z in a character-string may only be used to represent the leftmost leading numeric character positions which will be replaced by a space character when the contents of that character when the contents of that character position is zero. Each Z is counted in the size of the item.
9	Each 9 in the character-string represents a character position which contains a numeral and is counted in the size of the item.
0	Each 0 (zero) in the character-string represents a character position into which the numeral zero will be inserted. It is counted in the size of the item.
/	Each / (slash) in the character-string represents a character position in to which the strobe character will be inserted. It is counted in the size of the item.
,	Each , (comma) in the character-string represents a character position into which the character , (comma) will be inserted. This character position is counted in the size of the item and the character must not be the last character in the PICTURE character-string.
.	When the character . (period) appears in the character-string it is an editing symbol which represents the decimal point for alignment purposes. In addition, it represents a character position into which the . (period) will be inserted. The character is counted in the size of the item. In a program the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. (In the exchange the rules for the period apply to the comma and vice versa when the appear in a PICTURE clause.) The insertion character . (period) must not be the last character in the PICTURE character-string.
+, —, CR, DB	These symbols are used as editing sign control symbols and represent the character position into which the editing sign control symbol will be placed. These symbols are mutually exclusive in any one character string and each character used in the symbol is counted in determining the size of the data item.
*	Each * (asterisk) in the character-string represents a leading numeric character position into which an asterisk will be placed when the contents of that position is zero. Each * is counted in the size of the item.

CS The currency symbol in the character-string represents a character position into which a currency symbol is to be placed. This currency symbol is represented either by the currency sign or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. The currency symbol is counted in the size of the item. (The default symbol is $ (dollar)).

Allowable symbols for each data category.

The following rules apply:

Alphabetic Items

a) The PICTURE character-string can contain only the symbols A and B.

b) Its contents when represented in standard data format should be any combination of the 26 (twenty-six) letters of the roman alphabet and the space from the COBOL character set.

Numeric Items

a) The PICTURE character-string may only contain the symbols 9, S and V. The number of digit positions must range through 1 to 18 inclusive.

b) The contents of the item in standard format must be combination of the 10 arabic numerals and, if signed, a representation of the operational sign.

Alphanumeric Items

a) The PICTURE character-string is restricted to certain combinations of the symbols A, X and 9. The item is treated as if the character-string contained all X's. A character-string containing all A's or all 9's does not define an alphanuemeric item.

b) The contents of the character-string when represented in standard data format are allowable characters in the computer's character set.

Alphanumeric Edited Items

a) The PICTURE character-string can contain: A, X, 9, B, 0 (zero) and /. It must contain at least one of these combinations:

- at least one B and at least one X
- at least one 0 and at least one X
- at least one X and at least one /
- at least one A and at least one 0
- at least one A and at least one /

b) The contents of the items in standard data format may be any allowable character from the computer's character set.

Numeric Edited Items

a) The PICTURE character-string can contain the symbols: B, V, Z, 9, 0 (zero), *, /, , (comma), . (period), +, —, CR, DB or the currency symbol. The allowable combinations are determined from the order of precedence of symbols (see chart) and the editing rules (see later in this section).

b) The character-string must contain at least one 0 (zero), B, /, Z, *, +, —, , (comma), . (period), CR, DB or currency symbol and the number of digit positions that can be represented must range from 1 to 18 inclusive.

c) The contents of the character positions that are allowed to represent a digit in standard format, must be one of the numerals.

The Size of an Elementary Item

The size of an elementary item (i.e., the number of character positions it occupies in standard data format) is determined by the number of allowable symbols that represent character positions. An integer enclosed in parentheses following the symbols A, , (comma), X, 9, Z, *, B, /, 0 (zero), +, — or the currency symbol indicates the number of consecutive occurrences of the symbol.

5.4.2.6 Editing Rules for the PICTURE Clause

Editing is performed in two ways, either by *insertion*, or *suppression and replacement*. Insertion editing breaks down into four types. These are listed below together with the characters and categories each is valid for.

Simple Insertion:

Category:	Insertion Symbols:
Alphabetic	B
Alphanumeric edited	B 0 /
Numeric Edited	B 0 / ,

Examples:

Picture:	Data:	Edited Result:
99,999,000	12345	12,345,000
999,999	12345	012,345
A(5)BA(4)	NORSKDATA	NORSK DATA
X(4)B/BX(2)	TYPE25	TYPE / 25

Each insertion symbol is counted in the size of the item and represents the position where the equivalent character will be inserted.

Special Insertion:

Category:	Insertion Symbol:
Numeric Edited	. (period)

Examples:

Picture:	Data:	Edited Result:
99.99	123.4	23.40
99.99	12.34	12.34
99.99	1.234	01.23

The insertion symbol . (period) will be counted in the size of the item, it shows the position where the actual decimal point will be inserted. It is not allowed to appear in the same PICTURE character-string as the symbol V (denoting an assumed decimal point). These two symbols are mutually exclusive.

Fixed Insertion:

Category:	Insertion Symbols:
Numeric edited	+ — CR DB (editing sign control symbols) $ (currency symbol)

Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character-string. When the symbols 'CR' or 'DB' are used they represent two characters positions in determining the size of the item and they must represent the rightmost character positions that are counted in the size of the item. The symbol '+' or '—', when used, must be either the leftmost or rightmost character position to be counted in the size of the item. The currency symbol must be the leftmost character position to be counted in the size of the item except that it can be preceded by either a '+' or a '—' symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the PICTURE character string. Editing sign control symbols produce the following results depending upon the value of the data item.

Editing Symbol in Picture Character-String:	Result:	
	Data Item Positive or Zero	Data Item Negative
+	+	—
—	space	—
CR	2 spaces	CR
DB	2 spaces	DB

Examples:

Picture:	Data:	Edited Result:
+99.99	—12.345	—12.34
—99.99	+12.345	12.34
99.99+	+12.345	12.34+
$99.99	—12.34	$12.34
—$99.99	—12.34	—$12.34
$999.99 CR	+12.34	$012.34
$999.99 DB	—12.34	$012.34 DB

Floating Insertion:

Category:	Insertion Symbols:
Numeric edited	$ + –

Floating insertion editing occurs when two or more of the above insertion symbols appear as a string within the given PICTURE character-string.

Examples:

Picture:	Data:	Edited Result:
$$99	12	$12
$$$$$99	1234	$1234
$$$$$9.99	.12	$0.12
+++/+++,+99	12	+12
––––––9,999	123456	–123,456
$$$$$99.99CR	–123	$123.00CR

Within one PICTURE character-string the floating insertion symbols are mutually exclusive. Simple insertion symbols or the period may appear within a string of floating insertion symbols without causing discontinuity (except in the special case where there is only one floating insertion symbol in the string to the left of a simple one or period).

The leftmost character of the floating insertion string represents the leftmost limit of the floating symbol in the data item. The rightmost character of the floating string represents the rightmost limit of the floating symbols in the data item.

The second floating character from the left represents the leftmost limit of the numeric data that can be stored in the data item. Nonzero numeric data may replace all the characters at or to the right of this limit.

In a PICTURE character-string, there are only two ways of representing floating insertion editing. One way is to represent any or all of the leading numeric character positions on the left of the decimal point by the insertion character. The other way is to represent all of the numeric character positions in the PICTURE character-string by the insertion character.

If the insertion characters are only to the left of the decimal point in the PICTURE character-string, the result is that a single floating insertion character will be placed into the character position immediately preceding either the decimal point or the first nonzero digit in the data represented by the insertion symbol string, whichever is farther to the left in the PICTURE character-string. The character positions preceding the insertion character are replaced with spaces.

If all numeric character positions in the PICTURE character-string are represented by the insertion character, the result depends upon the value of the data. If the value is zero the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point.

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of nonfloating insertion characters being edited into the receiving data item, plus one for the floating insertion character.

Zero Suppression and Replacement Editing

The symbols Z and * are used to replace leading zeros in the edited result by blanks or asterisks respectively. They can form floating strings in the same way as the floating insertion symbols $, + and — described earlier. (A floating string of zero suppression or replacement symbols cannot appear in the same PICTURE character-string as a floating string of insertion symbols however.)

Examples:

Picture:	Data:	Result:
ZZ.ZZ	00.09	.09
ZZ.ZZ	00.00	
.	00.00	**.**
ZZ99.99	0000.00	00.00
*****.99CR	123	**123.00
*,***.**+	—123.00	**123.00—
ZZZ.ZZ+	0	

Any simple insertion symbols or the period may appear within a floating string of zero suppression or replacement characters and are regarded as part of this string.

When editing is performed, any leading zero 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:

1. Does not correspond to a suppression symbol.
2. Is the decimal point.
3. Contains nonzero data.

If, however, the value of the data is zero and all the numeric character positions in the PICTURE character string are represented by a Z, the resulting item will contain all spaces. If these positions are represented by asterisks, the resulting item, except for the decimal point, will contain asterisks.

Precedence Rules

The following chart shows the order of precedence when using characters as symbols in a character string. An 'X' at an intersection indicates that the symbol(s) at the top of the column may precede the symbol(s) at the left of the row. Arguments appearing in braces indicate that the symbols are mutually exclusive. The currency symbol is shown as CS.

At least one of the symbols A, X, Z, 9 or *, or at least two of the symbols +, — or CS must appear in a PICTURE string.

First Symbol / Second Symbol		Non-Floating Insertion Symbols									Floating Insertion & Suppressing/ Replacement Symbols						Other Symbols			
		B	0	/	,	.	{+ −}	{+ −}	{CR DB}	CS	{Z *}	{Z *}	{+ −}	{+ −}	CS	CS	9	A X	S	V
Non-Floating Insertion Symbols	B	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
	{CR DB}	X	X	X	X	X				X	X	X			X	X	X			X
	CS						X													
Floating Insertion and Suppressing/Replacement 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
	CS	X	X	X	X		X								X					
	CS	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
	A X	X	X	X													X	X		
	S																			
	V	X	X	X	X		X			X	X		X		X		X		X	

Figure 5.1. PICTURE Character Precedence Chart

5.4.2.7 The Redefines Clause

The REDEFINES clause allows the same computer storage area to be described by different data description entries.

Format:

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

(Note: The level number, semicolon and data-name-1 are shown in the above format for reasons of clarity. Level number and data-name-1 are not part of the REDEFINES clause.)

Data-name-2 is the *redefined item* while data-name-1 supplies an alternative description for the same area, i.e., is the *redefining item*.

The level numbers of data-name-1 and data-name-2 must be identical but not level 88.

General Rules:

1. Redefinition begins at data-name-1 and ends when a level number less than or equal to that of data-name-2 is encountered. No entry having a level number lower than those of data-names 1 and 2 may occur between these entries.

2. When the level number of data-name-1 is other than 01, it must specify the same number of character positions that the data item referenced by data-name-2 contains. It is important to observe that the REDEFINES clause specifies the redefinition of a storage area, not of the data items occupying the area.

3. Multiple redefinitions of the same character positions are permitted. The entries giving the new descriptions of the character positions must follow the entries defining the area being redefined, without intervening entries that define new character positions.

4. Multiple level 01 entries subordinate to any given level indicator represent redefinitions of the same area.

5. The entries giving the new description of the character positions must not contain any VALUE clauses, except in condition name entries.

Example:

```
02   A PICTURE A(6).
02   B REDEFINES A.
     05  B-1 PICTURE X(2).
     05  B-2 PICTURE 9(4).
02   C PICTURE 9(6).
02   D REDEFINES C.
     05  D-1 PICTURE 99.
     05  D-2 PICTURE 9999.
     05  D-3 REDEFINES D-2 PICTURE 99V99.
```

In this example A, C and D-2 are *redefined* items while B, D and D-3 are *redefining* items. Note that the REDEFINES clause has been specified for the item D-3 which is subordinate to a redefining item, D.

5.4.2.8 The Sign Clause

The SIGN clause specifies the position and mode of representation of the operational sign when it is necessary to describe these explicitly.

Format:

[SIGN IS] { LEADING / TRAILING } [SEPARATE CHARACTER]

The SIGN clause may be specified only for a numeric data description entry whose PICTURE contains the character 'S', or a group item containing at least one such numeric data description entry.

The numeric data description entries to which the SIGN clause applies must be described as usage is DISPLAY.

At most one SIGN clause may apply to any given numeric data description entry.

If the SEPARATE CHARACTER option is not present, then the operational sign is assumed to be associated with the LEADING OR TRAILING digit position (whichever is specified). The PICTURE character S is not counted in the size of the item.

If the SEPARATE CHARACTER option is present, then the operational sign is assumed to occupy the LEADING or TRAILING character position. In this case the PICTURE character S is included in the size of the item. The operational signs for positive and negative are the characters + and — (minus) one of which must be present in the data at object time.

5.4.2.9 The Synchronized Clause

The SYNCHRONIZED clause specifies the alignment of an elementary item on the natural boundaries of the computer memory.

Format:

This clause may only appear with an elementary item.

SYNC is an abbreviation of SYNCHRONIZED.

General Rules:

1. This clause specifies that the subject data item is to be aligned in the computer such that no other data item occupies any of the character positions between the leftmost (SYNC LEFT) or rightmost (SYNC RIGHT) natural boundaries delimiting this data item. If the number of character positions required to store this data item is less than the number of character positions between those word boundaries, the unused character positions (or portions thereof) must not be used for any other data item. Such unused character positions, however, are included in:

   a) the size of any group item(s) to which the elementary item belongs and
   b) the character positions redefined when this data item is the object of a REDEFINES clause.

2. SYNCHRONIZED LEFT specifies that the elementary item is to be positioned such that it will begin at the left character position of the word boundary in which the elementary itme is placed.

3. SYNCHRONIZED RIGHT specifies that the elementary item is to be positioned such that it will terminate on the right character position of the word boundary in which the elementary item is placed

4. Whenever a SYNCHRONIZED item is referenced in the source program, the original size of the item, as shown in the PICTURE clause, is used in determining any action that depends on size, such as justification, truncation or overflow.

5. If the data description of an item contains the SYNCHRONIZED clause and an operational sign, the sign of the item appears in the normal operational sign position, regardless of whether the item is SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT.

6. When the SYNCHRONIZED clause is specified in a data description entry of a data item that also contains an OCCURS clause, or in a data description entry of a data item subordinate to a data description entry that contains an OCCURS clause, each occurrence of the item is synchronized.

5.4.2.10 The Usage Clause

The USAGE clause specifies the format of a data item in the computer storage.

Format:

```
USAGE IS { COMPUTATIONAL      }
         { COMP               }
         { COMPUTATIONAL-1    }
         { COMP-1             }
         { COMPUTATIONAL-2    }
         { COMP-2             }
         { COMPUTATIONAL-3    }
         { COMP-3             }
         { PACKED-DECIMAL     }
         { DISPLAY            }
         { INDEX              }
```

If a COMPUTATIONAL item has a PICTURE character string then it can contain only '9's, the operational sign character 'S', or the implied decimal point character 'V'. (See the PICTURE clause earlier in this section).

COMP is a abbreviation for COMPUTATIONAL.

General Rules:

1. The USAGE clause can be written at any level. If the USAGE clause is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

2. This clause specifies the manner in which a data item is represented in the storage of a computer. It does not affect the use of the data item, although the specifications for some statements in the procedure division may restrict the USAGE clause of the operands referred to. The USAGE clause may affect the radix or type of character representation of the item.

3. The USAGE IS DISPLAY clause indicates that the format of the data is a standard data format.

4. If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, the usage is implicitly DISPLAY.

5. All COMPUTATIONAL items are capable of representing a value to be used in computations and must be numeric. If a group item is described as COMPUTATIONAL, the elementary items in the group are COMPUTATIONAL. The group item itself is not COMPUTATIONAL (cannot be used in computations).

6. On the ND-100 computational, computational-1 and computational-2 items are aligned on a word boundary even if the SYNCHRONIZED clause has not been specified.

7. Computational-3 and PACKED-DECIMAL items are stored in packed decimal format.

The terms Computational, Computational-1, Computational-2, Computational-3 and PACKED-DECIMAL are explained under "Computational Options" which follows.

5.4.2.11 Computational Options

The terms COMPUTATIONAL and COMPUTATIONAL-1 define integer variables. They can be specified as 16 bit (2 byte) words or 32 bit (4 byte) words. The size depends on the maximum number of digits in the item.

The sizes of COMPUTATIONAL (COMPUTATIONAL-1) items are shown below:

	ND-100	*ND-500*
PICTURE definition is omitted (default integer)	16 Bits (2 Bytes)	32 Bits (4 Bytes)
PICTURE S9 (n) where n < = 4	16 Bits (2 Bytes)	16 Bits (2 Bytes)
PICTURE S9 (n) where n > = 5	32 Bits (4 Bytes)	32 Bits (4 Bytes)

Integer variables are always treated as if signed, even when there is no sign character(s) in the PICTURE definition.

The range of permissible values is shown below:

Length:	*Range:*
16 bits (2 bytes)	–32768 to 32767
32 bits (4 bytes)	–2147483648 to 2147483647

COMPUTATIONAL AND COMPUTATIONAL-1 VALUES

Note: For fast performance, integer fields should be used as indexes, as operands in MOVE operations and for the arithmetic statements of COBOL.

The term COMPUTATIONAL-2 is used for the description of real numbers.The internal representation will be in floating point format.

On the ND-100 the COBOL system is self-adjusting for 48 and 32 bits REAL.

On the ND-500 the size of the real item depends on the numeric length of the PICTURE definition as shown below:

PICTURE definition is omitted	32 Bits
PICTURE S9 (n) V9 (m) where n+m < = 6	32 Bits
PICTURE S9 (n) V9 (m) where n+m > = 7	64 Bits

COMPUTATIONAL-2 variables may only be used as parameters in a subroutine call, or for converting (MOVE) to or from COMPUTATIONAL-3 variables.

No VALUE clause can be specified for COMPUTATIONAL-2 items.

COMPUTATIONAL-3 items are identical to PACKED-DECIMAL items. They appear in storage in packed decimal format. This is sometimes known as BCD (binary coded decimal). The digits are each represented by 4 bits so that there are two adjacent digits per byte. The sign is contained in the rightmost 4 bits of the rightmost byte. The numbers always fill an integral number of bytes and are right justified. If necessary the leftmost half byte is filled with zero.

each decimal digit is encoded as follows:

Digit/Sign:	Binary Representation:	Hexadecimal Representation:
0 +	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	2
3	0 0 1 1	3
4	0 1 0 0	4
5	0 1 0 1	5
6	0 1 1 0	6
7	0 1 1 1	7
8	1 0 0 0	8
9	1 0 0 1	9
+	1 0 1 0	A
–	1 0 1 1	B
+	1 1 0 0	C
–	1 1 0 1	D
+	1 1 1 0	E
–	1 1 1 1	F (ND-10 only)
unsigned	1 1 1 1	F (ND-100/ND-500 only)

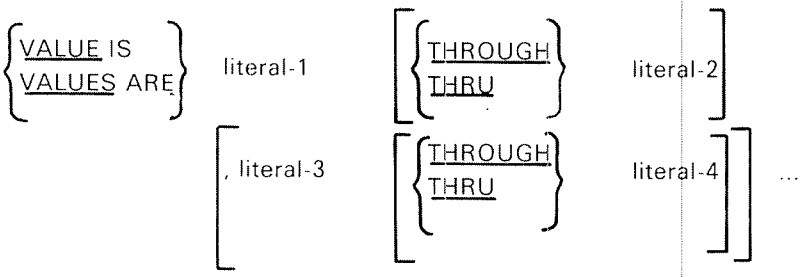
5.4.2.12 The Value Clause

The VALUE clause specifies the initial contents of a data item or the value associated with a condition name.

Format 1:

VALUE IS literal

Format 2:



The words THRU and THROUGH are equivalent.

The VALUE clause is used in condition name entries in the File, Linkage and Working-Storage sections. However, in the Working-Storage section only, it also serves to specify the initial value of any data item. The item takes this value at the beginning of the program; without the specification the value is unpredictable.

General Rules:

1. All numeric literals in the VALUE clause of an item must have a value within the range of values indicated by the PICTURE clause, and must not have a value which would require truncation of nonzero digits. Nonnumeric literals must not exceed the size indicated by the PICTURE clause. (A signed literal must have assigned with it a signed numeric PICTURE character-string.)

2. The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierachy of the item. The following rules apply:

   a) If the category of the item is numeric, all literals in the VALUE clause must be numeric. If the literal defines the value of a Working-Storage item, the literal is aligned in the data item according to the standard alignment rules. (See Standard Alignment Rules under "Classes and Categories of Data".)

   b) If the category of the item is alphabetic, alphanumeric, alphanumeric edited or numeric edited, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric. (See Standard Alignment Rules.) Editing characters in the PICTURE clause are included in determining the size of the data item (see the PICTURE clause), but have no effect on its size. Therefore, the VALUE for an edited item is presented in an edited form.

   c) Initialization takes place independent of any BLANK WHEN ZERO or JUSTIFIED clause that may be specified.

3. A figurative constant may be substituted in both Format 1 and Format 2 whenever a literal is specified.

4. The VALUE clause must not be written for a group containing items with descriptions including JUSTIFIED, SYNCHRONIZED or USAGE (other than USAGE IS DISPLAY).

5. In a condition name entry, the VALUE clause is required. The VALUE clause and the condition name itself are the only two clauses permitted in the entry. The characteristics of a condition name are implicitly those of its conditional variable.

6. Format 2 can be used only in connection with condition names and each condition name must have a separate level-88 entry. The special considerations for the use of format 1 are:

   a) The VALUE clause must not be specified for an entry that contains or is subordinate to an entry that contains a REDEFINES or OCCURS clause.

   b) If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal, and the group area is initialized without consideration for the individual elementary or group items contained with this group. The VALUE clause cannot be stated at the subordinate levels within this group.

See under the "Condition-Name Condition" for an example of the use of the VALUE clause.

5.4.2.13 The EXPORT Clause

The EXPORT clause enables separately-compiled programs to access data items.

Format:

EXPORT identifier

A communication can be established between COBOL and data areas in PLANC or common areas in FORTRAN without having to use parameters to effect the transfer.

Exported data must be defined in the Working-Storage Section.

For the corresponding IMPORT clause in the Linkage Section see Section 9.1.3.

Note:

- EXPORT/IMPORT are only allowed on 01/77 levels.
- No redefines are allowed on an identifier containing EXPORT/IMPORT, but EXPORT/IMPORT identifiers can be redefined in the usual way.
- The EXPORT and IMPORT clauses are an ND extension.

6 THE PROCEDURE DIVISION

A Procedure Division is needed in every COBOL program. It is composed of optional Declaratives and the procedures which contain the sections, paragraphs, sentences and statements used to solve a data processing problem.

Execution begins with the first statement in the Procedure Division after the Declaratives. Statements are executed in the order in which they are presented for execution (unless the rules imply a different order).

6.1 STRUCTURE OF THE PROCEDURE DIVISION

Format 1:

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

[DECLARATIVES.
{section-name SECTION [segment-number].[USE 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] ... } ...

Note: Segment-number will be treated by this compiler as comments only.

6.1.1 Declaratives

Declarative sections are preceded by the key word DECLARATIVES and followed by the key words END DECLARATIVES. They are provided for the processing of exceptional input-output conditions which cannot normally be tested by the programmer. These additional procedures are executed only at the time an I-O error occurs and cannot appear in the regular sequence of procedural statements. Therefore, they are written at the beginning of the Procedure Division in a series of Declarative sections. Each of these sections is preceded by a USE sentence which specifies the actions to be taken when the exceptional condition occurs. (See the USE statement in the I-O Statement section of the Procedure Division description.)

The key word DECLARATIVES is written on the line following the Procedure Division header.

DECLARATIVES and END DECLARATIVES, when they appear, must be followed by a period but without any text on the same lines. They must both be written in area A.

If declarative sections are specified the Procedure Division must be divided into sections.

6.1.2 Procedures

Procedures, whose names are user-defined, occur in the Procedure Division and may consist of one or more paragraphs and/or one or more sections.

A *sections* consists of a section header followed by any number (including none) of paragraphs.

A *section header* is a section name followed by the key word SECTION then a period and a space. A *section name*, which is used to identify a section, is user defined and must be unique.

A *paragraph* consists of a paragraph name, followed by a period followed by a space and then any number (or none) of sentences. A *paragraph name,* which identifies a paragraph, is user defined. It need not be unique since it can be qualified. If one paragraph in the program is contained within a section then all paragraphs must be contained in sections.

A *sentence* is made up of one or more statements followed by a period followed by a space. There are three catagories of sentence:

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

2. An *imperative sentence* is an imperative statement or series of imperative statements finally followed by a period and a space.

3. A *compiler directing* sentence is a single compiler directing statement followed by a period and a space.

A *statement* is a syntactically valid combiantion of words and symbols beginning with a COBOL verb. Statements, like sentences, are divided into three types:

1. A *conditional statement* specifies the action to be taken by the object program depending on the truth value of a condition.

2. An *imperative statement* directs that an unconditional action be taken by the object program. It may consist of a series of imperative statements.

3. A *compiler directing statement* causes a specific action to be taken by the compiler during compilation.

An *identifier* makes unique references to a data item. It may be qualified, indexed or subscripted.

6.2 ARITHMETIC EXPRESSIONS

6.2.1 Definition of an Arithmetic Expression

An arithmetic expression can be an identifier of a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Any arithmetic expression may be preceded by a unuary operator. The permissible combinations of variables, numeric literals, arithmetic operator and parentheses are given in the table below.

Those identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic may be performed.

6.2.1.1 Arithmetic Operators

There are five binary arithmetic operators and two unuary arithmetic operators that may be used in arithmetic expressions. They are represented by specific characters that must be preceded by a space and followed by a space.

Binary Arithmetic Operators:	Meaning:
+	Addition
—	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Unary Arithmetic Operators:	Meaning:
+	The effect of multiplication by numeric literal +1
—	The effect of multiplication by numeric literal —1.

6.2.1.2 Evaluation Rules

1. Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first, and within nested parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of execution is implied:

   1st — unuary plus and minus
   2nd — exponentiation
   3rd — multiplication and division
   4th — addition and subtraction

2. Parentheses are used either to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear or to modify the normal hierarchical sequence of execution in expressions where it is necessary to have some deviation from the normal precedence. When the sequence of execution is not specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right.

3. The ways in which operators, variables and parentheses may be combined in an arithmetic expression are summarized in the table where:

   a) The letter 'P' indicates a permissible pair of symbols
   b) The character '—' indicates an invalid pair
   c) 'Variable' indicates an identifier or literal.

First Symbol	*Second Symbol*				
	Variable	** / — + ***	*Unary + or —*	*(*	*)*
Variable	—	P	—	—	P
* / + — **	P	—	P	P	—
Unary + or —	P	—	—	P	—
(	P	—	P	P	—
)	—	P	—	—	P

Table of Combinations of Symbols in Arithmetic Expressions

4. An arithmetic expression may only begin with the symbol '(', '+', '—', or a variable and may only end with a ')' or a variable. There must be a one-to-one correspondence between left and right parentheses of an arithmetic expression such that each left parenthesis is to the left of its corresponding right parenthesis.

5. Arithmetic expressions allow the user to combine arithmetic operations with the restrictions on composite of operands and/or receiving data items. See, for example, syntax rules given for the ADD statement.

6.3 ARITHMETIC STATEMENTS

The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY and SUBTRACT statements. They have several common features.

1. The data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment is supplied throughout the calculation.

2. The maximum size of each operand is eighteen (18) decimal digits. The composite of operands, which is a hypothetical data item resulting from the superimposition of specified operands in a statement aligned on their decimal points must not contain more than eighteen decimal digits.

Overlapping Operands

When a sending and receiving item in an arithmetic statement or an INSPECT, MOVE, SET, STRING or UNSTRING statement share a part of their storage areas, the result of the execution of such a statement is undefined.

Multiple Results in Arithmetic Statements

The ADD, COMPUTER, DIVIDE, MULTIPLY, and SUBTRACT statements may have multiple results. Such statements behave as though they had been written in the following way:

1. A statement which performs all arithmetic necessary to arrive at the result to be stored in the receiving items, and stores that result in a temporary storage location.

2. A sequence of statements transferring or combining the value of this temporary location with a single result. These statements are considered to be written in the same left-to-right sequence that the multiple results are listed.

Incompatible Data

Except for the class condition (see the next section on Conditional Expressions) the contents of a data item are referenced in the Procedure Division and the contents of that data item are not compatible with the class specified for that data item by its PICTURE clause, then the result of such a reference is undefined.

6.3.1 Common Options

There are three options common to the arithmetic statements whose description follows.

6.3.1.1 The ROUNDED Option

If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested, the absolute value of the resultant-identifier is increased by one (1) whenever the most significant digit of the excess is greater than or equal to five.

6.3.1.2 The SIZE ERROR Option

If, after decimal point alignment, the absolute value of a result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. Division by zero always causes a size error condition. The size error condition applies only to the final results of an arithmetic operation and does not apply to intermediate results, except in the MULTIPLY and DIVIDE statements, in which case the size error condition applies to the intermediate results as well.

If the ROUNDED phrase is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action depends on whether or not the SIZE ERROR phrase is specified.

1. If the SIZE ERROR phrase is not specified and a size error condition occurs, the value of those resultant-identifier(s) affected is underlined. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation.

2. If the SIZE ERROR phrase is specfied and a size error condition occurs, then the values of resultant-identifier(s) affected by the size errors are not altered. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors occurring for other resultant-identifier(s). After execution is complete the imperative-statement in the SIZE ERROR phrase is performed.

6.3.1.3 The CORRESPONDING Option

This option allows operations to be performed on elementary items of the same name by specifying the group items to which they belong. The following rules apply:

1. Both identifiers used must be group items.

2. CORRESPONDING is equivalent to the abbreviation CORR and is valid for the MOVE statement.

3. A pair of data items, one from one group item and one from another, correspond if the following conditions are true:

   a) The two data items have the same name and the same qualifiers up to but not including the group level.

   b) At least one of the data items is an elementary item in the case of a MOVE statement with the CORRESPONDING option.

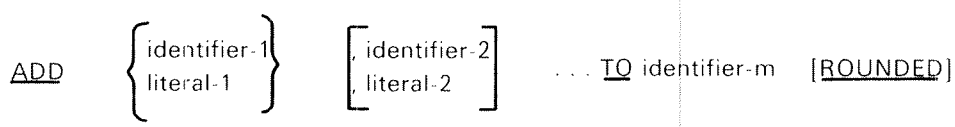
   c) The two data items do not include a REDEFINES, OCCURS, or USAGE IS INDEX clause. Such items will be ignored together with any subordinate items containing REDEFINES, CCCURS or USAGE IS INDEX clauses.

   d) The group items themselves, however, may contain or be subordinate to data items containing REDEFINES or OCCURS clauses.

6.3.1.4 The ADD Statement

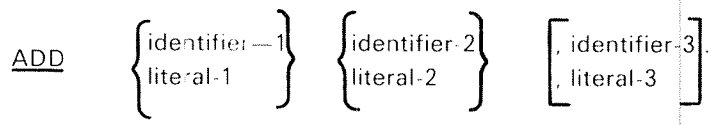
The ADD statement adds together two or more numeric operands and stores the resulting sum.

Format 1:



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

Format 2:



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

[; ON SIZE ERROR imperative-statement]

In format 1, each identifier must name an elementary numeric item.

In format 2, each identifier except those following the word GIVING must be elementary numeric items. Each identifier following the word GIVING must be either elementary numeric or numeric edited items.

Each literal must be a numeric literal.

When the TO option is used (format 1) all identifiers preceding it are added together and then added to and stored immediately in identifier-m. Then, if specified, they are added to and stored in identifier-n, etc.

When the GIVING option is used (format 2), the values of the preceding operands are added together and the sum is stored as the new value of identifier-m and (if specified) identifier-n, etc.

For the ROUNDED and SIZE ERROR options, see the preceding section on Common Options.

6.3.1.5 The COMPUTE Statement

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

Format:

COMPUTE identifier-1 [ROUNDED] [, identifier-2 [ROUNDED]] ...
= arithmetic expression [; ON SIZE ERROR imperative-statement]

Identifiers that appear only to the left of = must refer to either elementary numeric or elementary numeric edited items.

The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on receiving items in the ADD, SUBTRACT, MULTIPLY and DIVIDE statements. (When arithmetic operations must be combined use of the COMPUTE statement is more efficient than writing the separate arithmetic statements in series.)

When the COMPUTE statement is executed, the value of the arithmetic statement is calculated and then this value is stored as the new value of identifier-1, identifier-2, etc. (The arithmetic expression can be any arithmetic expression as defined earlier in this section.)

For the ROUNDED and SIZE ERROR phrases see 'Common Options' earlier in this section.

An arithmetic expression consisting of a single identifier or literal provides a method of setting the values of identifier-1, identifier-2, etc. equal to the values of that single identifier or literal.

The number of integer and decimal places provided by the compiler for intermediate results is shown in Appendix H. It is the user's responsibility to define the operands of any arithmetic statement so that they have large enough fields to provide the required accuracy of results.

6.3.1.6 The DIVIDE Statement

The DIVIDE statement divides one numeric data item into others and stores the resultant values equal to quotient and remainder.

Format 1:

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

[, identifier-3 [ROUNDED]] ... [;ON SIZE ERROR imperative-statement]

Format 2:

DIVIDE { identifier-1 / literal-1 } { INTO / BY } { identifier-2 / literal-2 } GIVING

identifier-3 [ROUNDED] [, identifier-4 [ROUNDED]]...

[;ON SIZE ERROR imperative-statement]

Format 3:

DIVIDE { identifier-1 / literal-1 } { INTO / BY } { identifier-2 / literal-2 } GIVING

identifier-3 [ROUNDED] [REMAINDER identifier-4]

[;ON SIZE ERROR imperative-statement]

Each identifier, except those following the words GIVING and REMAINDER, must be elementary numeric items. Those identifiers following GIVING and REMAINDER may also be numeric edited items.

Each literal must be a numeric literal.

In format 1 the value of identifier-1 or literal-1 is divided into the value of identifier-2 and the quotient obtained replaces this value. Similarly for identifiers 3, ... n, if specified.

In format 2, only one division takes place, the value of identifier-1 or literal-1 is divided into/by the value of identifier-2 or literal-2. The quotient is then stored in identifier-3 and (if specified) identifier-4, etc.

In format 3, the division process is as for format 2 except that the quotient is stored in identifier-3 and the value of the remainder in identifier-4.

For the ROUNDED and SIZE ERROR options, see the preceding section on Common Options.

6.3.1.7 The MULTIPLY Statement

The MULTIPLY statement computes the product of two numeric data items and stores it.

Format 1:

MULTIPLY { identifier-1 / literal-1 } BY identifier-2 [ROUNDED]

[, identifier-3 [ROUNDED]] ... [;ON SIZE ERROR imperative-statement]

Format 2:

MULTIPLY { identifier-1 / literal-1 } BY { identifier-2 / literal-2 }

GIVING identifier-3 [ROUNDED] [, identifier-4 [ROUNDED]] ...

[;ON SIZE ERROR imperative-statement]

Each literal must be a numeric literal.

Each identifier must be numeric elementary item, except that identifiers following the word GIVING in format 2 may also be elementary numeric items.

In format 1, identifier-2 is replaced by the product of it and the first operand. This process is continued for all subsequent identifiers.

When format 2 is used the value of identifier-1 or numeric-literal-1 is multiplied by the value of the second operand. The result is stored in identifier-3, identifier-4, etc.

6.3.1.8 The SUBTRACT Statement

The SUBTRACT statements subtracts one, or the sum of two or more, or more numeric data items from one or more items and stores the results.

Format 1:

SUBTRACT { identifier-1 / literal-1 } [, identifier-2 / , literal-2] ...

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

[;ON SIZE ERROR imperative-statement]

Format 2:

SUBTRACT { identifier-1 / literal-1 } [, identifier-2 / , literal-2] ...

FROM { identifier-m / literal-m }

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

[;ON SIZE ERROR imperative-statement]

Each identifier must represent a numeric elementary item except when following the word GIVING when it may also be an elementary numeric edited item.

In format 1 the identifiers or literals preceding FROM are added together and subtracted from identifiers m, n, ... in turn. After each subtraction the results are stored in these identifiers m, n, ...

In format 2, the identifier or literals preceding FROM are added together and subtracted from identifier-m or literal-m. The result of the subtraction is stored as the new value of identifier-n and any other specified identifiers.

6.4 CONDITIONAL EXPRESSIONS

Conditional expressions identify conditions that are tested to enable the object program to select between alternate paths of control depending upon the truth value of the condition. Conditional expressions are specified in the IF, PERFORM and SEARCH statements. There are two categories of conditions associated with conditional expressions: simple conditions and complex conditions.

SIMPLE CONDITIONS

The simple conditions are the relation, class, condition-name and sign conditions. A simple condition has a truth value of 'true' or 'false'.

RELATION CONDITION

A relation condition causes a comparison of two operands, each of which may be the data item referenced by an identifier, a literal, or the value resulting from an arithmetic expression. A relation condition has a truth value of 'true' if the relation exists between the operands. Comparison of two numeric operands is permitted regardless of the formats specified in their respective USAGE clauses. However, for all other comparisons the operands must have the same usage. If either of the operands is a group item, the nonnumeric comparison rules apply.

The format of a relation condition is as follows:

```
{identifier-1       }   {IS [NOT] GREATER THAN}   {identifier-2       }
{literal-1          }   {IS [NOT] LESS THAN   }   {literal-2          }
{arithmetic-expression-1}{IS [NOT] EQUAL TO   }   {arithmetic-expression-2}
                        {IS [NOT] >           }
                        {IS [NOT] <           }
                        {IS [NOT] =           }
```

NOTE: The required relational characters '>', '<', and '=' are not underlined to avoid confusion with other symbols such as '≥' (greater than or equal to).

COMPARISION OF NUMERIC OPERANDS

For operands whose class is numeric (refer to the Data Division, Classes and Categories of Data) a comparison is made with respect to the algebraic value of the operands. The length of the literal or arithmetic expession operands, in terms of number of digits represented, is not significant. Zero is considered an unique value regardless of the sign.

Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered positive for purposes of comparison.

COMPARISON OF NONNUMERIC OPERANDS

For nonnumeric operands, or one numeric and one nonnumeric operand, a comparison is made with ND's standard character set. If one of the operands is numeric it must be an integer data item or integer literal and the following rules apply:

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

b. If the nonnumeric item is a group item, the numeric item is treated as though it were moved to a group item of the same size, and the contents of this group were compared to the nonnumeric operand.

c. A non-integer numeric operand cannot be compared to a nonnumeric operand.

The ALPHABETIC test cannot be used with an item whose data description describes the item as numeric. The item 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.

The size of an operand is the total number of characters contained in it.

If the operands are *equal* in size:

Characters in corresponding positions are compared, beginning with the leftmost character. If a pair of unequal characters is encountered, they are tested to ascertain their relative positions in the collating sequence. The operand having the character higher in the sequence is considered to be the greater operand.

If the operands are *unequal* in size:

The comparison is made as if the shorter operand were extended to the right with enough spaces to make the operands of equal size.

CLASS CONDITION

The class condition determines whether the operand is alphabetic or numeric. Its format is:

identifier IS [NOT] { NUMERIC / ALPHABETIC }

The identifier is determined to be numeric if its contents consist only of a combination of the digits 0 through 9.

If its PICTURE does not contain an operational sign, then the identifier is considered as numeric if the contents are numeric and an operational sign is not present. Otherwise, if its PICTURE contains an operational sign, the identifier is considered to be numeric if it is an elementary item having numeric contents together with the presence of an operational sign.

Valid operations signs are:

For items described with the SIGN clause -

+ (53 octal) and — (55 octal)

The embedded operation signs -

+0 to +9	173, 101 to 111 (octal)
—0 to —9	175, 112 to 122 (octal)

For COMPUTATIONAL-3 items, see under Computational Options.

The NUMERIC test is not valid for alphabetic items or for group items which have operational signs present in items subordinate to them.

CONDITION-NAME CONDITION (CONDITIONAL VARIABLE)

This condition determines whether a conditional variable has a value equal to any of the values(s) associated with the condition-name. Its format is:

condition-name

The use of this condition is as an abbreviation for the relation condition and the rules for comparing a conditional variable with a condition-name are the same as specified for the relation condition.

If the condition-name is associated with a range or ranges of values, then the conditional variable is tested to determine whether or not its value falls in this range, 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.

As an example of its use, if the following is specified:

```
05  TYPE-REC PIC  X.
  88  TYPE-1  VALUE  A THRU  F.
  88  TYPE-2  VALUE  H.
  88  TYPE-3  VALUE  J  THRU  Z.
```

(Where TYPE-REC is a conditional variable) then, to determine a type classification of a record, the code:

```
IF TYPE-1 . . .
```

Can cause a branch for values of A, B, C, D, E or F. (Refer to VALUE clause in the Data Division and to 'Comparison of Nonnumeric Operands' earlier in this section.)

SIGN CONDITION

The sign condition determines whether or not the algebraic value of an arithmetic expression is less than, greater than, or equal to zero. The general format for a sign condition is as follows:

arithmetic-expression IS [NOT] { POSITIVE / NEGATIVE / ZERO }

When used, 'NOT' and the next key word specify one sign condition that defines the algebraic test to be executed for truth value; e.g., 'NOT ZERO' is a truth test for a nonzero (positive or negative) value. An operand is positive if its value is greater than zero, and zero if its value is equal to zero. The arithmetic expression must contain at least one reference to a variable.

COMPLEX CONDITIONS

A complex condition is formed by combining simple conditions, combined conditions and/or complex conditions with logical connectors (logical operators 'AND' and 'OR') or negating these conditions with logical negation (the logical operator 'NOT'). The truth value of a complex condition, whether parenthesized or not, is that which results from the interaction of all the logical operators on the individual values of simple conditions, or the intermediate values of conditions logically connected or logically negated.

The logical operators and their meanings are:

Logical Operator:	*Meaning:*
AND	Logical conjunction; the truth value is 'true' if both conditions are true; 'false' if one or both conditions is false.
OR	Logical inclusive OR; the truth value is 'true' if one or both of the conditions is true; 'false' if both conditions are false.
NOT	Logical negation or reversal of truth value; the truth value is 'true' if the condition if false; 'false' if the condition is true.

The logical operators must be preceded by a space and followed by a space.

NEGATED SIMPLE CONDITIONS

A simple condition (see earlier in this section) is negated through the use of the logical operator 'NOT'. The negated simple condition effects the opposite truth value for a simple condition. Its format is:

NOT simple condition

COMBINED CONDITIONS

A combine condition results from connecting conditions with one of the logical operators 'AND' or 'OR'. The format is:

condition $\left[\left\{\begin{matrix}\underline{\text{AND}}\\ \underline{\text{OR}}\end{matrix}\right\}\right.$ condition$\left.\right\}$ . . .

Where 'condition' may be:

1. A simple condition, or

2. A negated simple condition, or

3. A combined condition, or

4. A negated combined condition; i.e., the 'NOT' logical operator followed by a combined condition enclosed within parentheses, or

5. Combinations of the above, specified according to the rules summarized in the following table, Combinations of Conditions, Logical Operators, and Parentheses.

Although parentheses need never be used when either 'AND', 'OR' and 'NOT' is used. The table indicates the ways in which conditions and logical operators may be combined and parenthesized. There must be a one-to-one correspondence between left and right parentheses such that each left parentheses is to the left of its corresponding right parentheses.

Given the following element	*Location in conditional expression*		*In a left-to-right sequence of elements:*	
	First	*Last*	*Element, when not first, may be immediately preceded by only:*	*Element, when not last, may be immediately followed by only:*
simple-condition	Yes	Yes	OR, NOT, AND, (	OR, AND,)
OR or AND	No	No	simple-condition,)	simple-condition, NOT, (
NOT	Yes	No	OR, AND, (	simple-condition, (
(	Yes	No	OR, NO, AND, (	simple-condition, NOT, (
)	No	Yes	simple-condition,)	OR, AND,)

TABLE OF COMBINATIONS OF CONDITIONS, LOGICAL OPERATORS, AND PARENTHESES

Thus, the element pair 'OR NOT' is permissible while the pair 'NOT OR' is not permissible; 'NOT (' is permissible while 'NOT NOT' is not permissible.

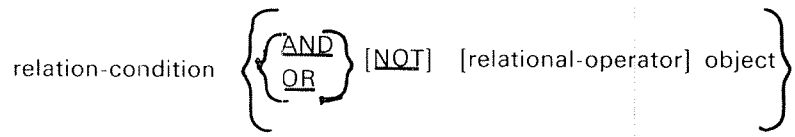
ABBREVIATED COMBINED RELATION CONDITIONS

When simple or negated simple relation conditions are combined with logical connectives in a consecutive sequence such that a succeeding relation condition contains a subject or subject and relational operator that is common with the preceding relation condition (and no parentheses are used within such a consecutive sequence), any relation condition except the first may be abbreviated by:

(1) The omission of the subject, or

(2) The omission of the subject and relational operator.

The format for an abbreviated combined relation condition is:



Within a sequence of relation conditons both of the above forms of abbreviation may be used. The effect of using such abbreviations is as if the last preceding stated subject were inserted in place of the omitted subject, and the last stated relational operator were inserted in place of the omitted relational operator. The result of such implied insertion must comply with the rules given in the table, Combinations of Conditions, Logical Operators and Parentheses, shown above. This insertion of an omitted subject and/or relational operator terminates once a complete simple condition is encountered within a complex condition.

The interpretation of the word 'NOT' in an abbreviated combined relation condition is as follows:

(1) If the word immediately following 'NOT' is 'GREATER', '>', 'LESS', '<', EQUAL, '=', then the 'NOT' participates as a part of the relational operator; otherwise

(2) The 'NOT' is interpreted as a logical operator and, therefore, the implied insertion of subject or relational operator results in a negated relation condition.

Some examples of abbreviated combined and negated combined relation conditions and expanded equivalents follow.

Abbreviated Combined Relation Condition	*Expanded Equivalent*
a > b AND NOT < c OR d	((a > b) AND (a NOT < c))OR (a NOT < d)
a NOT EQUAL b OR c	(a NOT EQUAL b) OR (a NOT EQUAL c)
NOT a = b OR c	(NOT (a = b))OR (a = c)
NOT (a GREATER b OR < c)	NOT ((a GREATER b) OR (a < c))
NOT (a NOT > b AND c AND NOT d)	NOT ((((a NOT > b) AND (a NOT > c)) AND (NOT (a NOT > d))))

CONDITION EVALUATION RULES

Parentheses may be used to specify the order in which conditions are evaluated when it is required to depart from the implied evaluation sequence. In this case, logical evaluation proceeds 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 parentheses are not used then the evaluation order is:

   1. Arithmetic expressions

   2. Simple conditions in the order -

      relation
      class
      condition-name
      sign

   3. Negated simple-conditions in the order as in 2.

   4. Combined conditions in the order -

      OR
      AND
      NOT

   5. Negated combined conditions in the order as in 4.

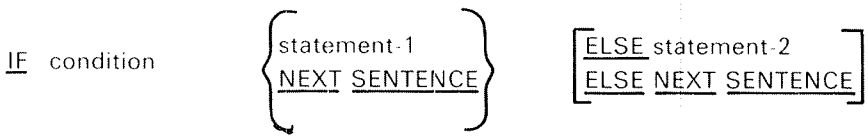
Consecutive operands at the same hierarchical level are evaluated from left to right.

6.5 CONDITIONAL STATEMENTS

6.5.1 The IF Statement

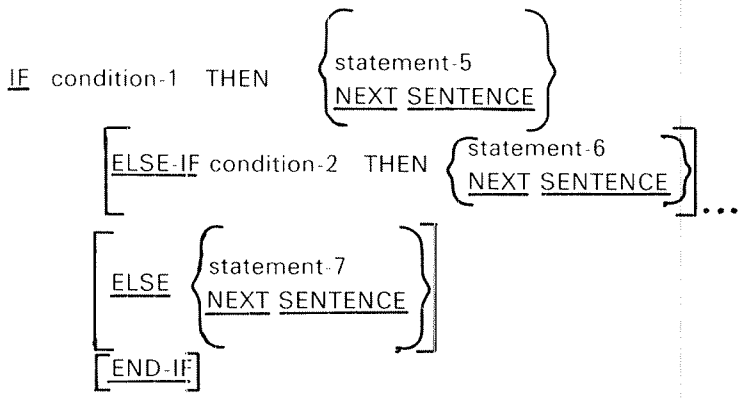
The IF statement causes a condition to be evaluated. The subsequent execution sequence depends upon whether the condition is true or false. The general format is:

Format 1:



Format 2 (An ND Extension):

Format 3 (An ND Extension):



General Rules for Format 1:

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

   a. Statement-1, if specified, is executed. If this contains a procedure-branching statement, control is transferred according to the rules of that statement. If it does not, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

   b. If the NEXT SENTENCE phrase is specified instead of statement-1, the ELSE phrase, if present, is ignored and control passes to the next executable sentence.

2. If the condition tested is *false*, one of the following occurs:

   a. ELSE statement-2, if specified, is executed. If this statement contains a procedure-branching statement, control is transferred according to the rules for that statement. Otherwise control is passed to the next executable sentence.

   b. ELSE NEXT SENTENCE, if specified, is executed, i.e. statement-1, if present, is ignored and control passes to the next executable sentence.

   c. If ELSE NEXT SENTENCE is omitted, control passes to the next executable sentence.

3. Statement-1 and/or statement-2 may contain an IF statement. In this case the statement is said to be nested. Statements 1 and 2 represent either an imperative statement or a conditional statement. Either of these may be followed by a conditional statement.

4. The ELSE NEXT SENTENCE option may be omitted if it immediately precedes the terminal period of the sentence.

General Rules for Format 2:

1. Statements 3 and 4 represent imperative statements.

2. If the condition is true and the ELSE clause is omitted, then if statement-3 has been coded, this statement together with any further imperative statements preceding the sentence terminator, will be executed. Control is then passed implicitly to the next sentence unless a GO-TO procedure-name appears in statement-3. If the condition is true and NEXT SENTENCE is coded, control passes explicitly to the next sentence.

3. If the condition is true and the ELSE clause is present then statement-4 (together with any further imperative statements preceding the sentence terminator) or the NEXT SENTENCE of this clause is executed. If the ELSE clause is absent, control passes to the next sentence following the END-IF.

4. No period character (.) should occur between the IF and END-IF verbs inclusively.

General Rules for Format 3:

1. Statements 5, 6 and 7 are imperative statements.

2. If the ELSE-IF clause is omitted then the rules are as for format 2. (Except that statement-5 should be substituted for statement-3 and statement-7 for statement-4.)

3. If condition-1 is false, then if condition-2 is true, then the rules are as for format-2 if statement-6 is substituted for statement-3 and statement-7 for statement-4.

6.5.1.1 Nested IF Statements

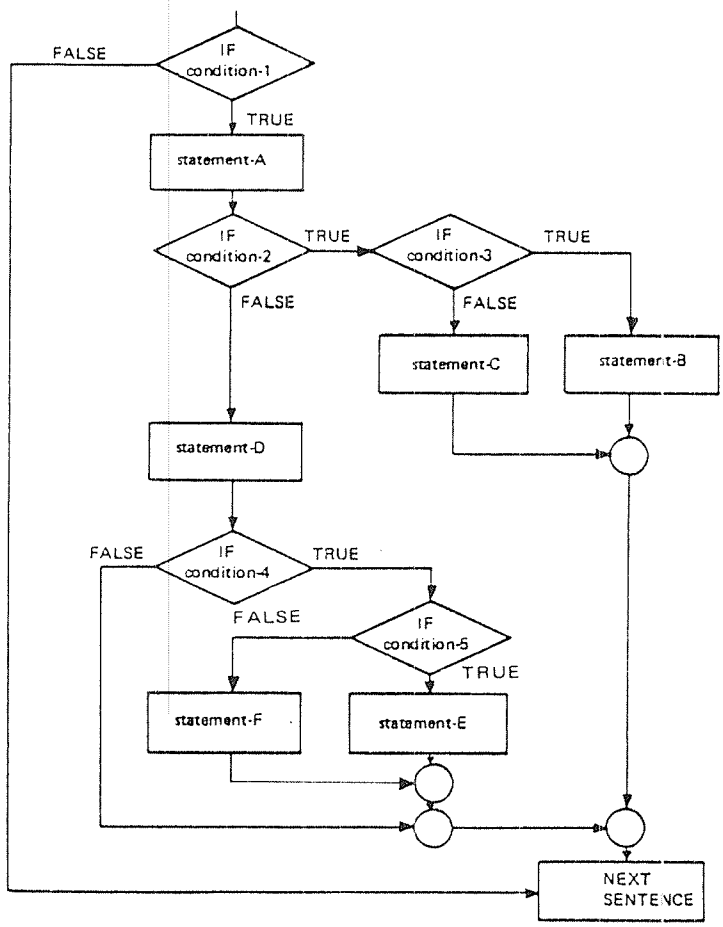
The presence of one or more IF statements within an initial IF statement constitutes a "nested" IF statement. Statements 1 and 2 may consist of one or more imperative statements and/or a conditional statement. If an IF statement appears as the whole or part of statements 1 or 2 it is said to be nested.

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

The structure of a possible nested IF statement may be exemplified as follows:

```
IF condition-1   statement-A
    IF conditon-2
        IF condition-3   statement-B
        ELSE statement-C
    ELSE statement-D
        IF condition-4
        IF condition-5   statement-E
            ELSE statement-F.
```

The Flowchart for this example would appear as:



6.5.2 **The DO Statement** (An ND-Extension)

A DO statement specifies a loop which can be used for coding iterative procedures. There are two basic formats:

Format 1:

DO [sentence] [(WHILE condition) sentence] . . . END-DO

Format 2:

DO FOR identifier FROM { identifier / integer-1 } [BY integer-2]

TO { identifier / integer-3 } [sentence] [(WHILE condition) statement] . . . END-DO

In format 2, the identifier must be a numeric item whose PICTURE specification does not contain a decimal point. i, j, and k are the initial, incremental, and terminal parameters respectively and they must be integer. The incremental parameter should be greater than or equal to 1, if it is not present it is assumed to be 1 (one).

At execution time the identifier takes the value of the initial parameter, and the loop is performed until either the initial parameter is greater than the terminal parameter, or until either the condition in the WHILE phrase (if present) is no longer true. Control then passes to the next executable statement following the corresponding END-DO statement.

In format 1, the identifier must be specified as in format 2. If the WHILE condition phrase does not appear the DO-loop may be regarded as an infinite loop (see Example 3 for an example of its use).

The WHILE condition phrase which appears in both formats, may also appear any number of times within the DO-loop. DO-loops may be nested up to 50 levels. Any DO-loop may be left via the EXIT verb. (See also EXIT-DO and EXIT-ALL-DO in section 6.8.3.)

EXAMPLE 1.

```
DO FOR N FROM 1 BY 1 TO 50
MOVE CORRESPONDING MASTER-REC(N) TO OUT-REC(N).
WRITE OUTRECT(N).
END-DO.
```

EXAMPLE 2.

```
DO WHILE I < 100.
   WHILE M = N.
   WHILE P NOT EQUAL R OR S.
WRITE OUT-FILE.
END-DO.
```

EXAMPLE 3.

```
DO.
* read file with unknown number of records
READ FILE IN-FILE AT END GO TO 1000.
END-DO
```

6.6 DATA MANIPULATION STATEMENTS

6.6.1 Screen Handling Facilities

Screen Handling for COBOL is an ND Extension for which four of the Data Manipulation Statements can be used. These are ACCEPT (format 3), ACCEPT-ERROR, BLANK, and DISPLAY (format 2), and they are described individually below. Section 6.6.2 provides a few examples of screen-handling in which their function and interaction is demonstrated.

These features can be used with all terminals which are suitable for the ND editors (Notis-WP, PED, ...).

6.6.1.1 The ACCEPT Statement

The ACCEPT statement causes low volume data to be made available to the specified identifier.

Format 1 - Data Transfer:

ACCEPT identifier [FROM mnemonic-name]

Format 2 - System Information Transfer:

ACCEPT identifier FROM { DATE | DAY | TIME | CPU-TIME }

Format 3 — Screen Handling:

```
ACCEPT position spec. identifier [WITH [BEEP]
                                       [SPACE-FILL]
                                       [LENGTH-CHECK]
                                       [AUTO-SKIP]
                                       [PROMPT]
                                       [BLANK-WHEN-ZERO]
                                       [MUST]
                                       [UPDATE]
                                       [INVISIBLE]
                                       [INVERSE-VIDEO]
                                       [BLINK]
                                       [UNDERLINE]
                                       [LOW-INTENSITY]
                                       [UPPER-CASE]
                                       [NORMAL]
                                       [UP Label]
                                       [DOWN Label]
                                       [HOME Label]
                                       [EXIT Label]
                                       [LEFT Label]
                                       [RIGHT Label]
                                       [CONTROL Label] ]
```

where Label is a paragraph or a section name, and

identifier is the name of the receiving field.

Position specifier is the screen position defined as:

(Line, column)

both line and column being defined by:

```
{ identifier [(±) integer] }
{       integer            }
```

Format 1 is used to transfer data from an input-output device into identifier. If the FROM option is omitted then the input device is assumed to be the system console in the case of RT-users, and the CR terminal in the case of Time-sharing and Batch users. When running batch or mode files in background mode, data is accepted from the next line on the respective file. (IF the FROM option is present then the mnemonic-name is treated as comments only.)

Format 2 is used to transfer system information (DAY, DATE, TIME, CPU-TIME) into identifier according to the rules of the MOVE statement.

DATE is composed of a sequence of data elements as follows:

> 2 digits for year of century, 2 digits for month of year, 2 digits for day of month. Therefore, September 1, 1980 would be expressed as 800901.

DAY has the sequence of data elements:

> 2 digits for year of century, 3 digits for day of year. Thus, September 1, 1980 is expressed as 80245.

TIME is composed of the data elements hours, minutes, seconds and hundreths of a second. For example, 2:41 p.m. would be expressed as 14410000.

CPU-TIME consists of the data element CPU-time expressed in milliseconds.

In format 3, the receiving field (identifier) is described by a PICTURE or USAGE specification. The data input field is a string of character positions starting at the location indicated by the position specifier. Valid data which may be entered is governed by the rules for the associated PICTURE specification (see the Picture Clause in Section 5.4.2.5).

The identifier may have its USAGE described as COMPUTATIONAL (see Section 5.4.2.11). In this case the size of the field for single-word items is 5 + a sign position, and for double-word items the size is 10 + the sign position.

The identifier may also have its USAGE described as COMPUTATIONAL-3.

Format 3 is used to accept data into a field from a CR-terminal. The options in the WITH phrase which describe the appearance of the field on the screen, can appear in any order or combination. However in some cases the type of options which are operative simultaneously will be terminal-dependent.

The effects of each are as follows:

BEEP will sound the terminal's audio alarm when the system is ready to ACCEPT the field.

SPACE-FILL is for use with numeric fields. Where the identifier has a PICTURE specification of 9's only, leading zeros are set to blanks. (On the screen only.)

LENGTH-CHECK causes the entry of a field terminator to be ignored until each input position has been operated upon.

AUTO-SKIP specifies that when an input field has filled by the operator, the field will be terminated automatically.

PROMPT results in the data input field on the screen being set so that all positions contain the period character (".") before input is accepted.

UPDATE will initialize the data input field with the initial contents of the receiving field. This data is then treated identically to that keyed-in by the operator, and editing can be performed. UPDATE and PROMPT can be used together on the same ACCEPT statement.

INVISIBLE will prevent the data entered into the input field from being displayed on the screen. This may be required for security reasons, such as when keying-in passwords etc.

INVERSE-VIDEO produces a bright background in the area allotted for the display of identifier.

BLINK causes the display of identifier to flash on and off.

UNDERLINE produces underlining of identifier.

LOW-INTENSITY results in a display of reduced intensity.

NORMAL resets the effect of a previous INVERSE-VIDEO, LOW-INTENSITY, BLINK or UNDERLINE.

MUST means that some data must be entered into the field of the ACCEPT statement before it can be left.

BLANK-WHEN-ZERO causes the item to be displayed as all blanks if its value is zero.

UP, DOWN, HOME, EXIT, LEFT, and RIGHT represent control keys which are terminal-dependent and which are used for moving the cursor between specific data input fields. These data input fields are identified by the name of the paragraph or section in which they occur in the Procedure Division.

CONTROL Label, which must be the last option coded on an ACCEPT statement, provides the user with an opportunity to test for errors of his own definition. On entering carriage return, the section or paragraph with the label "Label" receives control. If the user-defined error is found then an ACCEPT-ERROR statement following the test will return control to the ACCEPT statement, at the end of the section or paragraph, which will not have been "accepted". The field must now be re-entered. If the error has not been detected then return is to the next statement following the ACCEPT.

Carriage return (CR) acts as a terminator character. If LENGTH-CHECK has not been coded it terminates the ACCEPT and the cursor automatically moves to the beginning of the next data input field.

The carriage return may be used at any position in the data input field unless LENGTH-CHECK has been coded with the associated ACCEPT statement.

Editing within data input fields of alphanumeric types before termination of the ACCEPT statement may be performed using CRTL A to delete a single character at a time, CRTL E to insert characters, CTRL Q to delete all characters and left and right arrows to move the cursor inside the field.

CTRL A and CRTL Q may be used also in numeric fields.

Upon termination of the ACCEPT statement, data is transferred to the receiving field and edited according to the rules of the corresponding PICTURE specifications. With numeric fields there is an automatic display after "acceptance" of data input.

6.6.1.2 The ACCEPT-ERROR Statement

Format:

ACCEPT-ERROR

This statement is used in conjunction with an ACCEPT statement having the option CONTROL label. ACCEPT-ERROR is coded within the section or paragraph designated by 'label'. If a user-defined error has been detected, then ACCEPT-ERROR causes a return to the ACCEPT statement, at the end of the statement or paragraph, which will not have been 'accepted'. The field must now be re-entered. If the user-defined error is not detected, control passes to the next statement following the ACCEPT statement.

6.6.1.3 The BLANK Statement

The BLANK statement causes the whole or part of the screen to be erased.

Format 1:

BLANK SCREEN

Format 2:

BLANK [LINE / LINES] n_1 [TO n_2] [COLUMN n_3 TO n_4]

where i, j, k, n_1, n_2, n_3, and n_4 must be integers or identifiers defined with no decimal point.

With format 1, the entire screen is erased and the cursor is placed in the home position (line 1, column 1). Format 2 will blank out the line n_1 to n_2 inclusively, between columns n_3 and n_4 inclusively.

6.6.1.4 The DISPLAY Statement

The DISPLAY statement causes low volume data tc be transferred to the appropriate hardware device. It also has some screen handling facilities for convenient data presentation (viz., the FRAME and the FULL-BAR/SPARSE-BAR options).

Format 1:

DISPLAY { identifier-1 / literal-1 } [, identifier-2 / , literal-2] ... [WITH NO ADVANCING]

[UPON mnemonic-name]

Literal-1 and literal-2 may be any figurative constant, except ALL.

The operand(s) are transferred to the system output device with conversion, if necessary.

The UPON option has no effect and exists for syntax reasons only.

If the WITH NO ADVANCING phrase is specified, the system output device will not advance one line on the page before displaying the output. Otherwise, automatic advancement of one line will occur.

Format 2:

DISPLAY position spec. { identifier-3 / literal-3 } [identifier-4 / literal-4] [WITH [BEEP]
[SPACE-FILL]
[INVERSE-VIDEO]
[BLINK]
[UNDERLINE]
[LOW-INTENSITY]
[NORMAL]
[AUTO-ERASE]
[PROMPT]
[BLANK-WHEN-ZERO]]

Position spec., the screen position, is defined as:

(line, column)

both line and column being defined by:

{ identifier [(±) integer] / integer }

Format 2, which forms part of Screen Handling, displays data on a video terminal. Messages or the contents of a data item can appear on the screen with various forms of visual emphasis. The data consists of either literal-3 or identifier-3 and the display is described by the options listed in the WITH phrase. These options may appear in any order. However, in some cases the number of options which may appear simultaneously will be terminal-dependent.

They have the following meanings:

BEEP causes an audible alarm to sound when the DISPLAY statement is initialized.

SPACE-FILL is for use where identifier-4 describes a numeric field. If the PICTURE specification contains only 9's, leading zeros are set to blanks (on the screen only).

INVERSE-VIDEO produces a bright background in the area allotted for the display of identifier-4 or literal-4. The characters themselves appear at the normal background intensity.

BLINK causes the display of identifier-4 or literal-4 to flash on and off.

UNDERLINE produces underlining of literal-4 or the contents of identifier-4 when they are displayed on the screen.

LOW-INTENSITY causes a display of reduced intensity.

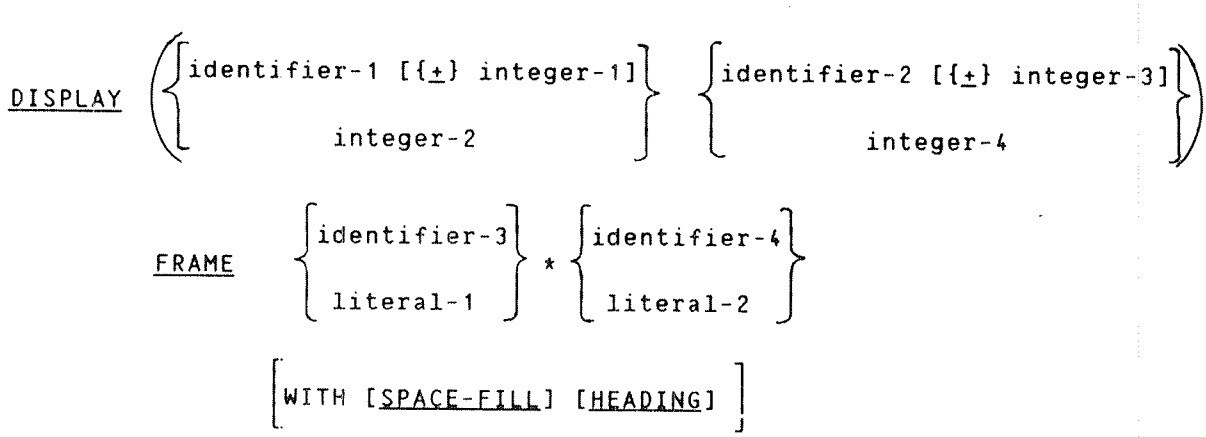
NORMAL resets the effect of a previous INVERSE-VIDEO, LOW-INTENSITY, BLINK, or UNDERLINE.

AUTO-ERASE. When the first character of a following ACCEPT statement is entered, all fields coded with AUTO-ERASE, of which there may be up to 16, will disappear automatically.

PROMPT. If the field is all zeros or all spaces, the prompt character period ('.') will appear in each position instead.

BLANK-WHEN-ZERO. Leading zeros in a field will be replaced by blanks (spaces). If the value of the field is zero, the field will contain all spaces.

Format 3:



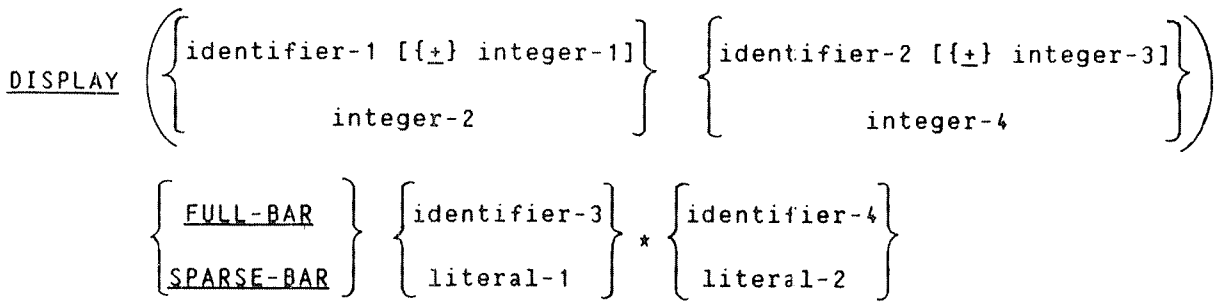
Format 3 is used to draw frames around selected areas of the screen. The part within the first parenthesis is a position specification, as in the previous format. The specified position is taken to be the upper left corner of a frame of the size given after the FRAME phrase. The first number after FRAME gives the number of lines down from the specified point that the frame will reach. The second number gives the number of columns that the frame will reach to the right of the specified point.

The format has two additional options:

SPACE-FILL, erases the interior of the frame, i.e., it writes spaces into each character position inside it.

HEADING makes COBOL draw a line segment across the third line inside the frame, thus making room for a heading to be written into the second line inside the frame.

Format 4:



Format 4 allows COBOL to draw vertical histogram bars from available data. It has a position specification part, like the previous formats. In format 4, however, the position specified inside the parenthesis is the lower left corner of the bar. There are two optional shadings, of which one must be selected:

FULL-BAR, giving the densest shading.

SPARSE-BAR, giving a half-tone shading.

The size of the bar must be specified after the shading option. The first of the two numbers defines the height of the bar, the second defines its width. The height of the bar may be up to four times the number of lines available for it. That means that a bar of height 4 is one line high, while a bar of height 88 may reach from the bottom to the top of a 22-line screen.

6.6.2 Screen Handling Examples

This section shows five simple programs to illustrate some of the features of ND COBOL screen handling. A description of the statements used will be found in section 6.6.1

EXAMPLE 1.

```
ND-100 COBOL COMPILER  -  9 JUN 1982      TIME: 16.55.41   DATE: 82.07.13
    SOURCE FILE: COBF:TEXT
    OBJECT FILE: COBF

      1
      2
      3         IDENTIFICATION DIVISION.
      4         PROGRAM-ID. DIAGONALS.
      5
      6       * This program produces a pattern of two crossing diagonals
      7       * which appear as blanked areas on a filled-in background.
      8
      9
     10         ENVIRONMENT DIVISION.
     11         CONFIGURATION SECTION.
     12         SOURCE-COMPUTER. NORD-100.
     13         OBJECT-COMPUTER. NORD-100.
     14
     15         DATA DIVISION.
     16
     17         WORKING-STORAGE SECTION.
     18         01  M              PIC 99 VALUE ZERO.
     19         01  J              PIC 99 VALUE 78.
     20         01  I              PIC 999 VALUE ZERO.
     21         01  N              PIC 99 VALUE ZERO.
     22         PROCEDURE DIVISION.
     23         100.
     24             BLANK SCREEN.
     25         1200.
     26             DO FOR N FROM 1 BY 1 TO 80.
     27             DO FOR I FROM 1 BY 1 TO 25.
     28             DISPLAY (I, N) "#".
     29             END-DO.
     30             END-DO.
     31         1300.
     32             DO FOR N FROM 2 BY 2 TO 25.
     33             MOVE N TO I.
     34             ADD N TO I.
     35             ADD N TO I.
     36             MOVE I TO M.
     37             ADD 3 TO M.
     38             BLANK LINE N COLUMN I TO M.
     39             END-DO.
```

```
40          1400.
41              DO FOR N FROM 2 BY 2 TO 25.
42              SUBTRACT 6 FROM J.
43              MOVE J TO M.
44              ADD 3 TO M.
45              BLANK LINE N COLUMN J TO M.
46              END-DO.
47          1700.
48              STOP RUN.
49
```

*** NO ERROR MESSAGES ***

EXAMPLE 2.

```
ND-100 COBOL COMPILER  -  9 JUN 1982      TIME: 14.34.03   DATE: 82.07.15
    SOURCE FILE: FORMS:TEXT
    OBJECT FILE: FORMS

      1
      2        IDENTIFICATION DIVISION.
      3        PROGRAM-ID. FORMS.
      4
      5       *This program shows how a form might be created containing
      6       *information - in this case names, addresses, and codes.
      7       *The contents are displayed and the opportunity is provided
      8       *to "accept" an update for each entry. It is possible to move
      9       *between the fields using control keys as individually coded
     10       *in the program with each ACCEPT statement.
     11
     12
     13        ENVIRONMENT DIVISION.
     14        CONFIGURATION SECTION.
     15        SOURCE-COMPUTER. NORD-100.
     16        OBJECT-COMPUTER. NORD-100.
     17
     18        DATA DIVISION.
     19
     20        WORKING-STORAGE SECTION.
     21        01  FRAME.
     22         02  HORIZ-LINE PIC X(80) VALUE "-----------------------------
     23       -    "-----------------------------------------------".
     24         02  NAM  PIC X(15) OCCURS 10 TIMES.
     25
     26         02  ADDR PIC X(30) OCCURS 10 TIMES.
     27
     28         02  CODE PIC 999 OCCURS 10 TIMES.
     29
     30        01  N PIC 99 VALUE ZERO.
     31        01  M PIC 99 VALUE ZERO.
     32
     33        PROCEDURE DIVISION.
     34
     35        5.  MOVE "ROSE COTTAGE" TO ADDR(1).
     36            MOVE "10 STRAWBERRY HILL" TO ADDR(2).
     37            MOVE "THE OLD MILL" TO ADDR(3).
     38            MOVE "132 OXFORD ROAD" TO ADDR(4).
     39            MOVE "1 DONNINGTON SQUARE" TO ADDR(5).
     40            MOVE "5 WHITE HORSE LANE" TO ADDR(6).
     41            MOVE "TUDOR LODGE" TO ADDR(7).
     42            MOVE "3 DEER LEAP WOOD" TO ADDR(8).
     43            MOVE "RIVERSIDE HOUSE, HENLEY" TO ADDR(9).
     44            MOVE "THE BARN, ABBOTS ANN" TO ADDR(10).
```

```
45
46          15.
47              MOVE "ANDERSON" TO NAM(1).
48              MOVE "ARCHER" TO NAM(2).
49              MOVE "BROWN" TO NAM(3).
50              MOVE "CARTER" TO NAM(4).
51              MOVE "EVANS" TO NAM(5).
52              MOVE "HYDE" TO NAM(6).
53              MOVE "LEWIS" TO NAM(7).
54              MOVE "NORTON" TO NAM(8).
55              MOVE "RUSSELL" TO NAM(9).
56              MOVE "WOOD" TO NAM(10).
57
58          20.
59              MOVE "505" TO CODE(1).
60              MOVE "399" TO CODE(2).
61              MOVE "002" TO CODE(3).
62              MOVE "900" TO CODE(4).
63              MOVE "417" TO CODE(5).
64              MOVE "015" TO CODE(6).
65              MOVE "666" TO CODE(7).
66              MOVE "818" TO CODE(8).
67              MOVE "077" TO CODE(9).
68              MOVE "202" TO CODE(10).
69
70          30.
71              BLANK SCREEN.
72
73        * INSERT FORM HEADERS
74
75          35.
76              DISPLAY (1, 1) HORIZ-LINE.
77              DISPLAY (2, 1) "|".
78              DISPLAY (2, 10) "NAME".
79              DISPLAY (2, 31) "|".
80              DISPLAY (2, 40) "ADDRESS".
81              DISPLAY (2, 72) "|"
82              DISPLAY (2, 74) "CODE".
83              DISPLAY (2, 80) "|".
84              DISPLAY (3, 1) HORIZ-LINE.
85
86        * REMAINDER OF FORM
87
88              DO FOR N FROM 4 BY 1 TO 24.
89
90              DISPLAY (N, 1)  "|".
91              DISPLAY (N, 31)  "|".
92              DISPLAY (N, 72)  "|".
93              DISPLAY (N, 80)  "|".
94
95              END-DO.
```

```
96
97        *  LOOP TO DISPLAY CONTENTS
98
99             DO FOR N FROM 4 BY 1 TO 13.
100
101            SUBTRACT 3 FROM N.
102
103            MOVE N TO M.
104
105            ADD 3 TO N.
106
107            DISPLAY (N, 10) NAM(M).
108            DISPLAY (N, 40) ADDR(M).
109            DISPLAY (N, 74) CODE(M).
110
111            END-DO.
112
113       * USE OF ACCEPT STMNT TO UPDATE FORM.
114
115        100.
116            MOVE 4 TO N.
117            MOVE 1 TO M.
118
119        101.
120            ACCEPT (N, 10) NAM(M)
121                           WITH UPDATE PROMPT
122                                DOWN 201
123                                RIGHT 102
124                                LEFT 103
125                                HOME 100
126                                UP 301
127                                EXIT 900.
128        102.
129            ACCEPT (N, 40) ADDR(M)
130                           WITH UPDATE PROMPT
131                                DOWN 202
132                                RIGHT 103
133                                LEFT 101
134                                HOME 100
135                                UP 302
136                                EXIT 900.
137
138        103.
139            ACCEPT (N, 74) CODE(M)
140                           WITH UPDATE PROMPT
141                                DOWN 203
142                                LEFT 102
143                                RIGHT 101
144                                HOME 100
145                                UP 303
146                                EXIT 900.
147
148        201.
149            PERFORM 500.
150            GO TO 101.
```

```
151
152         202.
153             PERFORM 500.
154             GO TO 102.
155
156         203.
157             PERFORM 500.
158             GO TO 103.
159
160         301.
161             PERFORM 600.
162             GO TO 101.
163
164         302.
165             PERFORM 600.
166             GO TO 102.
167
168         303.
169             PERFORM 600.
170             GO TO 103.
171
172         500.
173             ADD 1 TO N.
174             IF N IS GREATER THAN 13 THEN
175             SUBTRACT 1 FROM N
176             ELSE ADD 1 TO M
177             END-IF.
178
179         600.
180             SUBTRACT 1 FROM N.
181             IF N IS LESS THAN 4 THEN
182             ADD 1 TO N
183             ELSE SUBTRACT 1 FROM M
184             END-IF.
185
186
187         900.
188             STOP RUN.
189
```

*** NO ERROR MESSAGES ***

EXAMPLE 3.

```
ND-100 COBOL COMPILER  -  9 JUN 1982       TIME: 15.36.33   DATE: 82.07.13
    SOURCE FILE: COBE:COB
    OBJECT FILE: COBE

 1
 2          IDENTIFICATION DIVISION.
 3          PROGRAM-ID. SCREEN-PLAY.
 4
 5        * This program illustrates a few of the various ways of
 6        * visually displaying fields which the user may want to
 7        * update. Specific fields are accessed by use of control
 8        * keys. CR moves the cursor from field to field in the
 9        * order in which they are displayed. The screen is first
10        * filled with background characters.
11
12          ENVIRONMENT DIVISION.
13          CONFIGURATION SECTION.
14          SOURCE-COMPUTER. NORD-100.
15          OBJECT-COMPUTER. NORD-100.
16
17          DATA DIVISION.
18
19          WORKING-STORAGE SECTION.
20          77  LIN              PIC 99.
21          77  POS              PIC 99.
22          01  N                PIC XX VALUE "ND".
23          01  N1               PIC X(9) VALUE "NORWAY   ".
24          01  N2               PIC X(9) VALUE ".........".
25          01  N3               PIC 9(9) VALUE ZERO.
26          01  N4               PIC S9(3) COMP VALUE 0.
27
28          PROCEDURE DIVISION.
29          100.
30              BLANK SCREEN.
31           500.
32              PERFORM DISP
33                  VARYING LIN FROM 24 BY -1 UNTIL LIN < 1
34                  AFTER POS FROM 1 BY 2 UNTIL POS > 80.
35          DISP.
36              DISPLAY (LIN, POS) "ND".
37          1500.
38              ACCEPT (1, 1) N WITH UPDATE BEEP.
39          1700.
40               BLANK LINE 3 TO 9.
41          1800.
42              DISPLAY (5, 20) "COUNTRY :" WITH UNDERLINE.
43              ACCEPT (5, 30) N1 WITH UPDATE
44                                 UP 1500
45                                 EXIT 6000
46                                 HOME 5000
47                                 DOWN 3000.
```

```
48          2000.
49              BLANK LINE 12 COLUMN 8 TO 45.
50          3000.
51              DISPLAY (12, 10) "MONTH :" WITH INVERSE-VIDEO.
52              ACCEPT (12, 20) N2 WITH UPDATE
53                                  DOWN 3050
54                                  EXIT 1500
55                                  HOME 1800
56                                  UP 3000.
57          3050.
58              BLANK LINE 16 COLUMN 35 TO 55.
59
60          4000.
61              DISPLAY (16, 30) "SALES :" WITH BLINK.
62              ACCEPT (16, 40) N3 WITH UPDATE
63                                  UP 2000
64                                  HOME 3000
65                                  DOWN 4050
66                                  EXIT 6000.
67           4050.
68               BLANK LINE 21 COLUMN 20 TO 50.
69               BLANK LINE 22 COLUMN 20 TO 50.
70               BLANK LINE 23 COLUMN 20 TO 50.
71
72           5000.
73               DISPLAY (22, 24) "% CHANGE (+/-) :" WITH LOW-INTENSITY
74                                                       UNDERLINE.
75               ACCEPT (22, 42) N4 WITH UPDATE
76                                   UP 4000
77                                   DOWN 6000
78                                   HOME 3000
79                                   EXIT 2000.
80          6000.
81              STOP RUN.
82
```

*** NO ERROR MESSAGES ***

EXAMPLE 4.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. VIDEO.

*  This program is used to interrogate an existing file which
*  contains information on a video-film library. The choice is
*  of viewing either: a list of all films in the same category,
*  details regarding a film in any category, or the whole file
*  in alphabetic sequence. (The whole file can also be printed
*  out.) Only the more relevant parts of the program are shown.

 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. NORD-100.
 OBJECT-COMPUTER. NORD-100.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT VIDEO-FILE
     ASSIGN "VIDEO:DATA"
     ORGANIZATION INDEXED
     ACCESS DYNAMIC
     RECORD KEY CODE-NO
     ALTERNATE RECORD KEY CATEG WITH DUPLICATES
     ALTERNATE RECORD KEY TITLE WITH DUPLICATES
     STATUS V-STATUS.

     SELECT PRINT-FILE
         ASSIGN "L-P"
         STATUS V-STATUS.


 DATA DIVISION.
 FILE SECTION.
 FD VIDEO-FILE
    LABEL RECORDS STANDARD.
 01 VIDEO-REC.
    03  CODE-NO        PIC X(5).
    03  CATEG          PIC X(7).
    03  TITLE          PIC X(35).
    03  STARS          PIC X(20).
    03  RENT           PIC 9V99.
    03  IN-STOCK       PIC X.
    03  DATE-OUT       PIC X(6).
    03  DATE-BACK      PIC X(6).
    03  INCOME         PIC 999V99.
    03  FILLER         PIC X(30).


 WORKING-STORAGE SECTION.

 77  V-STATUS          PIC XX.
 77  REC-COUNT         PIC 9(4) VALUE ZERO.
 77  LINE-COUNT        PIC 99.
 77  OPTION            PIC X.
 77  REPLY             PIC X.
```

```
77  CATEGORY            PIC X(7).
77  NAME                PIC X(35).

PROCEDURE DIVISION.


BEGIN.
    OPEN I-O VIDEO-FILE WITH MULTI-USER MODE.
    OPEN OUTPUT PRINT-FILE.

*  Select an option

CHOOSE.
    BLANK SCREEN.
    DISPLAY (5, 20)"VIDEO LIBRARY INFORMATION PROGRAM"
           WITH UNDERLINE.
    DISPLAY (8, 24)"OPTIONS ARE :-"
                                          WITH AUTO-ERASE.
    DISPLAY (10, 22)"DISPLAY FILE BY CATEGORY -1"
                                          WITH AUTO-ERASE.
    DISPLAY (12, 22)"DISPLAY RECORD BY TITLE -2"
                                          WITH AUTO-ERASE.
    DISPLAY (14, 22)"DISPLAY FILE ALPHABETICALLY -3"
                                          WITH AUTO-ERASE.
    DISPLAY (16, 22)"PRINT FILE ALPHABETICALLY -4"
                                          WITH AUTO-ERASE.
    DISPLAY (18, 22)"EXIT FROM THE PROGRAM -5"
                                          WITH AUTO-ERASE.
    DISPLAY (20, 22) PLEASE ENTER YOUR OPTION"
                                          WITH AUTO-ERASE.
    ACCEPT (20, 49) OPTION WITH MUST.

    IF OPTION IS LESS THAN 1 OR GREATER THAN 5
                                  GO TO OPTION-ERROR.
    GO TO ONE, TWO, THREE, FOUR, FIVE, DEPENDING ON OPTION.


OPTION-ERROR.
    DISPLAY (20, 22)"INVALID OPTION, PRESS CR TO CONTINUE"
           WITH BEEP.
    ACCEPT (20, 61) REPLY.
    GO TO CHOOSE.

*  List all the films in a category

ONE.
    BLANK SCREEN.
    DISPLAY (4, 13)"DISPLAY THE CONTENTS OF ONE CATEGORY"
           WITH UNDERLINE.
    DISPLAY (6, 13)"ENTER REQUIRED CATEGORY :-".
    ACCEPT (6, 43) CATEGORY WITH PROMPT
                                  BLINK
                                  CONTROL VALID.

*  If an invalid category has been entered the above ACCEPT
*  will not have been "accepted". A known category must be
*  re-submitted.
```

```
        MOVE CATEGORY TO CATEG.
        START VIDEO-FILE KEY IS EQUAL TO CATEG
             INVALID KEY DISPLAY (10, 20)"ISAM FILE ERROR"
             WITH BEEP
             DISPLAY (10, 36) V-STATUS

*   CR to try again

             ACCEPT (10, 41) REPLY
             GO TO CHOOSE.
        PERFORM HEADER.
        GO TO ONE-NEXT.

    VALID.
        IF CATEGORY IS NOT EQUAL TO "HORROR" OR "WESTERN" OR
        "DRAMA" OR "ROMANCE" OR "SCI-FI" OR "CRIME"
        ACCEPT-ERROR.



    ONE-NEXT.
        READ VIDEO-FILE NEXT RECORD
            AT END ACCEPT (LINE-COUNT, 80) REPLY
            GO TO CHOOSE.

        IF CATEG IS NOT EQUAL TO CATEGORY
            ACCEPT (LINE-COUNT, 80) REPLY
            GO TO CHOOSE.

        DISPLAY (LINE-COUNT, 2) CODE-NO.
        DISPLAY (LINE-COUNT, 9) TITLE.
        DISPLAY (LINE-COUNT, 47) CATEG.
        DISPLAY (LINE-COUNT, 53) STARS.
        DISPLAY (LINE-COUNT, 76) RENT.
        ADD 1 TO LINE-COUNT.
        IF LINE-COUNT IS GREATER THAN 24
        ACCEPT (24, 80) REPLY
        PERFORM HEADER.
        GO TO ONE-NEXT.

    HEADER.
        BLANK SCREEN.
        DISPLAY (1, 28) "VIDEO LIBRARY CATALOGUE".
        MOVE 4 TO LINE-COUNT.
        DISPLAY (LINE-COUNT, 2) CODE-NO.
        DISPLAY (LINE-COUNT, 9) TITLE.
        DISPLAY (LINE-COUNT, 47) CATEG.
        DISPLAY (LINE-COUNT, 53) STARS.
        DISPLAY (LINE-COUNT, 76) RENT.



    TWO.
*   Display the details for one title

        BLANK SCREEN.
```

```
           DISPLAY (5, 20) "SEARCH FOR RECORD BY TITLE"
                  WITH UNDERLINE.
           DISPLAY (8, 10) "ENTER REQUIRED TITLE :-"
                  WITH INVERSE-VIDEO.
           ACCEPT (8, 34) NAME WITH PROMPT
                                  CONTROL ALPHA.

       *  The name of the video-film is checked for non-alphabetic
       *  characters. If any are found the above ACCEPT will not be
       *  taken and must be re-entered.

           MOVE NAME TO TITLE.
           START VIDEO-FILE KEY IS NOT LESS THAN TITLE
                INVALID KEY
                DISPLAY (10, 10) "TITLE NOT IN LIBRARY"
                       WITH BEEP.
                DISPLAY (11, 10) "PRESS CR"
                ACCEPT (11, 19) REPLY
                GO TO TWO.
           PERFORM HEADER.

         TWO-NEXT.
           READ VIDEO-FILE NEXT RECORD
               AT END DISPLAY (LINE-COUNT 35) "END OF FILE"
                     WITH BEEP
               ACCEPT (LINE-COUNT, 47) REPLY
               GO TO CHOOSE.

           DISPLAY (LINE-COUNT, 2) CODE-NO.
           DISPLAY (LINE-COUNT, 9) TITLE.
           DISPLAY (LINE-COUNT, 47) CATEG.
           DISPLAY (LINE-COUNT, 53) STARS.
           DISPLAY (LINE-COUNT, 76) RENT.
           ADD 1 TO LINE-COUNT.

           DISPLAY (LINE-COUNT, 35) "CORRECT RECORD Y/N?".
           ACCEPT (LINE-COUNT, 55) REPLY.
           IF REPLY IS EQUAL TO "Y" GO TO CHOOSE.
           BLANK LINE LINE-COUNT
           IF LINE-COUNT IS GREATER THAN 23
           PERFORM HEADER.
           GO TO TWO-NEXT.

         ALPHA.
           IF NAME IS NOT ALPHABETIC
           ACCEPT-ERROR.



         THREE.
       *  Display the file in alphabetic order of title.

           BLANK SCREEN.
           MOVE LOW-VALUES TO TITLE.
           START VIDEO-FILE KEY IS GREATER THAN TITLE
                INVALID KEY
                DISPLAY (2, 13) "ISAM FILE ERROR"
```

```
                WITH BEEP
            DISPLAY (2, 38) V-STATUS
            ACCEPT (2, 35) REPLY
            GO TO CHOOSE.
            PERFORM HEADER.

  THREE-NEXT.
      READ VIDEO-FILE NEXT RECORD
          AT END ACCEPT (LINE-COUNT, 80) REPLY
          GO TO CHOOSE.
      DISPLAY (LINE-COUNT, 2) CODE-NO.
      DISPLAY (LINE-COUNT, 9) TITLE.
      DISPLAY (LINE-COUNT, 47) CATEG.
      DISPLAY (LINE-COUNT, 53) STARS.
      DISPLAY (LINE-COUNT, 76) RENT.
      ADD 1 TO LINE-COUNT.
      IF LINE-COUNT IS GREATER THAN 24
      ACCEPT (24, 80) REPLY
      PERFORM HEADER.
      GO TO THREE-NEXT.

  FOUR.
*  Print the full catalogue alphabetically

      BLANK SCREEN.
      DISPLAY (5, 25) "PRINTING FULL ALPHABETIC CATALOGUE"
             WITH UNDERLINE.
      MOVE LOW-VALUES TO TITLE.
      MOVE ZERO TO REC-COUNT.
      START VIDEO-FILE KEY IS GREATER THAN TITLE
           INVALID KEY
           DISPLAY (7, 25) "ISAM FILE ERROR"
                  WITH BEEP
           DISPLAY (7, 42) V-STATUS
           ACCEPT (7, 50) REPLY
           GO TO CHOOSE.
               .
               .
*  Create the Print-Header
               .
               .
  FOUR-NEXT.
      READ VIDEO-FILE NEXT RECORD
          AT END
          DISPLAY (12, 22) " PROCESSING IS NOW COMPLETE"
                 WITH BEEP
          ACCEPT ( 12, 49) REPLY
          GO TO CHOOSE.
               .
               .
*      Create the print record
               .
               .

      IF LINE-COUNT IS GREATER THAN 60
      PERFORM PRINT-HEADER.
      GO TO FOUR-NEXT.
```

```
  FIVE.
*  Exit from the program

       BLANK SCREEN.
       CLOSE VIDEO-FILE.
       CLOSE PRINT-FILE.
       DISPLAY (12, 20) "}}} Returning to Main Menu {{{"
             WITH UNDERLINE.
       STOP RUN.
```

EXAMPLE 5.

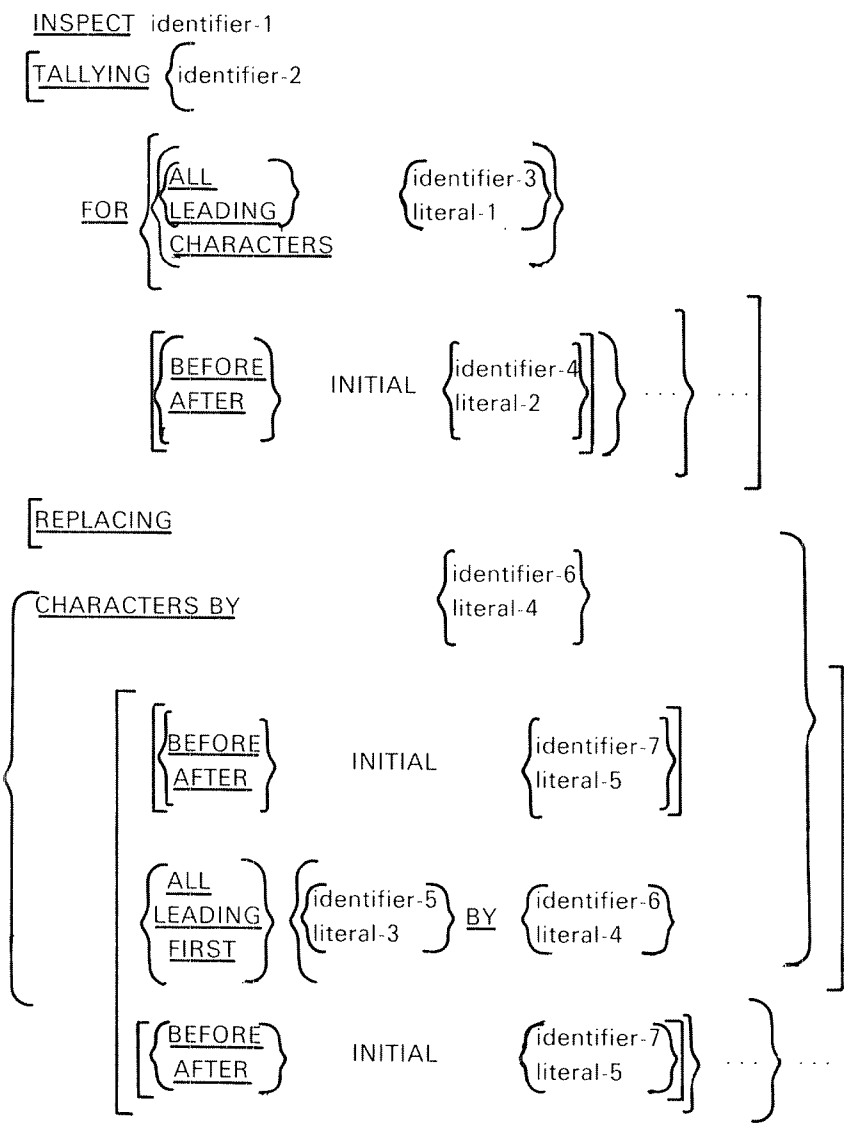
```
IDENTIFICATION DIVISION.
PROGRAM-ID. X-001.

*  This program demonstrates the facilities for framing selected portions of the
*  screen and for writing histogram bars onto it.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  NAME                        PIC X(30).
01  ANSWER                      PIC X.
01  I                           COMP.
01  J                           COMP.
01  K                           COMP.
01  X                           COMP.
01  Y                           COMP.

PROCEDURE DIVISION.

  500.  COMPUTE X = 19.
        COMPUTE Y = 9.

 1000.  BLANK SCREEN.
        DISPLAY (10,  1) "Your name:".
        ACCEPT (10, 12) NAME WITH PROMPT.
        BLANK LINE 10.
        DISPLAY ( 1,  1) FRAME 18 * 75 WITH HEADING.
        DISPLAY ( 2, 28) "M y  n a m e  i s".
        DO FOR I FROM 4 TO 17
           DISPLAY (I,  3) NAME WITH BLINK
           DISPLAY (I, 42) NAME WITH UNDERLINE
        END-DO.

 1500.  BLANK LINE 22.
        DISPLAY (22, 1) "Continue execution?" WITH UNDERLINE.
        ACCEPT (22, 20) ANSWER WITH PROMPT.
        IF ANSWER EQUAL "N" THEN PERFORM 3000.
        DISPLAY (Y, X) FRAME 12 * 34 WITH SPACE-FILL.
        DO FOR I FROM Y + 1 TO Y + 10
           DISPLAY (I, X + 2) NAME WITH INVERSE-VIDEO
        END-DO.
```

```
2000.   BLANK LINE 22.
        DISPLAY (22,  1) "Continue execution?" WITH UNDERLINE.
        ACCEPT (22, 20) ANSWER WITH PROMPT.
        IF ANSWER EQUAL "N" THEN PERFORM 3000.
        BLANK SCREEN.
        DISPLAY ( 1,  1) FRAME 20 * 73.
        DO FOR I FROM 2 BY 3 TO 71
           COMPUTE J = I
           DISPLAY (19, I) FULL-BAR J * 1
           COMPUTE J = 72 - I
           COMPUTE K = I + 1
           DISPLAY (19, K) SPARSE-BAR J * 1
        END-DO.

2500.   COMPUTE X = 5.
        COMPUTE Y = 3.
        PERFORM 1500.

3000.   BLANK LINE 22.
        DISPLAY (22, 11)
              "You have now used the ND COBOL Screen Handling"
              WITH UNDERLINE.
        STOP RUN.
```

6.6.3 The INSPECT Statement

The INSPECT statement specifies that characters in a data item are to be counted *or* replaced or counted *and* replaced.

Format:



Identifier-1, the inspected item must be either a group item or any category of elementary item with USAGE DISPLAY.

Identifier-2, the count field, must be an elementary integer data item.

All literals must be nonnumeric and any figurative constant except ALL. (If a figurative constant is used as literal-3 then the size of identifiers 6 and 7 must be one character in length.)

When the CHARACTERS phrase is used, literals 4 and 5 or identifiers 6 and 7 must be one character in length.

General Rules:

1. Either the TALLYING or REPLACING option must be given. Both may appear, but in this case all tallying occurs before any replacement is made.

2. All identifiers (except identifier-2) are treated by the INSPECT statement according to its category as:

   a. If alphabetic or alphanumeric - as a character-string.

   b. If alphanumeric edited, numeric edited or unsigned numeric - as though redefined as alphanumeric and the INSPECT statement refers to the alphanumeric item.

   c. If signed numeric - as though moved to an unsigned numeric data item of the same length and then treated as in rule b above.

3. Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE and AFTER phrase, and the mechanisms for tallying and/or replacing) begins at the leftmost character position of the data item identifier-1 and proceeds to the rightmost character position as described in the remaining general rules.

4. The rules for comparison are:

   a. The first TALLYING/REPLACING operand is compared with an equal number of the leftmost contiguous characters in the inspected item. A match occurs only if both are equal character-for-character.

   b. If no match occurs then the comparison is repeated for each successive TALLYING/REPLACING operand until either a match is found or all the operands have been acted upon.

   c. If a match is found then tallying/replacing takes place according to the following TALLYING/REPLACING option descriptions. The first character of the inspected item following the rightmost matching character is now the subject of the operations described in rules a and b above.

   d. If no match is found in this case then in the inspected item the first character following the leftmost inspected character now becomes the leftmost character position and processes of a and b above are repeated.

      The steps a and d, the comparison cycle, are repeated until the rightmost character has participated in a match or has been considered as the leftmost character position.

5. If the BEFORE/AFTER option is used then the previous rules are modified as described in the following TALLYING/REPLACING option descriptions.

TALLYING OPTION

Identifier-2 (an elementary integer item) is the *count field.* Identifier-3 or literal-1 is the *tallying field.*

If the BEFORE/AFTER option is not specified then the following actions occur on execution of INSPECT with TALLYING:

a. If ALL is used, the count field is increased by 1 for each non-overlapping occurrence of the tallying field.

b. If LEADING is specified, the count field is increased by 1 for each contiguous non-overlapping occurrence of this tallying field in the inspected item, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which this tallying field can take part in.

c. If CHARACTERS is specified, the count field is increased by 1 for each character position in the inspected item.

REPLACING OPTION

Identifier-5 or literal-3 is the *subject field,* identifier-6 or literal-4 is the *substitution field.* These two fields must be the same length and the following rules apply:

1. When the subject and substitution fields are character strings, each non-overlapping occurrence of the subject field in the inspected item is replaced by the character-string specified in the substitution field.

2. After replacement has occurred in any character position of the inspected item, no further replacement for that position is made during this INSPECT statement execution.

   When the BEFORE/AFTER option is not given then the following actions take place on execution of INSPECT with REPLACING:

   a. If CHARACTERS is specified, the substitution field must be one character in length. Each character in the inspected is replaced by the substitution field, beginning at the leftmost character and continuing to the rightmost.

   b. If ALL is specified, each non-overlapping occurrence of the subject field in the inspected item is replaced by the substitution field, beginning at the leftmost character and continuing to the rightmost.

   c. If LEADING is specified, each contiguous non-overlapping occurrence of the subject field of the inspected item is replaced by the substitution field provided that the leftmost occurrence is at the point where comparison began in the first comparison cycle in which this substitution field can participate.

   d. If FIRST is specified then the leftmost occurrence of the subject field in the inspected item is replaced by the substitution field.

BEFORE/AFTER OPTIONS

When these are specified the above rules for counting and replacing are modified thus:

Identifiers 4 and 7 and literals 2 and 5 are *delimiters* and are themselves not counted or replaced.

In the REPLACING option, if CHARACTERS is specified then the delimiter must be one character in length.

When BEFORE is used, counting and/or replacement of the inspected it begins at the leftmost character and continues until the first occurrences of the delimiter are encountered. If no delimiter occurs in the inspected item, counting and/or replacement continues to the rightmost character.

When AFTER is present, counting and/or 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 exists in the inspected item no counting/replacement takes place.

Following are six examples of the INSPECT statement:

(Note: identifier-2, the count field must be initialized before execution of the INSPECT statement.)

EXAMPLE 1.

INSPECT word TALLYING count FOR LEADING "L" BEFORE INITIAL "A", count-1 FOR LEADING "A" BEFORE INITIAL "L".

Where word = LARGE, count = 1, count-1 = 0.
Where word = ANALYST, count = 0, count-1 = 1.

EXAMPLE 2.

INSPECT word TALLYING count FOR ALL "L", REPLACING LEADING "A" BY "E" AFTER INITIAL "L".

Where word = CALLAR, count = 2, word = CALLAR.
Where word = SALAMI, count = 1, word = SALEMI.
Where word = LATTER, count = 1, word = LETTER.

EXAMPLE 3.

INSPECT word REPLACING ALL "A" BY "G" BEFORE INITIAL "X".

Where word = ARXAX, word = GRXAX.
Where word = HANDAX, word = HGNDGX.

EXAMPLE 4.

INSPECT word TALLYING count FOR CHARACTERS AFTER INITIAL "J" REPLACING ALL "A" BY "B".

Where word = ADJECTIVE, count = 6, word = BDJECTIVE.
Where word = JACK, count = 3, word = JBCK.
Where word = JUJMAB, count = 5, word = JUJMBB.

EXAMPLE 5.

INSPECT word REPLACING ALL "X" BY "Y", "B" BY "Z", "W" BY "Q" AFTER INITIAL "R".

Where word = RXXBQWY, word = RYYZQQY.
Where word = YZACDWBR, word = YZACDWZR.
Where word = RAWRXEB, word = RAQRYEZ.

EXAMPLE 6.

INSPECT word REPLACING CHARACTERS BY "B" BEFORE INITIAL "A".

word before: 1 2 X Z A B C D
word after: B B B B B A B C D

6.6.4 The MOVE Statement

The MOVE statement transfers data to one or more data areas in accordance with the editing rules.

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 represent the sending area; identifier-2, identifier-3, ..., represent the receiving area.

When format 2 is specified both identifiers must be group items. When CORRESPONDING is used, selected items in identifier-1 are moved to identifier-2 according to the rules given for the CORRESPONDING option in the earlier section on Arithmetic Statements. The results are the same as if each pair of corresponding identifiers had been referred to in a separate MOVE statement.

CORR is an abbreviation for CORRESPONDING.

General Rules:

1. The data in the sending area is moved into the first receiving area (identifier-2), then into the second receiving area (identifier-3) etc. Any subscripting or indexing associated with the sending item is evaluated immediately before the data is moved to the first receiving field. Similarly, any subscripting or indexing associated with receiving items is evaluated immediately before the data is moved in.

2. 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)

   Where temp has been defined as an intermediate result.

3. Any MOVE in which the sending and receiving items are both elementary items is an *elementary move*. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, alphanumeric edited. These categories are described in the PICTURE Clause. Numeric literals belong to the category numeric, and nonnumeric literals belong to the category alphanumeric. The figurative constant ZERO belongs to the category numeric. The figurative constant SPACE belongs to the category alphabetic. All other figurative constants belong to the category alphanumeric.

   The following rules apply to an elementary move between these categories:

   a. The figurative constant SPACE, an alphanumeric edited, or alphabetic data item must not be moved to a numeric or numeric edited data item.

   b. A numeric literal, the figurative constant ZERO, a numeric data item or a numeric edited data item must not be moved to an alphabetic data item.

   c. A non-integer numeric literal or a non-integer numeric data item must not be moved to an alphanumeric or alphanumeric edited data item.

   d. All other elementary moves are legal and are performed according to the rules given in general rule 4.

   e. A numeric edited item must not be moved to another numeric edited item.

   f. (An ND-Extension). A numeric edited item may be moved to a numeric item which is either integer or non-integer. This is equivalent to "de-editing".

4. Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, along with any editing specified for the receiving data item:

   a. When an alphanumeric edited or alphanumeric item is a receiving item, alignment and any necessary space filling takes place as defined under Standard Alignment Rules in the 'Working-Storage' Section of the Data Division. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled. If the sending item is described as being signed numeric, the operational sign will not be moved; if the operational sign occupied a separate character position (see the SIGN Clause), that character will not be moved and the size of the sending item will be considered to be one less than its actual size.

   b. When a numeric or numeric edited item is the receiving item, alignment by decimal point and any necessary zero-filling takes place as defined under the Standard Alignment Rules (except where zeroes are replaced because of editing requirements).

      1. When a signed numeric item is the receiving item, the sign of the sending item is placed in the receiving item. (See the SIGN Clause). Conversion of the respresentation of the sign takes place as necessary. If the sending item is unsigned, a positive sign is generated for the receiving item.

      2. When an unsigned numeric item is the receiving item, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.

      3. When a data item described as alphanumeric is the sending item, data is moved as if the sending item were described as an unsigned numeric integer.

   c. When a receiving field is described as alphabetic, justification and any necessary space-filling takes place as defined under the Standard Alignment Rules. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled.

Data in the following chart summarizes the legality of the various types of MOVE statements. The references are to the rule that prohibits the move or the behaviour of a legal move.

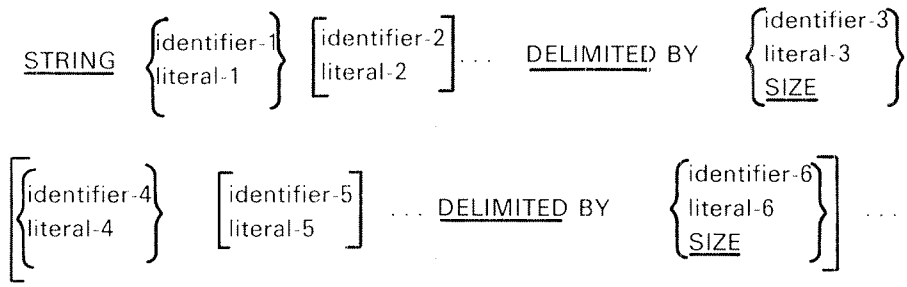
CATEGORY OF SENDING DATA ITEM		CATEGORY OF RECEIVING DATA ITEM			
		ALPHABETIC	ALPHANUMERIC EDITED ALPHANUMERIC	NUMERIC INTEGER NUMERIC NON-INTEGER	NUMERIC EDITED
ALPHABETIC		Yes/4c	Yes 4/a	No/3a	No/3a
ALPHANUMERIC		Yes/4c	Yes/4a	Yes/4b	Yes/4b
ALPHANUMERIC EDITED		Yes/4c	Yes/4a	No/3a	No/3a
NUMERIC	INTEGER	No/3b	Yes/4a	Yes/4b	Yes/4b
	NON-INTEGER	No/3b	No/3c	Yes/4b	Yes/4b
NUMERIC EDITED		No/3b	Yes/4a	Yes/3f	No/3e

5. Any move that is not an elementary move is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal respresentation to another. In such a move, the receiving area will be filled without consideration for the individual elementary or group items contained within either the sending or receiving area, except as noted in the general rules of the OCCURS Clause (See 'Table Handling' under Other Features).

6.6.5 The STRING Statement

The STRING statement enables the programmer to concatenate the complete or partial contents of two or more data items into a single data item.

Format:



INTO identifier-7 [WITH POINTER identifier-8]

[; ON OVERFLOW imperative-statement]

Each literal must be nonnumeric or any figurative constant except ALL.

The *sending fields* are given by identifier-7 which must represent an elementary alphanumeric data item.

The *pointer field* is identifier-8 which must represent an elementary numeric integer data item large enough to contain a value equal to the size plus one of the fields referenced by identifier-7. If no POINTER phrase exists, the default value of the logical pointer is one.

The *delimiters* are identifiers 3 and 6 or their corresponding literals.

When DELIMITED BY is specified, the contents of each sending field is transferred character-by-character until the rightmost character has been sent, or a delimiter for the sending field is reached.

All identifiers (except identifier-8) must have USAGE DISPLAY.

When the STRING statement is executed, the transfer of data is governed by the following rules:

Characters from the sending field are transferred to the sending field according to the rules for an alphanumeric to alphanumeric move, except that no space filling is provided.

If DELIMITED BY SIZE is specified each sending field is moved in its entirety to the receiving field.

If DELIMITED is specified without SIZE then the contents of each sending item is transferred character-by-character, starting with the leftmost and continuing until the end of the data or its delimiter is reached. (The delimiter itself is not transferred.)

If the POINTER option appears, the pointer field is explicitly available to the programmer. If this option does not appear it is as if the user had specified identifier-8 with an initial value of one.

When characters are transferred to the receiving field, the moves behave as though these characters were moved one at a time with the pointer field being incremented by one after each character is positioned. The value in the pointer cannot be changed in any other way. When processing is complete this value will be one character position greater than that of the last character transferred.

If this pointer value, at or after initiation of the STRING statement execution, becomes less than one or greater than the length of the receiving field, data transfer ceases. ON OVERFLOW, if specified, is now raised.

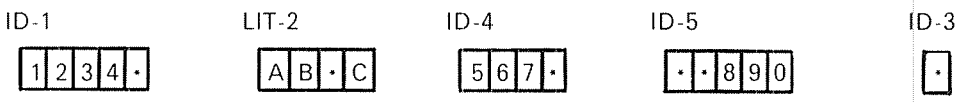
If ON OVERFLOW has not been specified, then, when the above conditions are encountered, control passes to the next executable statement.

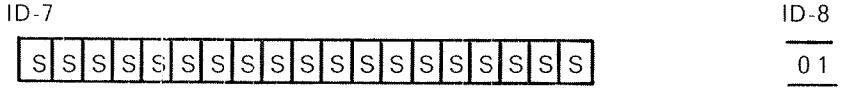
EXAMPLE.

If the following STRING statement is coded:

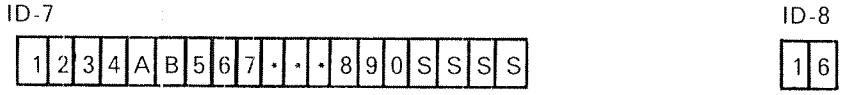
```
STRING ID-1  LIT-2  DELIMITED BY ID-3
  ID-4  ID-5  DELIMITED BY SIZE INTO
  ID-7  WITH POINTER  ID-8.
```

and at execution time the fields contain:





Then after execution the receiving field and the pointer field will appear as:



6.6.6 The UNSTRING Statement

The UNSTRING statement causes contiguous data in a single sending field to be separated and placed into multiple receiving fields.

Format:

```
UNSTRING identifier-1

[DELIMITED BY [ALL]  {identifier-2 | literal-1}  [, OR [ALL]  {identifier-3 | 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]
```

The DELIMITER IN option and the COUNT IN option may only appear if the DELIMITED BY option is also present.

Each literal must be a nonnumeric literal. Each may be any figurative constant without the word ALL.

Identifier-1 (the *sending* field) must be an alphanumeric data item.

Identifier-6 and identifier-9 must be type computational.

The DELIMITED BY option specifies the delimiters which control the amount of data (transferred from the sending field).

The *delimiters* are identifiers 2 and 3, or their corresponding literals, and each (which represents one delimiter) must be an alphanumeric data item. The maximum number of delimiters is 15.

If a delimiter contains two or more characters it will act as a delimiter only if the delimiter characters appear contiguously and in the sequence specified, in the sending field.

When two or more delimiters are specified in the DELIMITED BY option, an 'OR' condition exists between them. Each non-overlapping occurrence of any of the delimiters in the sending field in its specified sequence, is considered to be a match.

DELIMITED BY ALL results in one or more contiguous occurrences of any delimiter being treated as if they were only one occurrence; this occurrence is moved to the delimiter receiving field (if any), (identifiers 5, 8, . . .).

If DELIMITED BY ALL is not specified and two or more occurrences of any delimiter are found, then the current receiving field is either space or zero filled according to this field description.

When the UNSTRING statement is initiated, identifier-4 is the current *receiving* field. Receiving fields must have USAGE DISPLAY and must be one of the types:

alphabetic,
alphanumeric (not edited),
or numeric (not edited).

Data is transferred from the sending field according to the following rules:

If the POINTER option appears, it contains a value indicating a relative position in the sending field (it must be initialized prior to statement execution).

DELIMITED BY causes the examination to proceed from left to right, character-by-character until a delimiter is encountered. If no delimiter is found, the examination ends with the last character in the sending field.

If the DELIMITED BY option does not appear, the number of characters examined will be equal to the size of the current receiving field. (If the sign of the receiving field has been defined as occupying a separate character position, then the number of characters examined is one less than the size of this field.)

The characters thus examined (excluding delimiter(s) if any) are treated as an elementary alphanumeric data item and are transferred to the receiving field according to the rules of the MOVE statement.

The DELIMITED IN option causes the delimiting characters in the sending field to be treated as an elementary alphanumeric item and to be moved to the current delimiter receiving field (identifier-5) according to the rules of the MOVE statement. If, however, the delimiting condition is the end of the sending field, identifier-5 is filled with spaces.

If the COUNT IN option is specified, a value equal to the number of examined characters (excluding delimiter(s)) is moved to the data count field (identifier-6) according to rules for an elementary move (identifier-6 must be of type computational).

If the DELIMITED BY option appears then the sending field is further examined, beginning with the first character to the right of the delimiter. Otherwise, examination of the sending field begins with the first character to the right of the last character examined.

After data is transferred to the first receiving field (identifier-4), identifier-7 becomes the next receiving field. The preceding procedure is now repeated for this (and subsequently, for any succeeding receiving fields), until all characters in the sending field have been transferred or there are no more unfilled receiving fields.

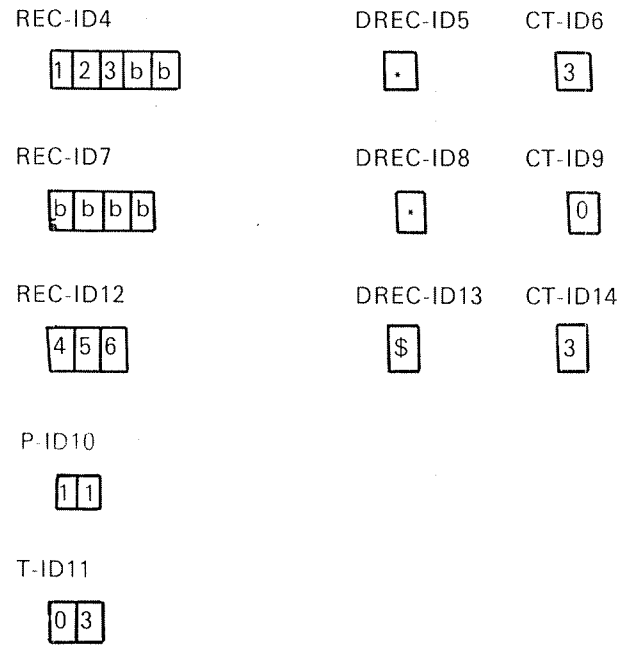
EXAMPLE:

The following UNSTRING statement:

```
UNSTRING  SEND-IDL DELIMITED BY ALL DEL-ID2 OR DEL-ID3
    INTO   REC-ID4  DELIMITER IN DREC-ID5 COUNT IN CT-ID6
           REC-ID7  DELIMITER IN DREC-ID8 COUNT IN CT-ID9
           REC-ID12 DELIMITER IN DREC-ID13 COUNT IN CT-ID14
    WITH POINTER P-ID10
    TALLYING IN  T-ID11
    ON OVERFLOW GO TO UNSTRING-OFL.
```

might have the following field contents at execution time

and the remaining fields, after execution, with contents:



Where b represents a space (blank character). Since SEND-ID1 still contains untransferred characters, the ON OVERFLOW condition will be raised.

If a further receiving field had been specified for this UNSTRING statement then the first character moved to it from the sending field would have been the leftmost character following the second $, i.e., the number 7. (Note the difference in effect of coding DELIMITED BY with or without ALL.)

When the execution of the UNSTRING statement has been completed, if a TALLYING IN option is present, then the field-count field (identifier-11) will have had its initial value incremented by the number of data receiving areas acted upon (including any null fields).

At this point, if a POINTER option has been specified, the pointer field (identifier-10) will contain a value equal to its initial value plus the number of characters examined in the sending field.

Execution of the UNSTRING statement will cease if an overflow condition exists. If ON OVERFLOW is specified the imperative-statement is executed. If ON OVERFLOW is not specified control passes to the next executable statement. An overflow condition occurs if:

a. The value in the pointer field is less than one or greater than the length of the sending field when UNSTRING is initiated.

b. During execution of the UNSTRING statement, after all receiving fields have been acted upon, the sending field still contains unexamined characters.

Any subscripting or indexing associated with the identifiers is evaluated immediately before data transfer.

6.7 INPUT-OUTPUT STATEMENTS

COBOL input-output statements transfer data to and from files stored on external devices and they control low-volume data going to or coming from media such as console typewriters and terminals.

The unit of data used by the COBOL program is termed a *record*.

The input-output statements which may be used in the Procedure Division are determined by the file descriptions in the Environment and Data Divisions.

6.7.1 I-O Status

If the FILE STATUS clause is specified in a file-control entry, a value is placed into the specified 2-character data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE or DELETE statement and before any applicable USE procedure is executed, to indicate the status of the I-O operation.

See Appendix I, Indexed/Relative I-O Status Summary.

6.7.1.1 Status Key 1

The leftmost character position of the FILE STATUS data item is set upon completion of an I-O operation to the following conditions.

'0' indicates Successful Completion

'1' indicates At End

'2' indicates Invalid Key

'3' indicates Permanent Error

'9' indicates Other Error

The meanings of the indications are:

0 — Successful Completion. The I-O statement was successfully executed.

1 — At End. *Indexed and Relative I-O.*

The Format 1 READ statement was unsuccessfully executed as the result of an attempt to read a record when no next logical record exists in the file.

At End. *Sequential I-O.*

The sequential READ statement was unsuccessfully executed either as a result of an attempt to read a record when no next logical record exists in the file or as a result of the first READ statement being executed for a file described with the OPTIONAL clause, and that file was not available to the program at the time its associated OPEN statement was executed.

2 — Invalid Key. The I-O statement was unsuccessfully executed as one of the following.

Sequence Error (Indexed I-O only)

Duplicate Key

No Record Found

Boundary Violation

Two programs attempting to access the same record (only with Indexed or Relative I-O, with MULTI-USER Access)

Invalid Key does not apply to Sequential I-O.

3 — Permanent Error. The input-output statement was unsuccessfully executed as the result of a boundary violation for a sequential file or as the result of an input-output error, such as data check parity error, or transmission error.

9 — Some other error.

6.7.1.2 Status Key 2

The rightmost character position of the FILE STATUS data item is known as status key 2 and is used to further describe the results of the input-output operation.

The value this character contains will have the meanings given in the table below, according to the appropriate file organization.

Status Key 2:	*Meaning:*
0	No further information
1	If Status Key 1 is '2' — Sequence error Otherwise — Password failure
2	If Status Key 1 is '0' — Duplicate key (Indexed files) If Status Key 1 is '2' — Duplicate key (Relative and Indexed files) Otherwise — Logic error
3	If Status Key 1 is '2' — No record found (Relative and Indexed files) Otherwise — Resource not available
4	If Status Key 1 is '2' — Boundary violation (Relative and Indexed files) If Status Key 1 is '3' — Boundary violation (Sequential files) Otherwise — No current record pointer
5	Invalid or incomplete file information
6	No file information given
7	Open successful

VALID COMBINATIONS OF STATUS KEYS 1 AND 2

The valid permissible combinations of the values of status key 1 and status key 2 are shown in the following figures. An 'X' at an intersection indicates valid permissible combination.

Status Key 1	Status Key 2				
	No Further Information (0)	Sequence Error (1)	Duplicate Key (2)	No Record Found (3)	Boundary Violation (4)
Successful Completion (0)	X		X		
At End (1)	X				
Invalid Key (2)		X	X	X	X
Permanent Error (3)	X				
Other Error (9)					

INDEXED I-O

Status Key 1	Status Key 2			
	No Further Information (0)	Duplicate Key (2)	No Record Found (3)	Boundary Violation (4)
Successful Completion (0)	X			
At End (1)	X			
Invalid Key (2)		X	X	X
Permanent Error (3)	X			
Other Error (9)				

RELATIVE I-O

Status Key 1	Status Key 2	
	No Further Information (0)	Boundary Violation (4)
Successful Completion (0)	X	
At End (1)	X	
Permanent Error (3)	X	X
Other Error (9)		

SEQUENTIAL I-O

6.7.1.3 The INVALID KEY Condition (Indexed and Relative I-O Only)

The INVALID KEY condition can occur as a result of the execution of a START, READ, WRITE, REWRITE or DELETE statement. For details of the causes of the condition see under the relevant statement headings.

When the INVALID KEY condition is recognized, the runtime-system takes these actions in the following order:

1. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an INVALID KEY condition. (See under I-O status earlier in this section.)

2. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative-statement. Any USE procedure specified for this file is not executed.

3. If the INVALID KEY phrase is not specified, but a USE procedure is specified, either explicitly or implicitly, for this file, that procedure is executed.

6.7.1.4 The AT END Condition

The AT END condition can occur as the result of a READ statement. For details see under the statement heading.

6.7.1.5 Current Record Pointer

The current record pointer is a conceptual entity for identifying the next record to be accessed within a given file. (It has no meaning for a file opened in output mode.)

The OPEN statement positions it at the first record in the file.

For a READ statement note the following:

1. If the OPEN statement positioned the current record pointer, the record identified by it is made available.

2. If a previous READ statement positioned the current record pointer then this is updated to point to the next existing record which is then made available.

3. (Indexed and Relative I-O only.) The START statement positions the current record pointer at the first record in the file that satisfies the comparision specified.

6.7.1.6 The CLOSE Statement

The CLOSE statement terminates the processing of files (with optional rewind for Sequential I-O).

Format 1 - Indexed and Relative I-O.

```
CLOSE file-name-1 [WITH LOCK] [, file-name-2 [WITH LOCK] ] . . .
```

Format 2 - Sequential I-O.

```
CLOSE file-name-3 [ [REEL]          [WITH NO REWIND] ]
                  [ [UNIT]                           ]
                  [ WITH  {NO REWIND}                ]
                  [       {LOCK     }                ]

[ , file-name-4 . . . [ [REEL]          [WITH NO REWIND] ] ] . . .
                      [ [UNIT]                           ]
                      [ WITH  {NO REWIND}                ]
                      [       {LOCK     }                ]
```

The files referenced in the CLOSE statement need not all have the same organization or access.

General Rules:

1. A CLOSE statement may only be executed for a file in an open mode.

2. The action taken if a file is in the open mode when a STOP RUN statement is executed is that the file will be closed. Note, however, that the last block in memory will not be written out so that the last record on the file may be lost.

General Rules For Indexed And Relative I-O:

1. If a CLOSE statement has been executed for a file, no other statement can be executed that references that file, either explicitly or implicitly, unless an intervening OPEN statement for that file is executed.

2. Following the successful execution of a CLOSE statement, the record area associated with file-name is no longer available. The unsuccessful execution of such a CLOSE statement leaves the availability of the record area undefined.

General Rule For Sequential I-0:

1. Treatment of mass storage files is logically equivalent to the treatment of a file on a tape.

6.7.1.7 The DELETE Statement

The DELETE statement logically removes a record from a mass storage file. It is not used with Sequential I-O.

Format:

DELETE file-name RECORD [; INVALID KEY imperative-statement]

The INVALID KEY phrase must not be specified for a DELETE statement which references a file which is in sequential access mode.

The INVALID KEY phrase must be specified for a DELETE statement which references a file which is not in sequential access mode and for which an applicable USE procedure is not specified.

General Rules:

1. The associated file must be open in the I-O mode at the time of the execution of this statement.

2. For files in the sequential access mode, the last input-output statement executed for file-name prior to the execution of the DELETE statement must have been a successfully executed READ statement. The run-time system logically removes from the file the record that was accessed by that READ statement.

3. For a file in random or dynamic access mode, the run-time system logically removes from the file that record identified by the contents of the RELATIVE KEY data item associated with file-name. If the file does not contain the record specified by the key, an INVALID KEY condition exists.

4. After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.

5. The execution of a DELETE statement does not affect the contents of the record area associated with file-name.

6. The current record pointer is not affected by the execution of a DELETE statement.

7. The execution of the DELETE statement causes the value of the specified FILE STATUS data item, if any, associated with file-name to be updated.

6.7.1.8 The OPEN Statement

The OPEN statement initiates the processing of files. For Indexed and Relative I-O it also performs checking and/or writing of labels and other input-output operations.

Format 1. Sequential Files.

```
OPEN { INPUT file-name-1 [WITH NO REWIND][,file-name-2 [WITH NO REWIND]] ...
     { OUTPUT file-name-3 [WITH NO REWIND] [,file-name-4 [WITH NO REWIND]]...
     { I-O      file-name-5 [,file-name-6 ] ...
     { EXTEND file-name-7 [,file-name-8] ...
```

Format 2. Indexed and Relative Files.

```
OPEN { INPUT  }
     { OUTPUT } file-name [WITH { MULTI-USER-MODE   }]
     { I-O    }                { IMMEDIATE-WRITE  }
                               { MANUAL-UNLOCK    }
   [ { INPUT  }
   [ { OUTPUT } file-name [WITH { MULTI-USER-MODE   }]
   [ { I-O    }                { IMMEDIATE-WRITE  }  ...
                               { MANUAL-UNLOCK    }
```

The files referenced in the OPEN statement need not all have the same organization or access. The I-O option can only be used for mass storage files. The EXTEND option is only valid for Sequential files.

The successful execution of an OPEN statement determines the availability of the file and results in that file being in an open mode. It also makes the associated record area available to the programs.

Prior to the successful execution of an OPEN statement for a file, no statement (except in Sequential I-O, for a SORT or MERGE statement with either GIVING or USING phrases) can be executed that references that file.

An OPEN statement must be successfully executed prior to the execution of any of the permissible input-output statements. The following Tables show permissible statements for each I-O classification.

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

PERMISSIBLE STATEMENTS FOR INDEXED AND RELATIVE I-O—OPEN

For Indexed I-O, an 'X' at an intersection indicates that the specified statement, used in the access method given for that row, may be used with the indexed file organization and the open mode given at the top of the column.

For Relative I-O, an 'X' at an intersection indicates that the specified statement, used in the access method given for that row, may be used with the relative file organization and the open mode given at the top of the column.

Statement	*Open Mode*			
	Input	*Output*	*Input-Output*	*Extend*
READ	X		X	
WRITE		X		X
REWRITE			X	

PERMISSIBLE STATEMENTS FOR SEQUENTIAL I-O — OPEN

An 'X' at an intersection indicates that the specified statement, used in sequential access mode, may be used with the sequential file organization and open mode given at the top of the column.

General Rules For Indexed And Relative I-O:

1. A file may be opened with the INPUT, OUTPUT and I-O options in the same program. After the initial execution of an OPEN statement for a file, each subsequent OPEN statement execution for this file must be preceded by the execution of a CLOSE statement (without the LOCK phrase for Indexed I-O) for the same file.

2. Execution of the OPEN statement does not obtain or release the first data record.

3. The file description entry for files open for INPUT or I-O must be equivalent to that used when this file was created.

4. For files being opened with the INPUT or I-O option, the OPEN statement sets the current record pointer to the first record currently existing within the file. For indexed files, the prime record key is established as the key of reference and is used to determine the first record to be accessed. If no records exist in the file, the current record pointer is set such that the next executed format 1 READ statement for the file will result in an AT END condition.

5. The I-O option permits the opening of a file for both input and output operations. Since this option implies the existence of the file, it cannot be used if the file is being initially created.

6. Upon successful execution of an OPEN statement with the OUTPUT option specified, a file is created. At that time the associated file contains no data records.

7. The options MULTI-USER-MODE, IMMEDIATE WRITE, and MANUAL-UNLOCK are ND Extensions. They are used with relative or indexed organized files and they have the following meanings:

   MULTI-USER MODE allows one program to be running concurrently on several terminals, each accessing the same relative or indexed organized file, and it also allows different programs running concurrently on several terminals to access the same relative or indexed organized file. In both of these cases, if the programs access the same record in the ISAM file, conflicts are prevented.

   IMMEDIATE-WRITE (single user only) causes records to be written back immediately to the file, a process which happens automatically in MULTI-USER-MODE; otherwise output is buffered. This option would be useful, for instance, where high security is required.

   MANUAL-UNLOCK will prevent the automatic unlock of records until an UNLOCK statement is encountered. However, the user is strongly advised to allow automatic unlock of records since the use of MANUAL-UNLOCK can lead to deadlocks.

   Note that the multi-user supervisor must be active before running programs in multi-user mode. The system supervisor for the users installation should do this.

General Rules For Sequential I-O:

1. A file may be opened with the INPUT, OUTPUT, EXTEND and I-O options in the same program. Following the initial execution of an OPEN statement, each subsequent OPEN statement for the same file must be preceded by the execution of a CLOSE statement for it.

2. Execution of the OPEN statement does not obtain or release the first data record.

   The file description entry for file-names 1, 2, 5, 6, 7, or 8 must be the equivalent to that used when the file was created.

3. If an input file is designated with the OPTIONAL phrase in its SELECT clause, the object program causes an interrogation for the presence or absence of this file. If the file is not present, the first READ statement for this file causes the AT END condition to occur. (See the READ statement later in this section.)

4. For files being opened with the INPUT or I-O option, the OPEN statement sets the current record pointer to the first record currently existing within the file. If no records exist in the file, the current record pointer is set such that the next executed READ statement for the file will result in an AT END condition.

5. The EXTEND option permits opening of the file for output operations. (The OPEN statement positions the file immediately following the last logical record of that file. Subsequent WRITE statements referencing the file will add records to the file as though the file had been opened with the OUTPUT option.)

6. The I-O option allows the opening of a mass storage file for both input and output operations. Since this option implies the existence of the file, it cannot be used if the mass storage file is being initially created.

7. Upon successful execution of an OPEN statement with the OUTPUT option specified, a file is created. At that time the associated file contains no data records.

8. If the OPTIONAL phrase has been given for the file in the FILE-CONTROL paragraph of the Environment Division and the file is not present, then the standard end-of-file processing is performed for that file if it is an input file. If it is an output file it is created.

6.7.1.9 The READ Statement

The READ statement makes available the next logical record from a file. The formats are:

Format 1:

READ file-name [NEXT] RECORD[INTO identifier] [WITH LOCK]
[; AT END imperative-statement]

Format 2. Indexed I-O Only:

READ file-name RECORD [INTO identifier] [WITH LOCK] [; KEY IS data-name]
[; INVALID KEY imperative-statement]

Format 3. Relative I-O Only:

READ file-name RECORD [INTO identifier] [WITH LOCK]
[;INVALID KEY imperative-statement]

The storage areas associated with file-name and with identifier must not be the same.

Format 1.

The NEXT phrase and the WITH LOCK phrase are not valid for sequential files. This format is used for sequential retrieval of records when files of other organizations are in dynamic access mode. The WITH LOCK phrase applies only to files opened in MULTI-USER-MODE, and is an ND Extension (see the OPEN statement).

Format 1 must be used for all files in sequential access mode.

If the AT END phrase appears and if no applicable USE procedure is given for file-name, a run-time error will result.

Format 2. Indexed I-O Only:

Data-name, which may be qualified, must identify a record key associated with file-name.

Formats 2 and 3:

These formats are used for files in dynamic or random access modes when records are to be retrieved randomly.

If the INVALID KEY phrase appears and if no applicable USE procedure is given for file-name, a run-time error will result.

The WITH LOCK phrase is an ND Extension and applies only to files opened in MULTI-USER-MODE (see the OPEN statement and General Rule 3 for Indexed and Relative I-O later in this section).

General Rules:

1. The associated file must be open in the INPUT or I-O mode when this statement is executed. (See the OPEN statement earlier.)

2. The execution of the READ statement causes the value of the FILE STATUS data item, if any, to be updated. (See I-O Status at the beginning of the I-O section.)

3. If the INTO phrase is specified, the record being read is moved from the record area to the area specified by identifier according to the rules specified for the MOVE statement. The implied MOVE does not occur if the execution of the READ statement was unsuccessful. Any indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

4. When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.

5. If, at the time of execution of a format 1 READ statement, the position of current record pointer for that file is undefined, the execution of that READ statement is unsuccessful.

6. When the AT END condition is recognized, the following actions are taken in the specified order:

   a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition.

   b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative-statement. Any USE procedure specified for this file is not executed.

   c. If the AT END phrase is not specified, then a USE procedure must be specified, either explicitly or implicitly, for this file, and that procedure is executed.

      When the AT END condition occurs, execution of the input-output statement which caused the condition is unsuccessful.

7. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined. For indexed files the key of reference is also undefined.

8. For a file for which dynamic access mode is specified, a format 1 READ statement with the NEXT phrase specified causes the next logical record to be retrieved from that file as described in rule 1.

General Rules For Indexed I-O:

1. The record to be made available by a format 1 READ statement is determined as follows:

   a. The record, pointed to by the current record pointer, is made available provided that the current record pointer was positioned by the START or OPEN statement and the record is still accessible through the path indicated by the current record pointer; if the record is no longer accessible (which may have been caused by deletion of the record or by a change in an alternate record key) the current record pointer is updated to point to the next existing record within the established key of reference and that record is then made available.

   b. If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file with the established key of reference. That record is then made available.

2. For an indexed file being sequentially accessed, records having the same duplicate value in an alternate record key which is the key of reference are made available in the same order in which they are released by execution of WRITE statements, or by execution of REWRITE statements which create such duplicate values.

3. For an indexed file if the KEY phrase is specified in a format 2 READ statement, data-name is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of format 1 READ statements for the file until a different key of reference is established for the file.

4. If the KEY phrase is not specified in a format 2 READ statement, the prime record key is established as the key of reference for this retrieval. If the dynamic access mode is specified, this key of reference is also used for retrievals by any subsequent executions of format 1 READ statements for the file until a different key of reference is established for the file.

5. If execution of a format 2 READ statement causes the value of the key of reference to be compared with the value contained in the corresponding data item of the stored records in the file, 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 so identified, the INVALID KEY condition exists and execution of the READ statement is unsuccessful.

General Rules For Indexed And Relative I-O:

1. If, at the time of the execution of a format 1 READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful.

2. When the AT END condition has been recognized, a format 1 READ statement for that file must not be executed without first executing one of the following:

   a. A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

   b. A successful START statement for that file.

   c. A successful format 2 (for Indexed I-O) READ statement for that file (or format 3 for Relative I-O).

3. The WITH LOCK phrase exists for the access of ISAM files which have been opened in MULTI-USER-MODE (see the OPEN statement). If the file has been opened in single-user mode, the phrase is treated as comments only.

   If coded, READ WITH LOCK ensures that two programs cannot modify the same record at the same time. A locked record can only be read. It will become unlocked automatically when it has been rewritten by the program which locked it, or when another record has been read, or when the file is closed. The record can also be unlocked upon execution of the UNLOCK statement (see Section 6.7.1.12).

   An attempt by two programs to modify the same record will raise the INVALID KEY condition with a file status code of 68 or 78 depending on whether the records are "locked" or not.

   It is the responsibility of the user program to provide the code which enables it to wait for a record to become accessible; a read loop might otherwise occur.

   Note also the requirements for relative or indexed organized files accessed in multi-user mode. All relative or indexed organized files, both the index and data part, must be contiguous SINTRAN files. The size of the index part (in SINTRAN pages) is found by using the ESTIMATE-INDEX-FILE-SIZE function in the INDEXED SEQUENTIAL ACCESS METHOD SERVICE program. The size (in SINTRAN pages) of the data part is:

   (maximum number of records * record length/2048) + 1

General Rules For Relative I-O Only:

1. The record to be made available by a format 1 READ statement is determined as follows:

   a. The record, pointed to by the current record pointer, is made available provided that the current record pointer was positioned by the START or OPEN statement and the record is still accessible through the path indicated by the current record pointer; if the record is no longer accessible, which may have been caused by the deletion of the records, the current record pointer is updated to point to the next existing record in the file and then that record is made available.

   b. If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file with the established key of reference and then that record is made available.

2. If the RELATIVE KEY phrase is specified, the execution of a format 1 READ statement updates the contents of the RELATIVE KEY data item such that it contains the relative record number of the record made available.

3. The execution of a format 2 READ statement sets the current record pointer to, and makes available, the record whose relative record number is contained in the data item named in the RELATIVE KEY phrase for the file. If the file does not contain such a record, the INVALID KEY condition exists and execution of the READ statement is unsuccessful.

General Rules For Sequential I-O:

1. The record to be made available by a format 1 READ statement is determined as follows:

   a. If the current record pointer was positioned by the execution of the OPEN statement, the record pointed to by the current record pointer is made available.

   b. If the current record pointer was positioned by the execution of a previous READ statement, the current record pointer is updated to point to the next existing record in the file and then that record is made available.

2. When the AT END condition has been recognized, a READ statement for that file must not be executed without first executing a successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

3. If at the time of the execution of a READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the READ statement is considered unsuccessful.

4. If a file described with the OPTIONAL phrase is not present at the time the file is opened, then at the time of execution of the first READ statement for the file, the AT END condition occurs and the execution of the READ statement is unsuccessful. The standard end-of-file procedures are not performed. Execution then proceeds as specified in general rule 6.

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 15.57.20   DATE: 21.10.80
    SOURCE FILE: ISAM-EX1

      1        IDENTIFICATION DIVISION.
      2        PROGRAM-ID.
      3              GEN-ISAM-1.
      4       ************************************************************************
      5       *      ISAM (INDEX SEQUENTIAL ACCESS METHOD ).
      6       *THE RECORD IS THE OUTPUT TO AN ISAM FILE USING THE *UNIQUE*
      7       *(IE: NO DUPLICATES ) DATA FOUND IN FIELD ISAM-KEY AS *KEY* VALUE.
      8       *
      9       *  BEFORE THIS JOB CAN BE RUN THE FOLLOWING *MUST* BE SO :
     10       *      A) FILE "ISAM-EX:DATA" MUST EXIST.
     11       *      B) FILE "ISAM-EX:ISAM" MUST NOT EXIST OR IF EXISTING
     12       *                                CONTAIN *NO DATA !!*
     13       ************************************************************************
     14        ENVIRONMENT DIVISION.
     15        INPUT-OUTPUT SECTION.
     16        FILE-CONTROL.
     17              SELECT ISAM-FILE ASSIGN TO "ISAM-EX:DATA",
     18                               ORGANIZATION IS INDEXED,
     19                               ACCESS MODE IS DYNAMIC,
     20                               RECORD KEY IS  ISAM-KEY,
     21                               FILE STATUS IS ISAMSTATUS.
     22        DATA  DIVISION.
     23        FILE SECTION.
     24
     25        FD ISAM-FILE
     26              RECORD CONTAINS 46 CHARACTERS,
     27              DATA RECORD IS ISAM-REC.
     28        01 ISAM-REC.
     29              02 ISAM-KEY     PIC X(6).
     30       *          :.....................MUST BE IN RECORD AREA !
     31              02 ISAM-TEXT     PIC X(40).
     32
     33        WORKING-STORAGE SECTION.
     34        01 ISAMSTATUS         PIC XX.
     35       *                                RETURN STATUS FROM ISAM.
     36       ************************************************************************
     37        PROCEDURE DIVISION.
     38        A001.
     39              OPEN I-O ISAM-FILE.
     40        A002.
     41              DISPLAY "ENTER KEY  (MAX 6 CHAR ) :",
     42              ACCEPT ISAM-KEY.
     43              IF ISAM-KEY = SPACES GO TO LIST.
     44       *                                SPACES INPUT , END DIALOG
     45              DISPLAY "ENTER TEXT (MAX 40 CHAR) :".
     46              ACCEPT ISAM-TEXT.
     47       *                                READ RECORDS FROM TERMINAL
     48              WRITE ISAM-REC , INVALID KEY,
     49                  DISPLAY "ISAM FILE ERROR :", ISAMSTATUS, ":".
     50              GO TO A002.
     51       *                                OUTPUT RECORD AND ASK AGAIN
     52        LIST.
     53              DISPLAY "ENTER ACCESS  KEY :".
     54              ACCEPT ISAM-KEY.
     55              IF ISAM-KEY  = SPACES  GO TO FINI.
     56              READ ISAM-FILE RECORD KEY IS ISAM-KEY INVALID KEY,
     57                               DISPLAY "** RECORD NOT FOUND !",
     58                               GO TO LIST.
     59              DISPLAY "REC: ", ISAM-KEY, ": ", ISAM-REC.
     60              GO TO LIST.
     61        FINI.
     62              CLOSE ISAM-FILE.
     63              DISPLAY "JOB END".
     64              STOP RUN.

*** NO ERROR MESSAGES ***
```

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 09.06.39   DATE: 22.10.80
    SOURCE FILE: REL-EX

      1        IDENTIFICATION DIVISION.
      2        PROGRAM-ID.
      3              GENRELATIVE.
      4       ****************************************************************
      5       *     SHOWS THE USAGE OF A RELATIVE FILE :
      6       * THE FILE *MUST* EXIST BEFORE THE RUN BUT MAY BE EMPTY, EACH
      7       * RECORD IS LOCATED DIRECTLY BY ITS RELATIVE (TO 1) POSITION IN
      8       * THE FILE BY ITS *NUMERIC* KEY VALUE.
      9       ****************************************************************
     10        ENVIRONMENT DIVISION.
     11        INPUT-OUTPUT SECTION.
     12        FILE-CONTROL.
     13               SELECT RELFILE ASSIGN "RELATIVE-EX:DATA" ,
     14                        ORGANIZATION IS RELATIVE,
     15                        ACCESS IS DYNAMIC,
     16                        RELATIVE KEY IS REL-KEY,
     17                        FILE STATUS IS REL-STATUS.
     18        DATA DIVISION.
     19        FILE SECTION.
     20
     21        FD  RELFILE
     22                  LABEL RECORD IS OMITTED
     23                  DATA RECORD IS  REL-RECORD
     24                  BLOCK CONTAINS  10 RECORDS
     25                  RECORD CONTAINS 60 CHARACTERS.
     26        01  REL-RECORD       PIC X(60).
     27       *                       RECORD CANNOT BE "QED" TYPE RECORD
     28
     29        WORKING-STORAGE SECTION.
     30        01 REL-STATUS         PIC XX.
     31        01 REL-KEY            PIC 999.
     32       *           :.........CANNOT APPEAR IN RELFILE RECORD AREA,
     33       *                       MAX POSSIBLE SIZE IS 999999, RESTRICTED
     34       *                       TO 999 IN THIS PROGRAM.
     35
     36        PROCEDURE DIVISION.
     37
     38        A000.
     39              OPEN I-O RELFILE.
     40        A002.
     41              DISPLAY "ENTER KEY (MAX 999 ) :".
     42              PERFORM GET-KEY.
     43              IF REL-KEY = ZEROES GO TO A003.
     44              DISPLAY "ENTER TEXT ( MAX 60 CHAR ) :".
     45              ACCEPT REL-RECORD.
     46              WRITE REL-RECORD INVALID KEY,
     47                              DISPLAY " ** RELFILE ERROR :", REL-STATUS.
     48              GO TO A002.
     49        A003.
     50              DISPLAY "ENTER ACCESS KEY :".
     51              PERFORM GET-KEY.
     52              IF REL-KEY = ZEROS GO TO A999.
     53
     54              READ RELFILE RECORD INVALID KEY,
     55                           DISPLAY " ** RECORD NOT FOUND !", REL-STATUS,
     56                           GO TO A003.
     57              DISPLAY "REC :", REL-KEY, ":", REL-RECORD.
     58              GO TO A003.
     59        A999.
     60              CLOSE RELFILE.
     61              DISPLAY "JOB END".
     62              STOP RUN.
     63        GET-KEY.
     64              ACCEPT REL-KEY.
     65              IF REL-KEY NOT NUMERIC ,
     66                              DISPLAY " ** KEY MUST BE NUMERIC ",
     67                              GO TO GET-KEY.
     68        GET-KEY-EXIT.
     69              EXIT.

*** NO ERROR MESSAGES ***
```

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 09.11.35   DATE: 22.10.80
    SOURCE FILE: (TD)GENSEQ

      1        IDENTIFICATION DIVISION.
      2        PROGRAM-ID.
      3              GENSEQ.
      4       ************************************************************************
      5       *      CREATES SQ-FILE AND LISTS.
      6       ************************************************************************
      7        ENVIRONMENT DIVISION.
      8        INPUT-OUTPUT SECTION.
      9        FILE-CONTROL.
     10              SELECT SQ-FILE ASSIGN "COB1:DATA" ,
     11                             ORGANIZATION IS SEQUENTIAL,
     12                             ACCESS IS SEQUENTIAL.
     13        DATA DIVISION.
     14        FILE SECTION.
     15        FD    SQ-FILE
     16                LABEL RECORDS STANDARD
     17                DATA RECORDS M-REC.
     18        01  M-REC.
     19              02  FILLER PIC X(10).
     20              02  SEQNUM PIC 9(5).
     21              02  FILLER PIC X(5).
     22              02  FILLER PIC X(40).
     23         WORKING-STORAGE SECTION.
     24        01    RANDNO          COMP,  VALUE ZERO.
     25        01    MAXRAND         COMP,  VALUE 1000.
     26        01    NORECS          PIC 9(4).
     27        01    RECCNT          COMP, VALUE 0.
     28
     29        PROCEDURE DIVISION.
     30        INIT-01.
     31              OPEN OUTPUT SQ-FILE.
     32              DISPLAY 'CREATE  RECORDS ?
     33              PERFORM GET-NORECS.
     34
     35              PERFORM CRE-SQ-FILE NORECS TIMES.
     36       *                                    BUILD THE INPUT FILE
     37              CLOSE SQ-FILE.
     38              DISPLAY 'FILE SQ-FILE CREATED.', RECCNT, 'RECORDS.'.
     39              OPEN INPUT SQ-FILE.
     40        LIST-FILE-0.
     41              MOVE 0 TO RECCNT.
     42        LIST-FILE-1.
     43              READ SQ-FILE AT END  GO TO LIST-END.
     44              ADD 1 TO RECCNT.
     45              DISPLAY 'REC ', RECCNT, ' SEQNUM = ', SEQNUM.
     46              GO TO LIST-FILE-1.
     47        LIST-END.
     48              CLOSE SQ-FILE.
     49              DISPLAY "JOB FINISH".
     50              STOP RUN.
     51
     52        CRE-SQ-FILE.
     53              CALL 'RND' USING RANDNO, MAXRAND.
     54              MOVE ALL '*' TO M-REC.
     55              MOVE RANDNO TO SEQNUM.
     56              ADD 1 TO RECCNT.
     57              DISPLAY "UT REC =", RECCNT, " KEY =", SEQNUM.
     58              WRITE M-REC.
     59
     60        GET-NORECS.
     61              ACCEPT NORECS.
     62              IF NORECS NOT NUMERIC,
     63                             DISPLAY "** NOT NUMERIC DATA ",
     64                             GO TO GET-NORECS.

*** NO ERROR MESSAGES ***
```

6.7.1.10 The REWRITE Statement

The REWRITE statement logically replaces a record existing in a mass storage file.

Format 1.

REWRITE record-name [FROM identifier]

Format 2. Indexed and Relative I-O Only.

REWRITE record-name [FROM identifier]
[; INVALID KEY imperative-statement]

Record-name and identifier must not refer to the same storage area.

Record-name is the name of a logical record in the File Section of Data Division.

For Relative I-O, the INVALID KEY phrase must be specified in the REWRITE statement for files in the random or dynamic access mode for which an appropriate USE procedure is not specified. It must not be specified for a REWRITE statement for a file in sequential access mode.

For Indexed I-O, the INVALID KEY phrase must be specified in the REWRITE statement for files which do not have an appropriate USE procedure for them.

General Rules:

1. The file associated with record-name (which must be a mass-storage file for Sequential I-O) must be open in the I-O mode at the time of execution of the statement.

2. The last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement.

3. The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced.

4. The logical record released by a successful execution of the REWRITE statement is no longer available in the record area.

5. The execution of a REWRITE statement with the FROM phrase is equivalent to the execution of:

   MOVE identifier TO record-name

   followed by the execution of the same REWRITE statement without the FROM phrase. The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of the REWRITE statement.

6. The current record pointer is not affected by the execution of a REWRITE statement.

7. The execution of the REWRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated.

8. For Relative I-O, a file accessed in either random or dynamic access mode, the runtime-system logically replaces the record referenced by the RELATIVE KEY data item for the file. If this file does not contain this record, the INVALID KEY condition exists. The updating operation will not take place.

   The following rules are for Indexed I-O only:

9 For a file in the sequential access mode, the record to be replaced is indicated by the prime record key. When the REWRITE statement is executed the value in the prime record key data item of the record to be replaced must be the same as that of the last record read from this file. For a file in random or dynamic access mode, the record to be replaced is specified by the prime record key data item.

10. The contents of alternate record key data items of the record being rewritten may differ from those in the record being replaced. The runtime-system utilizes the content of the record key data items during the execution of the REWRITE statement in such a way that subsequent access of the record may be based upon any of those specified record keys.

11. The INVALID KEY condition exists when:

    a. The access mode is sequential and the value contained in the prime record key data item of the record to be replaced is not equal to the value of the prime record key of the last record read from this file, or

    b. The value contained in the prime record key data item does not equal that of any record stored in the file, or

    c. The value contained in an alternate record key data item for which a DUPLICATES clause has not been specified is equal to that of a record already stored in the file.

    The updating operation does not take place and the data in the record area is unaffected.

6.7.1.11 The START Statement

The START statement provides a basis for logical positioning within an indexed or relative file, for subsequent retrieval of records.

Format:

START file-name [KEY { IS EQUAL TO / IS = / IS GREATER THAN / IS > / IS NOT LESS THAN / IS NOT < } data-name]

[INVALID KEY imperative-statement]

Note: The required relational characters '>', '<', and '=' are not underlined to avoid confusion with other symbols such as '≥' (greater than or equal to).

File-name must be the name of a file with sequential or dynamic access.

The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.

Data-name may be qualified, and for Relative I-O, it must be the data item specified in the RELATIVE KEY phrase of the associated file-control entry.

General Rules:

1. File-name must be open in the INPUT or I-O mode at the time that the START statement is executed. (See the OPEN statement.)

2. If the KEY option is not specified, the relational operator 'IS EQUAL TO' is implied.

3. If the KEY option is present, the comparison specified in the KEY relational operator is made between data-name and the corresponding key field associated with the records of the file.

4. The execution of the START statement causes the current value in the key data-name and the corresponding key field of the file's records to be compared. The current record pointer is positioned to the logical record in the file whose key field satisfies the comparison. (If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists, the execution of the START statement is unsuccessful, and the position of the current record pointer is undefined.)

5. The execution of the START statement also causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. (See I-O Status at the beginning of this section.)

General Rules For Indexed Files:

1. If the KEY option is not specified, then the IS EQUAL TO comparison is made with the prime RECORD KEY data item. After successful execution of the START statement RECORD KEY or ALTERNATE RECORD KEY becomes the key of reference for subsequent READ statement.

2. If a KEY option is present, then the comparison is made with the data item which may be the prime RECORD KEY, and ALTERNATE RECORD KEY, or an alphanumeric data item subordinate to a record key having its leftmost character position corresponding to the leftmost character position of that record key.

3. The current record pointer is positioned as in general rule 4. If the operands in the comparison are of unequal length, the comparison takes place as if the longer field were truncated on the right of the length of the shorter field.

4. If the execution of the START statement is not successful, the key of reference is undefined.

General Rule For Relative Files:

1. The KEY data item used in the comparison is that associated with RELATIVE KEY, whether or not the KEY option appears. Thus, when the KEY option does not appear, the data-name must specify RELATIVE KEY.

```
NORD-10/100 COBOL COMPILER - VER  01.10.80       TIME: 09.17.45   DATE: 22.10.80
    SOURCE FILE: ISAM-EX2

 1        IDENTIFICATION DIVISION.
 2        PROGRAM-ID.
 3              GEN-ISAM-2.
 4       ***********************************************************************
 5       *      ISAM (INDEX SEQUENTIAL ACCESS METHOD ).
 6       *THE RECORD IS THE OUTPUT TO AN ISAM FILE USING THE *UNIQUE*
 7       *(IE: NO DUPLICATES ) DATA FOUND IN FIELD ISAM-KEY AS *KEY* VALUE.
 8       *
 9       *  BEFORE THIS JOB CAN BE RUN THE FOLLOWING *MUST* BE SO :
10       *      A) FILE "ISAM-EX:DATA" MUST EXIST.
11       *      B) FILE "ISAM-EX:ISAM" MUST NOT EXIST OR IF EXISTING
12       *                                CONTAIN *NO DATA !!*
13       ***********************************************************************
14        ENVIRONMENT DIVISION.
15        INPUT-OUTPUT SECTION.
16        FILE-CONTROL.
17              SELECT ISAM-FILE ASSIGN TO "ISAM-EX:DATA",
18                               ORGANIZATION IS INDEXED,
19                               ACCESS MODE IS DYNAMIC,
20                               RECORD KEY IS  ISAM-KEY,
21                               FILE STATUS IS ISAMSTATUS.
22        DATA  DIVISION.
23        FILE SECTION.
24
25        FD ISAM-FILE
26              RECORD CONTAINS 46 CHARACTERS,
27              DATA RECORD IS ISAM-REC.
28        01 ISAM-REC.
29              02 ISAM-KEY      PIC X(6).
30       *          :.....................MUST BE IN RECORD AREA !
31              02 ISAM-TEXT     PIC X(40).
32
33        WORKING-STORAGE SECTION.
34        01 ISAMSTATUS          PIC XX.
35       *                                RETURN STATUS FROM ISAM.
36       ***********************************************************************
37        PROCEDURE DIVISION.
38        A001.
39              OPEN I-O ISAM-FILE.
40        A002.
41              DISPLAY "ENTER KEY  (MAX 6 CHAR ) :",
42              ACCEPT ISAM-KEY.
43              IF ISAM-KEY = SPACES GO TO LIST.
44       *                                SPACES INPUT , END DIALOG
45              DISPLAY "ENTER TEXT (MAX 40 CHAR) :".
46              ACCEPT ISAM-TEXT.
47       *                                READ RECORDS FROM TERMINAL
48              WRITE ISAM-REC , INVALID KEY,
49                  DISPLAY "ISAM FILE ERROR :", ISAMSTATUS, ":".
50              GO TO A002.
51       *                                OUTPUT RECORD AND ASK AGAIN
52        LIST.
53              DISPLAY "ENTER ACCESS  KEY :".
54              ACCEPT ISAM-KEY.
55              IF ISAM-KEY  = SPACES  GO TO FINI.
56              START ISAM-FILE KEY IS EQUAL TO ISAM-KEY, INVALID KEY
57                              DISPLAY "** KEY NOT FOUND !",
58                              GO TO LIST.
59              READ ISAM-FILE RECORD, INVALID KEY
60                              DISPLAY "** RECORD NOT FOUND !",
61                              GO TO LIST.
62              DISPLAY "REC: ", ISAM-KEY, ": ", ISAM-REC.
63              GO TO LIST.
64        FINI.
65              CLOSE ISAM-FILE.
66              DISPLAY "JOB END".
67              STOP RUN.

*** NO ERROR MESSAGES ***
```

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 15.56.05   DATE: 21.10.80
    SOURCE FILE: REL-EX2

      1        IDENTIFICATION DIVISION.
      2        PROGRAM-ID.
      3              GEN-EX2.
      4        ENVIRONMENT DIVISION.
      5        INPUT-OUTPUT SECTION.
      6        FILE-CONTROL.
      7                SELECT RELFILE ASSIGN "RELATIVE-EX:DATA" ,
      8                         ORGANIZATION IS RELATIVE,
      9                         ACCESS IS DYNAMIC,
     10                         RELATIVE KEY IS REL-KEY,
     11                         FILE STATUS IS REL-STATUS.
     12        DATA DIVISION.
     13        FILE SECTION.
     14
     15        FD  RELFILE
     16                  LABEL RECORD IS OMITTED
     17                  DATA RECORD IS  REL-RECORD
     18                  BLOCK CONTAINS  10 RECORDS
     19                  RECORD CONTAINS 60 CHARACTERS.
     20        01  REL-RECORD       PIC X(60).
     21       *                       RECORD CANNOT BE "QED" TYPE RECORD
     22
     23        WORKING-STORAGE SECTION.
     24        01 REL-STATUS          PIC XX.
     25        01 REL-KEY             PIC 999.
     26       *           :..........CANNOT APPEAR IN RELFILE RECORD AREA,
     27       *                       MAX POSSIBLE SIZE IS 999999, RESTRICTED
     28       *                       TO 999 IN THIS PROGRAM.
     29
     30        PROCEDURE DIVISION.
     31
     32        A000.
     33              OPEN I-O RELFILE.
     34        A002.
     35              DISPLAY "ENTER KEY (MAX 999 ) :".
     36              PERFORM GET-KEY.
     37              IF REL-KEY = ZEROES GO TO A003.
     38              DISPLAY "ENTER TEXT ( MAX 60 CHAR ) :".
     39              ACCEPT REL-RECORD.
     40              WRITE REL-RECORD INVALID KEY,
     41                              DISPLAY " ** RELFILE ERROR :", REL-STATUS.
     42              GO TO A002.
     43        A003.
     44              DISPLAY "ENTER ACCESS KEY :".
     45              PERFORM GET-KEY.
     46              IF REL-KEY = ZEROS GO TO A999.
     47
     48              START RELFILE KEY IS EQUAL REL-KEY, INVALID KEY,
     49                          DISPLAY " ** RECORD NOT FOUND !", REL-STATUS,
     50                          GO TO A003.
     51
     52              READ RELFILE.
     53
     54              DISPLAY "REC :", REL-KEY, ":", REL-RECORD.
     55              GO TO A003.
     56        A999.
     57              CLOSE RELFILE.
     58              DISPLAY "JOB END".
     59              STOP RUN.
     60        GET-KEY.
     61              ACCEPT REL-KEY.
     62              IF REL-KEY NOT NUMERIC ,
     63                              DISPLAY " ** KEY MUST BE NUMERIC ",
     64                              GO TO GET-KEY.
     65        GET-KEY-EXIT.
     66              EXIT.

*** NO ERROR MESSAGES ***
```

6.7.1.12 The UNLOCK Statement

The UNLOCK statement unlocks records which have been locked in MANUAL-UNLOCK-MODE. Its format is:

Format: UNLOCK file name

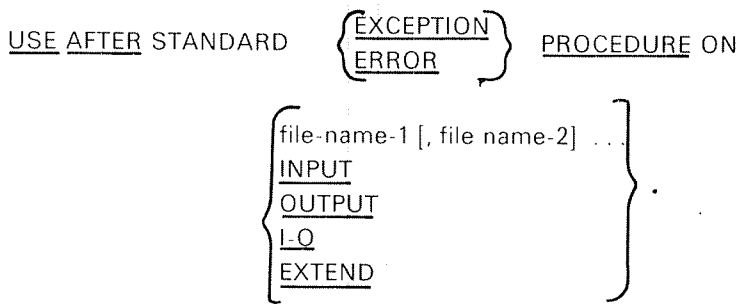
The UNLOCK statement is an ND Extension and it is used for programs accessing relative or indexed organized files in MULTI-USER-MODE (see the Open statement).

Records are normally unlocked automatically after a REWRITE or another READ on the same file, or when the file is closed. However the MANUAL-UNLOCK option on the OPEN statement, if present, will prevent this until an UNLOCK statement is encountered for the file.

6.7.1.13 The USE Statement

The USE statement specifies procedures for I-O error handling in addition to the standard procedures provided by the I-O control system.

Format:



The EXTEND option is valid for Sequential I-O only.

A USE statement, when present, must immediately follow a section header in the Declaratives Section of the Procedure Division. (See under Declaratives at the beginning of the Procedure Division description.)

The USE statement itself is never executed, it merely defines the conditions requiring execution of the USE procedure.

The files referenced need not all have the same organization or access.

THE EXCEPTION/ERROR PROCEDURE

This procedure is executed after completion of the standard system I-O routine or when an AT END or INVALID KEY option has not been specified in the input/output statement.

EXCEPTION/ERROR procedures are activated when:

a. An OPEN statement is issued for a file already in the open status, or for a nonexistent file.

b. A file is in the OPEN status and the execution of a CLOSE statement is unsuccessful.

c. If an I-O error occurs during execution of a READ, WRITE, REWRITE, START or DELETE statement.

After execution of the EXCEPTION/ERROR procedure, control is returned to the statement in the invoking routine following the statement which activated this procedure.

Within a USE procedure there must not be any reference to any non-declarative procedures. There is no interface between the two types. (However, a PERFORM statement may refer to a USE procedure.)

Within an EXCEPTION/ERROR procedure, no statement may be executed that would cause execution of a USE procedure that had been previously invoked and had not yet returned control to the invoking routine.

Note: EXCEPTION/ERROR procedures can be used to check the status key values whenever an input-output error occurs.

6.7.1.14 The WRITE Statement

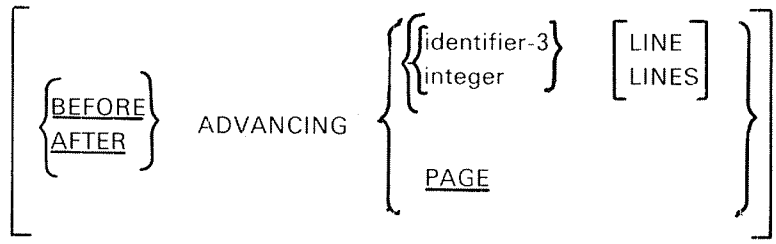
The WRITE statement releases a logical record for an output or an input-output file. For Sequential I-O it can be used for vertical positioning of lines within a logical line.

Format 1. Indexed and Relative I-O.

WRITE record-name [FROM identifier-1] [; INVALID KEY imperative-statement]

Format 2. Sequential I-O.

WRITE record-name [FROM identifier-2]



Record-name and identifiers 1 or 2 must not reference the same storage area.

For format 1, record-name is the name of a logical record in the File Section of the Data Division and may be qualified.

With format 1, the INVALID KEY phrase must appear if no USE procedure is specified for the associated file.

For format 2, when identifier-3 is used in the ADVANCING phrase, it must be the name of an elementary data item (whose value may be zero). Integer may also be zero.

General Rules:

1. For Indexed and Relative I-O, the associated file must be open in the OUTPUT or I-O mode at the time of execution of this statement. For Sequential I-O the file must be open in either OUTPUT or EXTEND modes.

2. The results of the execution of the WRITE statement with the FROM phrase are equivalent to the execution of:

   a. The statement:

      MOVE identifier TO record-name

   according to the rules specified for the MOVE statement, followed by:

   b. The same WRITE statement without the FROM phrase.

   The contents of the record area prior to the execution of the implicit MOVE statement have no effect on the execution of this WRITE statement.

   After execution of the WRITE statement is complete, the information in the area referenced by identifier is available, even though the information in the area referenced by record-name may not be.

3. The current record pointer is unaffected by the execution of a WRITE statement.

4. The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated.

5. The maximum record size for a file is established at the time the file is created and must not subsequently be changed.

6. The number of character positions on a mass storage device required to store a logical record in a file may or may not be equal to the number of character positions defined by the logical description of that record in the program.

7. The execution of the WRITE statement releases a logical record to the operating system.

General Rules For Indexed I-O:

1. Execution of the WRITE statement causes the contents of the record area to be released. The runtime-system utilizes the content of the record keys in such a way that subsequent access of the record key may be made based upon any of those specified record keys.

2. The value of the prime record key must be unique within the records in the file.

3. The data item specified as the prime record key must be set by the program to the desired value prior to the execution of the WRITE statement. (See general rule 2.)

4. If sequential access mode is specified for the file, records must be released to the runtime-system in ascending order of prime record key values.

5. If random or dynamic access mode is specified, records may be released to the runtime-system in any program-specified order.

6. When the ALTERNATE RECORD KEY clause is specified in the file control entry for an indexed file, the value of the alternate record key may be non-unique only if the DUPLICATES phrase is specified for that data item. In this case the runtime-system provides storage of records such that when records are accessed sequentially, the order of retrieval of those records is the order in which they are released to the runtime-system.

7. The INVALID KEY condition exists under the following circumstances:

   a. When sequential access mode is specified for a file opened in the output mode, and the value of the prime record key is not greater than the value of the prime record key of the previous record, or

   b. When the file is opened in the output or I-O mode, and the value of the prime record key is equal to the value of a prime record key of a record already existing in the file, or

   c. When the file is opened in the output or I-O mode, and the value of an alternate record key for which duplicates are not allowed equals the corresponding data item of a record already existing in the file, or

   d. When an attempt is made to write beyond the externally defined boundaries of the file.

8. When the INVALID KEY condition is recognized, the execution of the WRITE statement is unsuccessful, the contents of the record area are unaffected and the FILE STATUS data item, if any, associated with file-name of the associated file is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules given for the INVALID KEY condition.

General Rules For Relative I-O:

1. When a file is opened in the output mode, records may be placed into the file by one of the following:

   a. If the access mode is sequential, the WRITE statement will cause a record to be released to the runtime-system. The first record will have a relative record number of one (1) and subsequent records released will have relative record numbers of 2, 3, 4 . . . . If the RELATIVE KEY data item has been specified in the file control entry for the associated file, the relative record number of the record just released will be placed into the RELATIVE KEY data item by the runtime-system during execution of the WRITE statement.

   b. If the access mode is random or dynamic, prior to the execution of the WRITE statement the value of the RELATIVE KEY data item must be initialized in the program with the relative record number to be associated with the record in the record area. That record is then released to the runtime-system by execution of the WRITE statement.

2. When a file is opened in the I-O mode and the access mode is random or dynamic, records are to be inserted in the associated file. The value of the RELATIVE KEY data item must be initialized by the program with the relative record number to be associated with the record in the record area. Execution of a WRITE statement then causes the contents of the record area to be released to the runtime-system.

3. The INVALID KEY condition exists under the following circumstances:

   a. When the access mode is random or dynamic, and the RELATIVE KEY data item specifies a record which already exists in the file, or

   b. When an attempt is made to write beyond the externally defined boundaries of the file.

4. When the INVALID KEY condition is recognized, the execution of the WRITE statement is unsuccessful, the contents of the record area are unaffected, and the FILE STATUS data item, if any, of the associated file is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules given for the INVALID KEY condition.

General Rules For Sequential I-O:

1. The ADVANCING phrase allows control of the vertical positioning of each line on a printed page. If the ADVANCING phrase is not used, automatic advancing will act as if the user had specified AFTER ADVANCING 1 LINE. If the ADVANCING phrase is used, advancing is provided as follows:

   a. If identifier-3 is specified, the page is advanced the number of lines equal to the current value associated with identifier-3

   b. If interger is specified, the page is advanced the number of lines equal to the value of integer.

   c. If the BEFORE phrase is used, the line is presented before the page is advanced according to rules a and b above.

   d. If the AFTER phrase is used, the line is presented after the page is advanced according to rules a and b above.

   e. If PAGE is specified, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page.

2. When an attempt is made to write beyond the externally defined boundaries of a sequential file, an exception condition exists and the contents of the record area are unaffected. The following action takes place:

   a. The value of the FILE STATUS data item, if any, of the associated file is set to a value indicating a boundary violation.

   b. If an USE AFTER STANDARD EXCEPTION declarative is explicitly or implicitly specified for the file, that declarative procedure will then be executed.

   c. If an USE AFTER STANDARD EXCEPTION declarative is not explicitly or implicitly specified for the file, the result is undefined.

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 09.11.35   DATE: 22.10.80
    SOURCE FILE: (TD)GENSEQ

      1        IDENTIFICATION DIVISION.
      2        PROGRAM-ID.
      3                GENSEQ.
      4       ************************************************************************
      5       *       CREATES SQ-FILE AND LISTS.
      6       ************************************************************************
      7        ENVIRONMENT DIVISION.
      8        INPUT-OUTPUT SECTION.
      9        FILE-CONTROL.
     10                SELECT SQ-FILE ASSIGN "COB1:DATA" ,
     11                               ORGANIZATION IS SEQUENTIAL,
     12                               ACCESS IS SEQUENTIAL.
     13        DATA DIVISION.
     14        FILE SECTION.
     15        FD    SQ-FILE
     16                LABEL RECORDS STANDARD
     17                DATA RECORDS M-REC.
     18        01  M-REC.
     19              02  FILLER PIC X(10).
     20              02  SEQNUM PIC 9(5).
     21              02  FILLER PIC X(5).
     22              02  FILLER PIC X(40).
     23         WORKING-STORAGE SECTION.
     24        01    RANDNO          COMP,  VALUE ZERO.
     25        01    MAXRAND         COMP,  VALUE 1000.
     26        01    NORECS          PIC 9(4).
     27        01    RECCNT          COMP, VALUE 0.
     28
     29        PROCEDURE DIVISION.
     30        INIT-01.
     31              OPEN OUTPUT SQ-FILE.
     32              DISPLAY 'CREATE  RECORDS ?
     33              PERFORM GET-NORECS.
     34
     35              PERFORM CRE-SQ-FILE NORECS TIMES.
     36        *                                   BUILD THE INPUT FILE
     37              CLOSE SQ-FILE.
     38              DISPLAY 'FILE SQ-FILE CREATED.', RECCNT, 'RECORDS.'.
     39              OPEN INPUT SQ-FILE.
     40        LIST-FILE-0.
     41              MOVE 0 TO RECCNT.
     42        LIST-FILE-1.
     43              READ SQ-FILE AT END  GO TO LIST-END.
     44              ADD 1 TO RECCNT.
     45              DISPLAY 'REC ', RECCNT, ' SEQNUM = ', SEQNUM.
     46              GO TO LIST-FILE-1.
     47        LIST-END.
     48              CLOSE SQ-FILE.
     49              DISPLAY "JOB FINISH".
     50              STOP RUN.
     51
     52        CRE-SQ-FILE.
     53              CALL 'RND' USING RANDNO, MAXRAND.
     54              MOVE ALL '*' TO M-REC.
     55              MOVE RANDNO TO SEQNUM.
     56              ADD 1 TO RECCNT.
     57              DISPLAY "UT REC =", RECCNT, " KEY =", SEQNUM.
     58              WRITE M-REC.
     59
     60        GET-NORECS.
     61              ACCEPT NORECS.
     62              IF NORECS NOT NUMERIC,
     63                              DISPLAY "** NOT NUMERIC DATA ",
     64                              GO TO GET-NORECS.

*** NO ERROR MESSAGES ***
```

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 15.57.20   DATE: 21.10.80
    SOURCE FILE: ISAM-EX1

      1        IDENTIFICATION DIVISION.
      2        PROGRAM-ID.
      3              GEN-ISAM-1.
      4       ***********************************************************************
      5       *      ISAM (INDEX SEQUENTIAL ACCESS METHOD ).
      6       *THE RECORD IS THE OUTPUT TO AN ISAM FILE USING THE *UNIQUE*
      7       *(IE: NO DUPLICATES ) DATA FOUND IN FIELD ISAM-KEY AS *KEY* VALUE.
      8       *
      9       *  BEFORE THIS JOB CAN BE RUN THE FOLLOWING *MUST* BE SO :
     10       *      A) FILE "ISAM-EX:DATA" MUST EXIST.
     11       *      B) FILE "ISAM-EX:ISAM" MUST NOT EXIST OR IF EXISTING
     12       *                                CONTAIN *NO DATA !!*
     13       ***********************************************************************
     14        ENVIRONMENT DIVISION.
     15        INPUT-OUTPUT SECTION.
     16        FILE-CONTROL.
     17              SELECT ISAM-FILE ASSIGN TO "ISAM-EX:DATA",
     18                             ORGANIZATION IS INDEXED,
     19                             ACCESS MODE IS DYNAMIC,
     20                             RECORD KEY IS  ISAM-KEY,
     21                             FILE STATUS IS ISAMSTATUS.
     22        DATA  DIVISION.
     23        FILE SECTION.
     24
     25        FD ISAM-FILE
     26              RECORD CONTAINS 46 CHARACTERS,
     27              DATA RECORD IS ISAM-REC.
     28        01 ISAM-REC.
     29              02 ISAM-KEY     PIC X(6).
     30       *          :.....................MUST BE IN RECORD AREA !
     31              02 ISAM-TEXT     PIC X(40).
     32
     33        WORKING-STORAGE SECTION.
     34        01 ISAMSTATUS         PIC XX.
     35       *                                RETURN STATUS FROM ISAM.
     36       ***********************************************************************
     37        PROCEDURE DIVISION.
     38        A001.
     39              OPEN I-O ISAM-FILE.
     40        A002.
     41              DISPLAY "ENTER KEY  (MAX 6 CHAR ) :",
     42              ACCEPT ISAM-KEY.
     43              IF ISAM-KEY = SPACES GO TO LIST.
     44       *                                SPACES INPUT , END DIALOG
     45              DISPLAY "ENTER TEXT (MAX 40 CHAR) :".
     46              ACCEPT ISAM-TEXT.
     47       *                                READ RECORDS FROM TERMINAL
     48              WRITE ISAM-REC , INVALID KEY,
     49                  DISPLAY "ISAM FILE ERROR :", ISAMSTATUS, ":".
     50              GO TO A002.
     51       *                                OUTPUT RECORD AND ASK AGAIN
     52        LIST.
     53              DISPLAY "ENTER ACCESS  KEY :".
     54              ACCEPT ISAM-KEY.
     55              IF ISAM-KEY  = SPACES  GO TO FINI.
     56              READ ISAM-FILE RECORD KEY IS ISAM-KEY INVALID KEY,
     57                             DISPLAY "** RECORD NOT FOUND !",
     58                             GO TO LIST.
     59              DISPLAY "REC: ", ISAM-KEY, ": ", ISAM-REC.
     60              GO TO LIST.
     61        FINI.
     62              CLOSE ISAM-FILE.
     63              DISPLAY "JOB END".
     64              STOP RUN.

*** NO ERROR MESSAGES ***
```

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 09.06.39   DATE: 22.10.80
    SOURCE FILE: REL-EX

      1        IDENTIFICATION DIVISION.
      2        PROGRAM-ID.
      3              GENRELATIVE.
      4       ************************************************************************
      5       *     SHOWS THE USAGE OF A RELATIVE FILE :
      6       * THE FILE *MUST* EXIST BEFORE THE RUN BUT MAY BE EMPTY, EACH
      7       * RECORD IS LOCATED DIRECTLY BY ITS RELATIVE (TO 1) POSITION IN
      8       * THE FILE BY ITS *NUMERIC* KEY VALUE.
      9       ************************************************************************
     10        ENVIRONMENT DIVISION.
     11        INPUT-OUTPUT SECTION.
     12        FILE-CONTROL.
     13                SELECT RELFILE ASSIGN "RELATIVE-EX:DATA" ,
     14                          ORGANIZATION IS RELATIVE,
     15                          ACCESS IS DYNAMIC,
     16                          RELATIVE KEY IS REL-KEY,
     17                          FILE STATUS IS REL-STATUS.
     18        DATA DIVISION.
     19        FILE SECTION.
     20
     21        FD  RELFILE
     22                  LABEL RECORD IS OMITTED
     23                  DATA RECORD IS  REL-RECORD
     24                  BLOCK CONTAINS  10 RECORDS
     25                  RECORD CONTAINS 60 CHARACTERS.
     26        01  REL-RECORD       PIC X(60).
     27       *                       RECORD CANNOT BE "QED" TYPE RECORD
     28
     29        WORKING-STORAGE SECTION.
     30        01 REL-STATUS         PIC XX.
     31        01 REL-KEY            PIC 999.
     32       *           :.........CANNOT APPEAR IN RELFILE RECORD AREA,
     33       *                      MAX POSSIBLE SIZE IS 999999, RESTRICTED
     34       *                      TO 999 IN THIS PROGRAM.
     35
     36        PROCEDURE DIVISION.
     37
     38        A000.
     39              OPEN I-O RELFILE.
     40        A002.
     41              DISPLAY "ENTER KEY (MAX 999 ) :".
     42              PERFORM GET-KEY.
     43              IF REL-KEY = ZEROES GO TO A003.
     44              DISPLAY "ENTER TEXT ( MAX 60 CHAR ) :".
     45              ACCEPT REL-RECORD.
     46              WRITE REL-RECORD INVALID KEY,
     47                          DISPLAY " ** RELFILE ERROR :", REL-STATUS.
     48              GO TO A002.
     49        A003.
     50              DISPLAY "ENTER ACCESS KEY :".
     51              PERFORM GET-KEY.
     52              IF REL-KEY = ZEROS GO TO A999.
     53
     54              READ RELFILE RECORD INVALID KEY,
     55                          DISPLAY " ** RECORD NOT FOUND !", REL-STATUS,
     56                          GO TO A003.
     57              DISPLAY "REC :", REL-KEY, ":", REL-RECORD.
     58              GO TO A003.
     59        A999.
     60              CLOSE RELFILE.
     61              DISPLAY "JOB END".
     62              STOP RUN.
     63        GET-KEY.
     64              ACCEPT REL-KEY.
     65              IF REL-KEY NOT NUMERIC ,
     66                          DISPLAY " ** KEY MUST BE NUMERIC ",
     67                          GO TO GET-KEY.
     68        GET-KEY-EXIT.
     69              EXIT.

*** NO ERROR MESSAGES ***
```

6.8 PROCEDURE BRANCHING STATEMENTS

6.8.1 The ALTER Statement

Format

ALTER procedure-name-1 TO [PROCEED TO]procedure-name-2

[, procedure-name-3 TO [PROCEED TO]procedure-name-4] ...

and it is used to modify a simple GO TO statement elsewhere in the Procedure Division, thus changing the sequence of execution of program statements.

Each procedure-name-1, procedure-name-3,..., is the name of a COBOL paragraph that consists of only a simple GO TO statement.

Each procedure-name-2, procedure-name-4,..., is the name of a paragraph in the Procedure Divison.

The ALTER statement in effect replaces the former operand of that GO TO by procedure-name. Consider the ALTER statement in the context of the following program segment.

```
GATE.       GO TO MF—OPEN
MF—OPEN.    OPEN INPUT MASTER—FILE
            ALTER GATE TO PROCEED TO NORMAL
NORMAL.     READ MASTER—FILE, AT END GO TO EOF—MASTER
```

Examination of the above code reveals the technique of "shutting a gate", providing for a one-time, initializing-program step.

AVOID THE ALTER STATEMENT

The ALTER statement should not be used as it has a number of undesirable effects.

a) the object code produced will not be completely reentrant. depending on program structure this could increase dramatically the memory reguirements during execution.

b) the source listing will not show any *obvious* changes and thus be more difficult to debug.

6.8.2 The CONTINUE Statement

Format

CONTINUE

This statement has no effect and is treated as comments.

6.8.3 The EXIT Statement

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

Format 1.

EXIT

Format 2.

EXIT-DO

Format 3.

EXIT-ALL-DO

General Rules for Format 1:

1. An EXIT statement is used only when assigning a procedure-name to a given point in a program. Such an EXIT statement has no other effect on the compilation of the program.

2. An EXIT statement can be used to leave a DO --- END-DO loop.

General Rule for Formats 2 and 3:

1. The EXIT-DO statement in format 2 is used to leave the single DO-loop within which it appears. The EXIT-ALL-DO statement however, is used to leave all nested DO-loops within which it occurs. (See the DO-statement description, section 6.5.2.)

6.8.4 The GO TO Statement

The GO TO statement causes control to be transferred from one part of the Procedure Division to another.

Format 1.

GO TO [procedure-name-1]

Format 2.

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

Identifier is the name of a numeric elementary item described without any positions to the right of the assumed decimal point.

When a paragraph is referenced by an ALTER statement, that paragraph can consist only of a paragraph header followed by a Format 1 GO TO statement.

A Format 1 GO TO statement, without procedure-name-1, can only appear in a single statement paragraph.

If a GO TO statement represented by Format 1 appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

General Rules:

1. When a GO TO statement, represented by Format 1 is executed, control is transferred to procedure-name-1 or to another procedure-name if the GO TO statement has been modified by an ALTER statement.

2. If procedure-name-1 is not specified in Format 1, an ALTER statement, referring to this GO TO statement, must be executed prior to the execution of this GO TO statement.

3. When a GO TO statement represented by Format 2 is executed, control is transferred to procedure-name-1 procedure-name-2, etc., depending on the value of the identifier being 1, 2, ..., n. If the value of the identifier is anything other than the positive or unsigned integers 1,2 ..., n, then no transfer occurs and control passes to the next statement in the normal sequence for execution.

4. Integer n must be in the range 1 to 100.

5. The maximum number of procedure-names that can be specified with a GO TO statement is 100.

6.8.5 The PERFORM Statement

The PERFORM statement permits the execution of a separate body of program steps. Three formats of the PERFORM statement are available:

Format 1

```
PERFORM range [ { identifier-1 } TIMES]
               { integer      }
```

Format 2

```
PERFORM range  UNTIL  condition-1
```

Format 3

```
PERFORM range  VARYING { identifier-5 } FROM { identifier-6 }
                      { index-name-3  }      { index-name-4 }
                                             { literal-3    }

BY { identifier-7 } UNTIL condition-1 [AFTER { identifier-8 }
   { literal-4    }                         { index-name-5 }

FROM { identifier-9  } BY { identifier-10 } UNTIL condition-2
     { index-name-6  }    { literal-6    }
     { literal-5     }

[ AFTER { identifier-11 } FROM { identifier-12 }
        { index-name-7  }      { index-name-8  }
                               { literal-6     }

BY { identifier-13 } UNTIL condition-3 ] ]
   { literal-7     }
```

In the above syntax, range is the construct

```
procedure-name-1 [ { THROUGH } procedure-name-2 ]
                   { THRU    }
```

where THROUGH is synonomous with THRU.
Procedure-names 1 and 2 must have a section or paragraph in the Procedure Division.
Where both are specified, if either is a procedure-name inside Declaratives,then both must be procedure-names in Declaratives.

Each index-name identifies an index to be used in table references.
Each literal represents a numeric literal (in the BY phrase this must not be zero).
Condition-names 1,2 and 3 may be any conditional expressions (see under Conditional Expressions').
Each identifier must name an elementary numeric item.

Each index-name identifies an index to be used in table references.
Each literal represents a numeric literal (in the BY phrase this must not be zero).
Condition-names 1,2 and 3 may be any conditional expressions (see under Conditional Expressions').
Each identifier must name an elementary numeric item.

General Rules:

1. Whenever a PERFORM statement is executed, control is transferred to the first statement of the procedure named as procedure-1. Control is always returned to the statement following the PERFORM statement and the point from which it is returned is determined as follows:

   a. If procedure-name-1 is a pragraph name and a procedure-name-2 is not present, the return is made after the execution of the last statement of procedure-name-1.

   b. If procedure-name-1 is a section name and a procedure-name-2 is not present, the return is made after the execution of the last sentence of the last paragraph of procedure-name-1.

   c. If procedure-name-2 is present and it is a paragraph name, the return is made after the execution of the last statement of that paragraph.

   d. If procedure-name-2 is present and it is a section name, the return is made after the execution of the last sentence of the last paragraph in the section.

2. GO TO and PERFORM statements may be specified within the performed procedure. When the performed procedures include another PERFORM statement, the sequence of procedures associated with the embedded PERFORM statement must be included in or excluded from the performed procedures of the first PERFORM statement.

3. The TIMES option. Identifier-1, if used, must name an integer item. (If the integer is zero or negative when the PERFORM statement is initiated, control passes to the statement following the PERFORM statement). The procedure(s) referred to are executed the number of times specified by the integer or the value in identifier-1. Once the PERFORM statement has been initiated, any reference to identifier-1 cannot vary the number of times the procedures are executed.

4. The UNTIL option. The procedures referred to are performed until the condition is satisfied. Control is then passed to the next executable statement following the PERFORM statement. If the condition is already true when the PERFORM statement initiated then the specified procedure(s) are not executed.

5. The VARYING option. This increments or decrements idientifers or index-names until the condition(s) in the UNTIL option are satisfied when control is passed to the next executable statement following the PERFORM statement.

6. With format 3, when varying two identifiers, the AFTER variable (identifier-8) is set to the value of identifier-9. When condition-1 is evaluated, if it is true, control is transferred to the next executable statement. If false, range is executed once before identifier-8 is augmented by identifier-10 or literal-6. And so on.

6.8.6 Using the PERFORM Statement

With format 1, the designated range is performed (i.e., executed remotely) a fixed number of times, as determined by an integer or by the value of an integral data-item.

In format 2, identifier-2 is set to the value of literal-1 or the current value of identifier-3 at the beginning of the execution. If condition-1 is false the designated range is performed and then condition-1 is evaluated again. The cycle is repeated (augmenting data-name-2 with the current BY value) until condition-1 is true.

In format 3 we may now vary not only the object of the VARYING phrase but objects of the AFTER phrases as well.

Varying two identifiers we have:

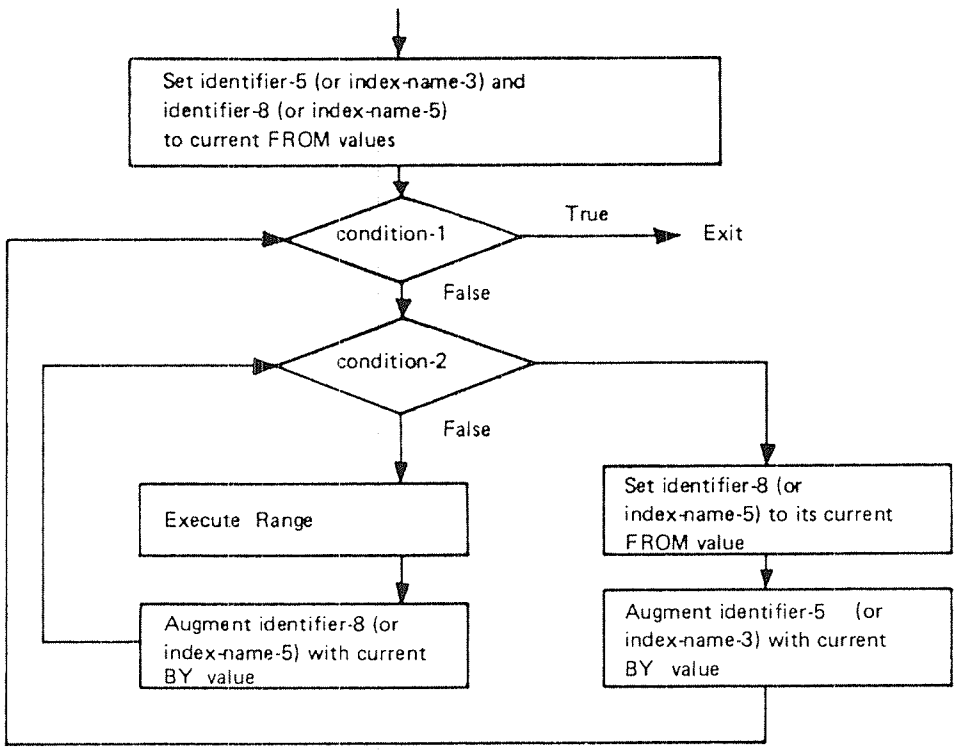


Figure 6.8:

Varying three identifiers gives us:

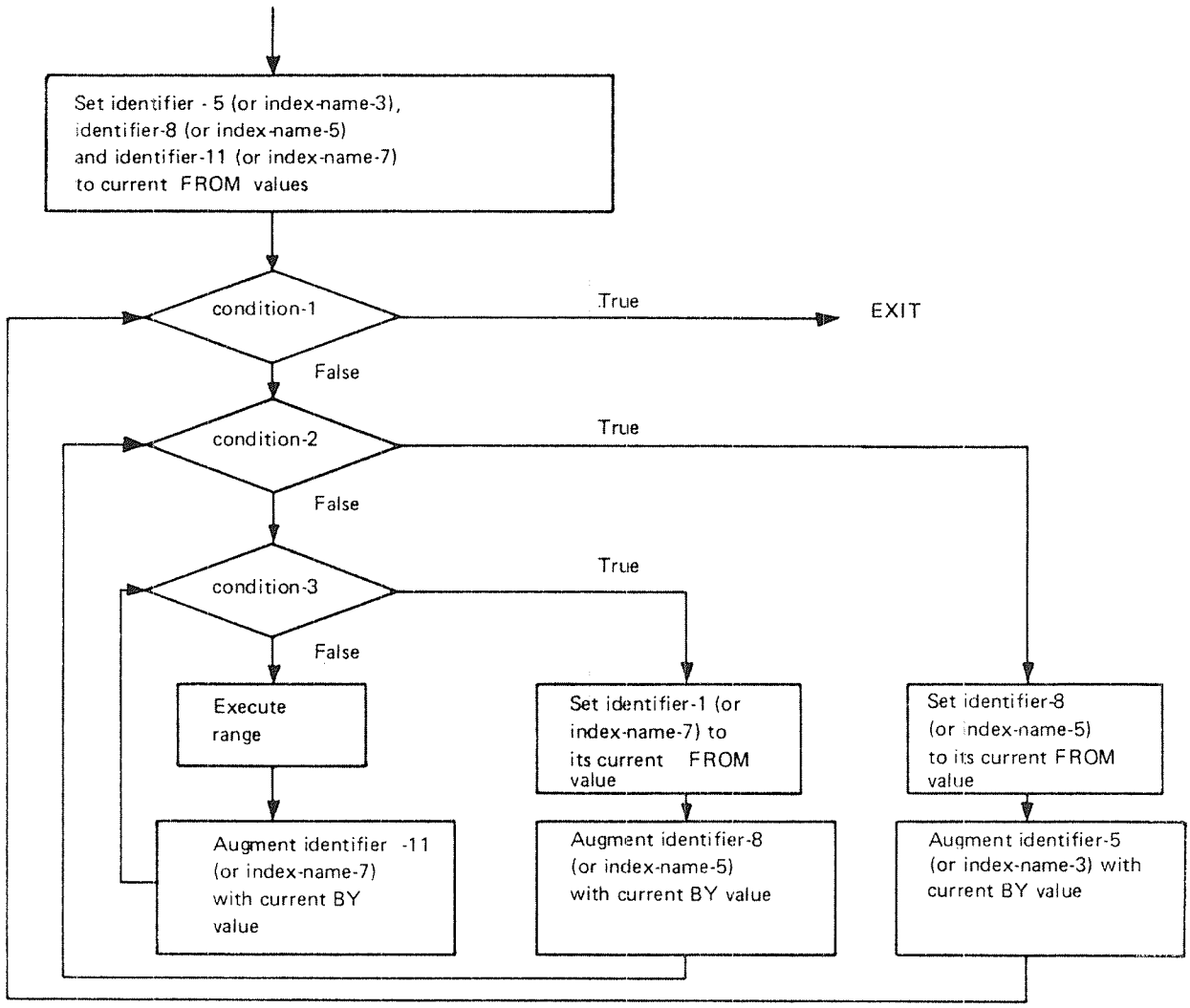


Figure 6.9:

The format-3 PERFORM statement is particularly useful in table handling when one statement can search a whole three dimensional table.

A run-time, it is illegal to have concurrently active perform ranges whose terminus points are the same.

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 09.11.35   DATE: 22.10.80
    SOURCE FILE: (TD)GENSEQ

      1        IDENTIFICATION DIVISION.
      2        PROGRAM-ID.
      3                GENSEQ.
      4       ******************************************************************
      5       *      CREATES SQ-FILE AND LISTS.
      6       ******************************************************************
      7        ENVIRONMENT DIVISION.
      8        INPUT-OUTPUT SECTION.
      9        FILE-CONTROL.
     10                SELECT SQ-FILE ASSIGN "COB1:DATA" ,
     11                               ORGANIZATION IS SEQUENTIAL,
     12                               ACCESS IS SEQUENTIAL.
     13        DATA DIVISION.
     14        FILE SECTION.
     15        FD    SQ-FILE
     16                LABEL RECORDS STANDARD
     17                DATA RECORDS M-REC.
     18        01  M-REC.
     19              02  FILLER PIC X(10).
     20              02  SEQNUM PIC 9(5).
     21              02  FILLER PIC X(5).
     22              02  FILLER PIC X(40).
     23         WORKING-STORAGE SECTION.
     24        01    RANDNO          COMP,  VALUE ZERO.
     25        01    MAXRAND         COMP,  VALUE 1000.
     26        01    NORECS          PIC 9(4).
     27        01    RECCNT          COMP, VALUE 0.
     28
     29        PROCEDURE DIVISION.
     30        INIT-01.
     31              OPEN OUTPUT SQ-FILE.
     32              DISPLAY 'CREATE  RECORDS ?.'
     33              PERFORM GET-NORECS.
     34
     35              PERFORM CRE-SQ-FILE NORECS TIMES.
     36        *                                   BUILD THE INPUT FILE
     37              CLOSE SQ-FILE.
     38              DISPLAY 'FILE SQ-FILE CREATED.', RECCNT, 'RECORDS.'.
     39              OPEN INPUT SQ-FILE.
     40        LIST-FILE-0.
     41              MOVE 0 TO RECCNT.
     42        LIST-FILE-1.
     43              READ SQ-FILE AT END  GO TO LIST-END.
     44              ADD 1 TO RECCNT.
     45              DISPLAY 'REC ', RECCNT, ' SEQNUM = ', SEQNUM.
     46              GO TO LIST-FILE-1.
     47        LIST-END.
     48              CLOSE SQ-FILE.
     49              DISPLAY "JOB FINISH".
     50              STOP RUN.
     51
     52        CRE-SQ-FILE.
     53              CALL 'RND' USING RANDNO, MAXRAND.
     54              MOVE ALL '*' TO M-REC.
     55              MOVE RANDNO TO SEQNUM.
     56              ADD 1 TO RECCNT.
     57              DISPLAY "UT REC =", RECCNT, " KEY =", SEQNUM.
     58              WRITE M-REC.
     59
     60        GET-NORECS.
     61              ACCEPT NORECS.
     62              IF NORECS NOT NUMERIC,
     63                               DISPLAY "** NOT NUMERIC DATA ",
     64                               GO TO GET-NORECS.

*** NO ERROR MESSAGES ***
```

6.8.7 The STOP Statement

The STOP statement is used to terminate or delay execution of the object program.

Format

STOP { RUN / literal }

STOP RUN terminates execution of a program, returning control to the operating system.

The form STOP literal causes the specified to be displayed on the console, and execution to be suspended. Execution of the program is resumed only after operator intervention. Presumably, the operator performs a function suggested by the content of the literal, prior to resuming program execution.

6.9 COMPILER DIRECTING STATEMENTS

6.9.1 The COPY Statement

Prewritten source programs can be included in a source program at compile time. These prewritten programs can be saved in user-created libraries without recoding and incorporated later in the COBOL program by means of the COPY statement.

Format

COPY file-name.

Where file-name is the name of a Sintran file. (Default file type SYMB).

The COPY statement must be preceded by a space and terminated by a period. It may occur any where in the source program where a character string or separator may occur. However, a COPY statement must not occur within a COPY statement.

The effect of processing a COPY statement is that the library text associated with file-name is copied into the source program, logically replacing the entire COPY statement beginning with the word COPY and ending with the period, inclusive.

PART II OTHER FEATURES

SORT/MERGE

TABLE HANDLING

INTER—PROGRAM COMMUNICATION

DEBUGGING WITH THE SYMBOLIC DEBUGGER

7 SORT/MERGE

Sort and Merge enable the programmer to order one or more files of records, or to combine two or more identically ordered files of records, according to a set of user-specified keys contained within each of these records.

COBOL has special language features for sorting and merging so that the programmer does not need to program these operations in detail.

7.1 SORT CONCEPTS

Sort produces an ordered file from one or more files that may be completely unordered with regard to the sort sequence.

A COBOL program containing a sort may have one or more input files handled by an input procedure. Within this procedure a RELEASE statement (analogous to a WRITE statement) places records one at a time onto the sort file. When all the records have been placed on this file the sorting operation is executed. All the sort file records are now arranged in the sequence specified by the keys.

Upon completion of the sorting operation, individual records can be accessed, one at a time, through a RETURN statement, should they need to be modified. If the user does not want to modify the sorted records, the SORT statement's GIVING option names the sorted output file.

7.2 MERGE CONCEPTS

Merge produces an ordered file from two or more input files, each of which is already ordered in the merge sequence.

The COBOL program can contain any number of merge operations each of which can have independent output procedures. After merging, individual records can be accessed by use of the RETURN statement, for modification if required. Otherwise, the GIVING option is used to name the merged output file.
Sort/Merge handles fixed or variable length records.

The files specified in the USING and GIVING phrases of the SORT/MERGE statement must be described in the FILE-CONTROL paragraph as having sequential organization. No I-O statement may be executed for the file named in the sort/merge file description.

7.3 SORT/MERGE—ENVIRONMENT DIVISION

File-control entries are required for each file to be used as input or output. A file-control entry is also required for the sort/merge file itself. For the I-O files the format is:

FILE—CONTROL. file-control entry [file-control entry] ...

For the sort file, the format of the allowable clauses in the file-control entry is:

SELECT file-name ASSIGN TO assignment-name-1.

Each sort/merge file described in the Data Division must be named once and once only in a file-control entry.

The ASSIGN clause associates a sort/merge file with a storage medium.

7.4 SORT/MERGE—DATA DIVISION

In the File Section there must be FD entries for each I-O file together with a record description entry. For each sort/merge file there must be an SD entry as well as a record description. The SD entry has the following format:

Format

SD file-name [; RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS

[DEPENDING ON identifier]]

[; RECORDING MODE IS { F / TEXT—FILE / T / V }]

[; DATA { RECORD IS / RECORDS ARE } data-name-1 [data-name-2] ...].

Where the file-name must specify a sort/merge file.

The RECORDS CONTAINS clause defines the size of the data records. As the size of each record is completely defined within the record description entry, this clause is never required. However, the number of characters in all fixed-length elementary items, plus the sum of the maximum number of those in any variable-length item subordinate to the record, determines its size.

The DATA RECORDS clause serves only as documentation for the names of the data records with their associated file.

Data-name-1 and data-name-2 are the names of data records which must have 01 level-number record descriptions, with the same names, associated with them. The presence of more than one data-name indicates that the file contains more than one type of data record which may be of differing sizes, formats etc.

7.5 SORT/MERGE — PROCEDURE DIVISION

A sort input procedure must contain a RELEASE statement to make each record available to the sorting operation. A sort/merge output operation must have a RETURN statement which makes a sorted/merged record available to the output procedure. The RELEASE statement has the format:

RELEASE record-name [FROM identifier]

Record-name must be the name of a logical record in the associated SD entry and may be qualified.

Record-name and identifier must not refer to the same storage area.

When the FROM option is used, the RELEASE statement is the equivalent of a MOVE statement operation of identifier to record-name, followed by a RELEASE statement operation for the record-name. Moving takes place according to the rules for the MOVE statement without the CORRESPONDING option. After the move, information in the record area is no longer available but that in the data area associated with the identifier may still be accessed.

When control passes from the Input Procedure the sort file consists of all those records placed in it by execution of RELEASE statements.

The RETURN statement obtains records from the final phase of a sort or merge operation. Its format is:

RETURN file-name RECORD [INTO identifier]

; AT END imperative-statement.

Within an Output Procedure at least one RETURN statement must be specified.

The file-name must be described by a Data Division SD entry.

The storage areas associated with the identifier and the record area of the file-name must not be the same.

The execution of a RETURN statement causes the next record, in the order specified by the keys listed in the SORT or MERGE statement, to become available by the Output Procedure. If more that one record description is associated with more than one file-name, these records share the same storage.

After the execution of a RETURN statement, only the contents of the current record are available; if any data items lie beyond the length of the current record their contents are undefined.

After all the records have been returned from file-name, the AT END imperative statement is executed and no further RETURN statements map be executed as part of the current output procedure.

7.5.1 The SORT Statement

The SORT statement creates a sort file by executing input procedures, or by transferring records from another file. It then sorts records in the sort file on a set of specified keys. In its final phase it makes each record from this file available, in sorted order, to some output procedure or to an output file. Its format is given below:

```
SORT file-name-1 ON { ASCENDING  } KEY data-name-1 [,data-name-2] ...
                    { DESCENDING }

                 ON { ASCENDING  } KEY data-name-3 [,data-name-4]...]...
                    { DESCENDING }

{ INPUT PROCEDURE IS section-name-1 [{ THROUGH } section-name-2 ] }
                                     { THRU    }
{ USING file-name-2                                               }

{ OUTPUT PROCEDURE IS section-name-3 [{ THROUGH } section-name-4 ] }
                                      { THRU    }
{ GIVING file-name-3                                               }
```

File-name-1 is the name given in the SD entry describing the records being sorted.

When the SORT statement is executed, all records contained on file-name 2 are sorted according to the specified keys. This input file must not be open at the time the SORT statement is executed; it is automatically opened and closed by the SORT operation (with any implicit functions also performed).

The INPUT PROCEDURE option specifies or more section-names of a procedure that is to modify input records before the sorting operation begins. Control is therefore passed to this procedure before file-name-1 is sequenced by the SORT statement. The compiler inserts a return mechanism at the end of the last section in the input procedure and when control passes the last statement of this procedure, the records that have been released to file-name-1 are sorted.

The input procedure must not contain any SORT statements or any transfer of control to points outside it. The execution of a CALL statement however follows standard linkage conventions.

7.5.2 Options Common to Sort and Merge

The ASCENDING/DESCENDING phrases specify that the records are to be processed in an ascending or descending sequence (whichever option is used) based on the specified sort keys.

The data items identified by KEY data-names must not contain an OCCURS clause or be subordinate to an entry containing as OCCURS clause.

Key data items must be of fixed length, they may be qualified but not subscripted or indexed.

If the USING phrase is specified, all records in file-name-2 for SORT (in file-names 2 and 3 for MERGE) are transferred automatically to file-name-1. At the time the SORT/MERGE statements are executed these files must not be open. The compiler opens, reads and makes records available and closes files automatically.

The OUTPUT PROCEDURE option specifies one or more section-names of a procedure that is to modify records from the sort or merge operation.

The procedure takes control when all records have been sorted/merged.

The compiler inserts a return mechanism at the end of the last section in the output procedure so that, when control passes the last statement of this procedure, the return mechanism causes control to pass to the next executable statement following the SORT or MERGE statement.

The output procedure must not itself contain any SORT/MERGE statements but it must include at least one RETURN statement to make the sorted/merged records available for processing.

The GIVING phrase causes all the sorted/merged records to be transferred to the output file. (File-name-3 for SORT operations, file-name-4 for MERGE operations).

When the SORT/MERGE statements are executed the output file must not be open. The compiler opens, reads and makes records available. The terminating function is performed as for a CLOSE statement.

7.5.3 The Merge Statement

The MERGE statement combines two identically sequenced files on a set of specified keys, and during the process makes records available in merge order, to an output file or procedure. It has the 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 }

  USING  file-name-2, file-name-3

{ OUTPUT PROCEDURE IS section-name-3 [ { THROUGH } section-name-4 ] }
                                       { THRU    }
{ GIVING  file-name-4                                               }
```

File-name-1 is the name given in the SD entry which describes the records being merged.

When the MERGE statement is executed, all records on file-names-2 and 3 are merged according to the key(s) specified. These files must not be open when this statement is executed; they are automatically opened and closed by the MERGE operation.

For the statement options see under the SORT statement.

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 09.18.10   DATE: 22.10.80
    SOURCE FILE: SORT-EX1

 1        IDENTIFICATION DIVISION.
 2        PROGRAM-ID.
 3                SORT-EX1.
 4       ***********************************************************************
 5       * CREATES IN-FILE,LISTS, SORTS,CREATING UT-FIL,LISTS.
 6       ***********************************************************************
 7        ENVIRONMENT DIVISION.
 8        INPUT-OUTPUT SECTION.
 9        FILE-CONTROL.
10                SELECT IN-FILE ASSIGN "COB1:DATA",
11                               ORGANIZATION IS SEQUENTIAL,
12                               ACCESS IS SEQUENTIAL.
13                SELECT UT-FILE ASSIGN "COB2:DATA".
14                SELECT S-FILE ASSIGN "SORT:DATA".
15        DATA DIVISION.
16        FILE SECTION.
17       ***********************************************************************
18       *  NOTE: ALL FILES,INPUT/OUTPUT AND SORT CAN BE FIXED (F),
19       *  VARIABLE (V) OR TEXT (T), EXCEPT RELATIVE AND INDEXED
20       *  THAT CAN'T BE TEXT.
21       ***********************************************************************
22        FD    IN-FILE
23                LABEL RECORDS STANDARD,
24                DATA RECORDS M-REC MOD-REC.
25        01  M-REC              PIC X(60).
26        01  MOD-REC.
27              02 FILLER        PIC X(10).
28              02 SEQNUM        PIC 9(5).
29              02 FILLER        PIC X(5).
30              02 FILLER        PIC X(40).
31        FD    UT-FILE
32                LABEL RECORDS STANDARD
33                DATA RECORD IS N-REC.
34        01    N-REC.
35              02 FILLER        PIC X(10).
36              02 SEQNUM2       PIC 9(5).
37              02 FILLER        PIC X(5).
38              02 FILLER        PIC X(40).
39
40       ***********************************************************************
41       * NOTE THAT THE SORT DESCRIPTOR (SD) DOES NOT NEED ANY
42       * FILE-DESCRIPTON-ENTRY, IF NOT RECORDING MODE T OR V
43       * IS USED.
44       ***********************************************************************
45        SD  S-FILE.
46        01  S-REC.
47            02 FILLER            PIC X(10).
48            02 S-KEY             PIC 9(5).
49            02 FILLER            PIC X(5).
50            02 FILLER            PIC X(40).
51
52        WORKING-STORAGE SECTION.
53        01    RANDNO              COMP,  VALUE ZERO.
54        01    MAXRAND             COMP,  VALUE 1000.
55        01    NORECS              PIC 9(4).
56        01    RECCNT              COMP,  VALUE 0.
57
58
```

```
59        PROCEDURE DIVISION.
60
61        MAIN SECTION.
62        INIT-01.
63              OPEN OUTPUT IN-FILE.
64              DISPLAY 'CREATE NO RECORDS ? (<9999 LEAD 0, S )'.
65              PERFORM GET-NORECS.
66              MOVE 0 TO RECCNT.
67
68              PERFORM CRE-IN-FILE NORECS TIMES.
69        *                           BUILD THE INPUT FILE FOR SORT
70              CLOSE IN-FILE.
71
72              DISPLAY 'FILE IN-FILE CREATED.', RECCNT, 'RECORDS.'.
73              MOVE 0 TO RECCNT.
74        ****************************************************************
75        *       ALL FILES REFERED TO BY THE SORT VERB MUST BE CLOSED
76        *       BEFORE THE SORT IS STARTED, OTHERWISE RUNTIME ERROR OCCURS
77        ****************************************************************
78              SORT S-FILE ON ASCENDING KEY S-KEY,
79                       USING IN-FILE
80                       GIVING UT-FILE.
81
82              OPEN INPUT UT-FILE
83              PERFORM LIST-UT-FILE.
84              CLOSE UT-FILE.
85              DISPLAY "JOB FINISH".
86              STOP RUN.
87        CRE-IN-FILE SECTION.
88        CRE-FILE-1.
89              CALL 'RND' USING RANDNO, MAXRAND.
90              MOVE ALL '*' TO M-REC.
91              MOVE RANDNO TO SEQNUM.
92              ADD 1 TO RECCNT.
93              DISPLAY "UT REC =", RECCNT, "KEY =", SEQNUM.
94              WRITE M-REC.
95        CRE-FILE-END.
96              EXIT.
97        LIST-UT-FILE SECTION.
98        LIST-FILE-0.
99              MOVE 0 TO RECCNT.
100       LIST-FILE-1.
101             READ UT-FILE AT END GO TO LIST-END.
102             ADD 1 TO RECCNT.
103             DISPLAY 'REC', RECCNT, 'SEQNUM = ',SEQNUM2.
104             GO TO LIST-FILE-1.
105       LIST-END.
106             EXIT.
107       GET-NORECS SECTION.
108       GET-NO.
109             ACCEPT NORECS.
110             IF NORECS NOT NUMERIC,
111                              DISPLAY '** NOT NUMERIC DATA '.,
112                              GO TO GET-NO.
113       GET-EXIT.
```

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 09.08.09   DATE: 22.10.80
    SOURCE FILE: SORT-EX2

 1        IDENTIFICATION DIVISION.
 2        PROGRAM-ID.
 3                SORT-EX2.
 4       ************************************************************************
 5       * CREATES IN-FILE,LISTS, SORTS USING PROCEDURES,CREATING UT-FILE.
 6       ************************************************************************
 7        ENVIRONMENT DIVISION.
 8        INPUT-OUTPUT SECTION.
 9        FILE-CONTROL.
10                SELECT IN-FILE ASSIGN "COB1:DATA",
11                               FILE STATUS IS IN-FILE-STATUS.
12                SELECT UT-FILE ASSIGN "COB2:DATA",
13                               FILE STATUS IS UT-FILE-STATUS.
14                SELECT S-FILE ASSIGN "SORT:DATA",
15                               FILE STATUS IS SFILE-STATUS.
16        DATA DIVISION.
17        FILE SECTION.
18        FD    IN-FILE
19                BLOCK CONTAINS 10 RECORDS,
20                DATA RECORDS IN-REC.
21        01  IN-REC.
22              02  FILLER      PIC X(10).
23              02  SEQNUM      PIC 9(5).
24              02  FILLER      PIC X(5).
25              02  FILLER      PIC X(80).
26        FD    UT-FILE
27                BLOCK CONTAINS 10 RECORDS.
28        01    UT-REC.
29              02  FILLER      PIC X(10).
30              02  SEQNUM2     PIC 9(5).
31              02  FILLER      PIC X(5).
32              02  FILLER      PIC X(80).
33        SD    S-FILE
34              .
35        01    S-REC.
36              02  FILLER      PIC X(10).
37              02  S-KEY       PIC 9(5).
38              02  FILLER      PIC X(5).
39              02  FILLER      PIC X(80).
40         WORKING-STORAGE SECTION.
41        01    NORECS          PIC 9(4).
42        01    RECCNT          COMP , VALUE 0.
43       ************************************************************************
44       * PARAMETERS FOR CALL TO RND (RANDOM NUMBER GENERATOR)
45       ************************************************************************
46        01    RANDNO          COMP,  VALUE ZERO.
47        01    MAXRAND         COMP,  VALUE 1000.
48       ************************************************************************
49       *  STATUS DATA-NAME(S) MUST BE DEFINED AND MUST BE 2 BYTES LONG
50       ************************************************************************
51        01  IN-FILE-STATUS      PIC XX.
52        01  UT-FILE-STATUS      PIC XX.
53        01  SFILE-STATUS        PIC XX.
54       ************************************************************************
55       * START/END-TIME USED FOR ACCEPTING TIME FROM SYSTEM
56       ************************************************************************
57        01  START-TIME          PIC 9(8).
58        01  END-TIME            PIC 9(8).
59       ************************************************************************
60       * SORT-START/END ARE THE RECIEVING EDIT FIELDS FOR START/END-TIME
61       ************************************************************************
62        01  SORT-START          PIC 99,99,99,99.
63        01  SORT-END            PIC 99,99,99,99.
64
65        PROCEDURE DIVISION.
66
```

```
67        MAIN SECTION.
68        INIT-01.
69              OPEN OUTPUT IN-FILE.
70              DISPLAY 'CREATE NO RECORDS ?'.
71              PERFORM GET-NORECS.
72              MOVE 0 TO RECCNT.
73
74              PERFORM CRE-IN-FILE NORECS TIMES.
75              CLOSE IN-FILE.
76
77              DISPLAY 'IN-FILE CREATED.', RECCNT, 'RECORDS.'.
78              MOVE 0 TO RECCNT.
79              ACCEPT START-TIME FROM TIME.
80
81              SORT S-FILE ON ASCENDING KEY S-KEY,
82                        INPUT  PROCEDURE IS SORT-PROC-IN
83                        OUTPUT PROCEDURE IS SORT-PROC-UT.
84
85              ACCEPT END-TIME FROM TIME.
86              PERFORM SORT-TIMES.
87              DISPLAY "JOB FINISH".
88              STOP RUN.
89        MAIN-END.
90              EXIT.
91        ****************************************************************
92        * CALLING 'RND' TO GENERATE RANDOM DATA FOR THE RECORD, WRITES
93        ****************************************************************
94         CRE-IN-FILE SECTION.
95         CRE-FILE-1.
96              CALL 'RND' USING RANDNO, MAXRAND.
97              MOVE RANDNO TO SEQNUM.
98              ADD 1 TO RECCNT.
99              DISPLAY "UT REC =", RECCNT, "KEY =", SEQNUM.
100             WRITE IN-REC.
101        CRE-FILE-END.
102             EXIT.
103       ****************************************************************
104       *       MOVE SORT TIMES INTO EDIT FIELDS FOR DISPLAYING
105       ****************************************************************
106        SORT-TIMES SECTION.
107        SORTT.
108             MOVE START-TIME TO SORT-START.
109             MOVE END-TIME   TO SORT-END.
110             DISPLAY "START SORT AT :", SORT-START.
111             DISPLAY "END   SORT AT :", SORT-END.
112        SORT-TIMES-END.
113             EXIT.
114       ****************************************************************
115       *       CALLED ONLY FROM THE SORT VERB TO READ AND PASS RECORDS
116       *       FROM THE IN-FILE INTO THE SORT.
117       *       IN-FILE IS OPENED/READ/CLOSED WITHIN THE ROUTINE
118       ****************************************************************
119        SORT-PROC-IN SECTION.
120        SORTIN.
121             DISPLAY ":::::::::::  SORT-PROC-IN START ::::::::::".
122             OPEN INPUT IN-FILE.
123        SORTIN-1.
124             READ IN-FILE AT END GO TO SORT-IN-END.
125       *================================================================
126       *       THE RELEASE PASSES THE RECORD INTO THE SORT
127       *================================================================
128             RELEASE S-REC FROM IN-REC.
129             GO TO SORTIN-1.
130        SORT-IN-END.
131             CLOSE IN-FILE.
132             DISPLAY ":::::::::::  SORT-PROC-IN ENDED ::::::::::".
133        SORT-IN-FINI.
134             EXIT.
```

```
135
136     ****************************************************************
137     *       CALL ONLY FROM THE SORT VERB TO ACCEPT RECORDS AND WRITE
138     *       THEM ONTO  UT-FILE.
139     *       UT-FILE IS OPENED/WRITTEN/CLOSED WITHIN THE ROUTINE.
140     ****************************************************************
141      SORT-PROC-UT SECTION.
142      SORTUT.
143            DISPLAY "**********  SORT-PROC-UT START **********".
144            MOVE 1 TO RECCNT.
145            OPEN OUTPUT UT-FILE.
146      SORTUT1.
147     *================================================================
148     *       RETURN PASSES A SORTED RECORD FROM THE SORT INTO PROGRAM
149     *================================================================
150            RETURN S-FILE INTO UT-REC, AT END ,
151                    DISPLAY "SFILE-ERR, STATUS :=:", SFILE-STATUS,
152                            GO TO SORTUT-END.
153            DISPLAY 'REC ', RECCNT, ' SEQNUM = ', SEQNUM2 ,
154                          ' STATUS =', UT-FILE-STATUS.
155            WRITE UT-REC.
156            ADD 1 TO RECCNT.
157            GO TO SORTUT1.
158      SORTUT-END.
159            CLOSE UT-FILE.
160            DISPLAY "**********  SORT-PROC-UT ENDED **********".
161      SORT-FINI.
162            EXIT.
163     ****************************************************************
164      GET-NORECS SECTION.
165      GET-NO.
166            ACCEPT NORECS.
167            IF NORECS NOT NUMERIC ,
168                            DISPLAY "** NOT NUMERIC DATA",
169                            GO TO GET-NO.
170      GET-EXIT.
171            EXIT.
```

```
@NRL
RELOCATING LOADER LDR-1935F
*LOAD SORT-EX2 RND COBLIB
FREE: 077447-177777
*RUN

CREATE NO RECORDS ?  001A
** NOT NUMERIC DATA  10
UT REC =00001 KEY = 00258
UT REC =00002 KEY = 00615
UT REC =00003 KEY = 00158
UT REC =00004 KEY = 00320
UT REC =00005 KEY = 00501
UT REC =00006 KEY = 00746
UT REC =00007 KEY = 00564
UT REC =00008 KEY = 00195
UT REC =00009 KEY = 00894
UT REC =00010 KEY = 00047
IN-FILE CREATED.00010 RECORDS.
::::::::::  SORT-PROC-IN START ::::::::::
::::::::::  SORT-PROC-IN ENDED ::::::::::
**********  SORT-PROC-UT START **********
REC 00001  SEQNUM = 00047 STATUS =00
REC 00002  SEQNUM = 00158 STATUS =00
REC 00003  SEQNUM = 00195 STATUS =00
REC 00004  SEQNUM = 00258 STATUS =00
REC 00005  SEQNUM = 00320 STATUS =00
REC 00006  SEQNUM = 00501 STATUS =00
REC 00007  SEQNUM = 00564 STATUS =00
REC 00008  SEQNUM = 00615 STATUS =00
REC 00009  SEQNUM = 00746 STATUS =00
REC 00010  SEQNUM = 00894 STATUS =00
SFILE-ERR, STATUS :=:10
**********  SORT-PROC-UT ENDED **********
START SORT AT :14,35,30,72
END   SORT AT :14,35,52,42
JOB FINISH
```

8 TABLE HANDLING

A table is a set of contiguous data items having the same data description.

Tables of data are common components of business data processing problems. Although items of data that make up a table could be described as contiguous data items, there are two reasons why this approach is not satisfactory. First, from a documentation standpoint, the underlying homogeneity of the items would not be readily apparent; and second, the problem of making available an individual element of such a table would be severe when there is a decision as to which element is to be made available at object time.

In COBOL a table is defined with an OCCURS clause in its data description entry. This clause specifies that the named item is to be repeated as many times as stated. The item so named is considered to be a *table element* and its name and description apply to each repetition (or occurrence) of the item. Since the occurences do not have unique data-names, reference to a particular occurrence can only be made by giving the data-names of the table element, together with the occurrence number of the required item within the element.

The occurrence number is known as a *subscript* and the method of supplying this number for individual table elements is called *subscripting.* A related technique for table referencing is called *indexing* and both of these methods of specifying occurrence numbers are described in this section.

8.1 TABLE DEFINITION

COBOL allows tables in one, two, or three dimensions.

To define a *one-dimensional table,* the programmer uses an OCCURS clause as part of the data description of the table element, but the OCCURS clause must not appear in the description of group items which contain the table element.

Example:

```
01  TABLE-1.
    02  ELEMENT-1 OCCURS 20 TIMES.
        03  ELEMENT-A  PIC X (2).
        03  ELEMENT-B  PIC 9 (5).
```

TABLE-1 is the group element containing the table. ELEMENT-1 names a table element of a one-dimensional table which occurs 20 times. ELEMENT-A and ELEMENT-B are elementary items.

Defining a one-dimensional table within each occurrence of an element of another one-dimensional table gives rise to a *two-dimensional table.* To define a two-dimensional table, then, an OCCURS clause must appear in the data description of the element of the table, and in the description of only one group item which contains that table element.

Example:

```
01  TABLE-2.
    02  ELEMENT-1 OCCURS 5 TIMES.
        03  ELEMENT-2 OCCURS 4 TIMES.
            04  ELEMENT-A   PIC 9(10).
            04  ELEMENT-B   PIC X(5).
```

ELEMENT-1 is an element of a one-dimensional table occurring five times. ELEMENT-2 is an element of a two-dimensional table occurring four times within each occurrence of ELEMENT-1.

To define a *three-dimensional table,* the OCCURS clause should appear in the data description of the element of the table and in the description of 2 group items which contain the element.

Example:

```
01  CENSUS TABLE.
    05  CONTINENT—TABLE OCCURS 6 TIMES.
        10  CONTINENT—NAME                          PIC X( 9).
        10  COUNTRY—TABLE OCCURS 5 TIMES.
            15  COUNTRY—NAME                        PIC X(12).
            15  CITY—TABLE OCCURS 100 TIMES.
                20  CITY—NAME                       PIC X( 4).
                20  CITY—POPULATION                 PIC X( 5).
```

In the above example we have a table of one dimension for CONTINENT—NAME, two dimensions for COUNTRY—NAME and three dimensions for CITY—NAME and CITY—POPULATION.

8.1.1 Table References

Whenever the user refers to a table element, the reference must indicate which occurrence of the element is intended. For access to a one-dimensional table, the occurrence number of the desired element provides complete information. For access to tables of more than one dimension, an occurrence number must be supplied for each dimension of the table accessed. In the last example then, a reference to the 4th CONTINENT—NAME would be complete, where as a reference to the 4th COUNTRY—NAME would not. To refer to COUNTRY—NAME, which is an element of a two-dimensional table, the user must refer to, for example, the 4th COUNTRY—NAME within the 6th CONTINENT—TABLE.

One method by which occurrence numbers may be specified is to append one or more subscripts to the data-name. A subscript is an integer whose value specifies the occurrence number of an element. The subscript can be represented either by a literal which is an integer or by a data-name which is defined elsewhere as a numeric elementary item with no character positions to the right of the assumed decimal point. In either case, the subscript, enclosed in parentheses, is written immediately following the name of the table element. A table reference must include as many subscripts as there are dimensions in the table whose element is being referenced. That is, there must be a subscript for each OCCURS clause in the hierarchy containing the data-name, including the data-name itself. In the example, references to CONTINENT—NAME require only one subscript, reference to COUNTRY—NAME requires two, and references to CITY—NAME requires two, and references to CITY—NAME and CITY—POPULATION require three.

When more than one subscript is required, they are written in order of successively less inclusive dimensions of the data organization. When a data-name is used as a subscript, it may be used to refer to items in many different tables. These tables need not have elements of the same size. The data-name may also appear as the only subscript with one item and as one of two or three subscripts with another item. Also, it is permissible to mix literal and data-name subscripts, for example: CITY—POPULATION(4, NEWKEY, 42).

8.1.1.1 Subscripting

Subscripting is the method of providing table references using subscripts. A *subscript* is an integer value specifying the occurrence number of a table element.

The format is:

```
{ data-name      } (subscript-1  [, subscript-2 [ , subscript-3 ] ]  )
{ condition-name }
```

Subscripts can only be used when referring to an individual item within a table element.

Data-name must be the name of a table element and may be qualified.

The subscript can be represented by a literal or a data-name.

If a literal, a subscript must be an integer having a value $\geqslant 1$. It must not be negative.

If a data-name, a subscript must be described as elementary numeric integer.

Where more that one subscript is required, the subscripts are written in the order of successively less-inclusive data dimensions. Each subscript must be separated from the next by either a space, or a comma followed by a space. (The comma is not required).

8.1.1.2 Indexing

Another method of referring to items in a table is indexing. An *index* is a compiler-generated storage area used to store table element occurrence numbers; the index contains a displacement value from the beginning of the table element that is equivalent to an occurrence number.

Its format is:

```
{data-name     }   (({index-name-1  [ { + } literal-2 ]}
{condition-name}   ({literal-1       [ { - }           }
    [{index-name-2  [ { + } literal-4 ]}
    [{literal-3     [ { - }           }
        [{index-name-3  [ { + } literal-6 ]}]]])
        [{literal-5     [ { - }           }]]
```

The index-name is specified through the OCCURS clause. It must be initialized by the SET statement before use.

In *direct indexing,* the index-name is in the form of a subscript. The value contained in the index is calculated as the occurrence number minus one, multiplied by the length of the individual table entry. For Example:

```
03  ELEMENT OCCURS 20 INDEXED BY INDX-1  PIC X(2).
```

The tenth occurrence of ELEMENT generates a value in INDX-1 of (10-1) * 2 = 18.

With *relative indexing,* the index-name is followed by a space, followed by a + or —, followed by another space, followed by an unsigned literal. The literal (i.e., occurrence number) is converted to an index value before being added to or subtracted from the index-name index.

For example, if we have:

```
01  TABLE-3.
  02  ELEMENT-1 OCCURS 2 TIMES INDEXED BY INDX-1.
    03  ELEMENT-2 OCCURS 3 TIMES INDEXED BY INDX-2.
      04  ELEMENT-3 OCCURS 2 TIMES INDEXED BY INDX-3 PIC X(5).
```

Then, each occurrence of ELEMENT-1 is 30 characters in length (3 * 2 * 5). Each occurrence of ELEMENT-2 is 10 characters in length (2 * 5) and each occurrence of ELEMENT-3 is 5 characters in length.

A reference using relative indexing such as

```
ELEMENT-3  (INDX-1 + 1, INDX-2 — 1, INDX-3 + 2)
```

would produce the computation for the displacement of:

```
    (address of ELEMENT-3)
+   ((contents of INDX-1) +1—1) *30
+   ((contents of INDX-2) —1—1) *10
+   ((contents of INDX-3) +2—1) *5
```

8.2 TABLE HANDLING — DATA DIVISION

The clauses used for Table Handling are OCCURS and USAGE IS INDEX.

8.2.1 The OCCURS Clause

This clause eliminates the need for separate entries for repeated data items and supplies information required for the application of subscripts and indexes.

Format 1 (Fixed length Tables)

OCCURS integer-2 TIMES

[{ ASCENDING / DESCENDING } KEY IS data-name-2 [, data-name-3] ... ...]

[INDEXED BY index-name-1 [, index-name-2] ...]

Format 2 (Variable length Tables)

OCCURS integer-1 TO integer-2 TIMES DEPENDING ON data-name-1

[{ ASCENDING / DESCENDING } KEY IS data-name-2 [, data-name-3] ... ...]

[INDEXED BY index-name-1 [, index-name-2] ...]

Integer-1, when used, must be less than integer-2. All integers must be positive.

If the data-name, which is the subject of this entry, or an entry subordinate to it is to be referred to by indexing, then the INDEXED BY clause is required.

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

a. has a level-number of 01, 77, or 88

b. described an item of variable size, i.e., if any subordinate item contains an OCCURS DEPENDING ON clause.

8.2.2 Fixed Length Tables - Format 1

In format 1, the value of integer-2 specifies the exact number of occurrences.

8.2.3 Variable Length Tables - Format 2.

This format specifies that the subject of this entry has a variable number of occurrences. The current value of the data item referenced by data-name-1 represents the maximum number of occurrences and the value of integer-1 the minimum.

The value of the data item referenced by data-name-1 must fall within the range integer-1 through integer-2.

The ASCENDING/DESCENDING KEY option (both formats) specifies that the repeated data is arranged in ascending or descending order according to the values contained in data-name-2, data-name-3 etc. (The order is determined according to the rules for comparison of operands - see Comparison of Numeric and Nonnumeric operands under Conditional Statements in the Procedure Division description.)

The data-names are listed in their descending order of significance.

8.2.4 The USAGE Clause

The USAGE IS INDEX clause specifies that the data item has an index format.

Format

[USAGE IS] INDEX

The data item is an *index data item* and is treated as computational it will occupy 2 bytes in storage.

An index data item can be referenced explicity only in a SEARCH or SET statement, a relation condition, the USING phrase of a Procedure Division header, or the USING phrase of a CALL statement.

The USAGE clause can be written at any level, if written at group level it applies to every elementary item in the group. (The USAGE clause of a elementary item cannot contradict the USAGE clause of a group to which the item belongs).

An index data item can be part of a group referred to in a MOVE or input - output statement, in which case no conversion will take place.

8.3 TABLE HANDLING — PROCEDURE DIVISION

In the Procedure Division, Table Handing makes use of the SEARCH and SET statements. Also, comparisons may be made between index-names and/or index data items as described under 'Relation Conditions' below.

RELATION CONDITIONS.

Comparison tests may be made between:

1. Two index-names. This is equivalent to comparing their occurrence numbers.

2. An index-name and a data item. The occurrence number correspoinding to the value of the index-name is compared to the data item or literal.

3. An index data item and an index name or another index data item. The actual values are compared without conversion.

4. The results of any other comparison involving an index data item are undefined.

8.3.1 The SEARCH Statement

Data that has been arranged in the form of a table is very often searched. In COBOL the SEARCH statement provides facilities, through its two options, for producing serial and non-serial searches. In using the SEARCH statement, the programmer may vary an associated index-name or data-name. This statement also provides facilities for execution of imperative statements when certain conditions are true.

Format 1

```
SEARCH identifier-1 [ VARYING { identifier-2  } ]
                              { index-name-1  }
    [; AT END imperative-statement-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]

     ; WHEN { data-name-1      } { IS EQUAL TO } { identifier-3 }
            { condition-name-1 } { IS =        } { literal-1    }
                                                 { arithmetic-expression-1 }
   [ AND    { data-name-2      } { IS EQUAL TO } { identifier-4 }
            { condition-name-2 } { IS =        } { literal-2    }
                                                 { arithmetic-expression-2 } ]  ...

   { imperative-statement-2 }
   { NEXT SENTENCE          }
```

NOTE: The required relational character '=' is not underlined to avoid confusion with other symbols.

The SEARCH statement searches a table for an element that satifies the specified condition, and adjusts the associated index to indicate that element.

In both formats, identifier-1 must not be subscripted or indexed, but its description in the Data Division must contain an OCCURS clause and an INDEXED BY clause.

Identifier-2, if present, must be described as USAGE IS INDEX or as a numeric elementary item without any positions to the right of the assumed decimal point.

Format 1

1. The search operation begins at the current index setting. If, at this point, the value of the index-name associated with identifier-1 is not greater than the highest possible occurrence number, the following takes, place:
   a. The conditions in the WHEN option are evaluated in the order in which they are written, making use of the index settings wherever specified.

   b. If none of the conditions are satisfied, the index-name for identifier-1 is incremented to correspond to the next table element. Then process a. is repeated.

   c. If one of the conditions is satisfied upon evalutation, the search terminates immediately and the imperative statement associated with that condition is executed. The index-name remains pointing to the table element that caused the condition.

   d. If, however, the incremented index-name value is greater than the highest possible occurrence number (i.e., the end of the table has been reached) the search terminates. If the AT END phrase is specified, imperative-statement-1 is now executed. Otherwise, control passes to the next executable sentence.

2. At the beginning of the search operation, if the value of the index-name associated with identifier-1 is greater than the highest possible occurrence number, then the search terminates as in step d above.

3. When the VARYING phrase is not used, the index that is used for the search operation is the first (or only) index-name given in the INDEXED BY phrase of identifier-1.

4. If the VARYING index-name-1 option appears then one of the following applies:

   a. When index-name-1 is the index for identifier-1, then this index is used for the search. If this is not the case (or the VARYING identifier-2 is present) the first, or only index-name is used.

   b. If index-name-1 is an index for another table element, then the first (or only) index-name for identifier-1 will be used for the search. The occurrence number represented by index-name is incremented by the same amount as, and at the same time as, the search index-name.

5. If the VARYING identifier-2 option appears and identifier-2 is an index data item, then this item is incremented by the same amount as, and at the same time as, the search index. If identifier-2 is not an index data item, then it is incremented by the value one (1) at the same time as the search index is in- cremented.

   A flowchart of a Format 1 type SEARCH operation containing two WHEN phrases follows:

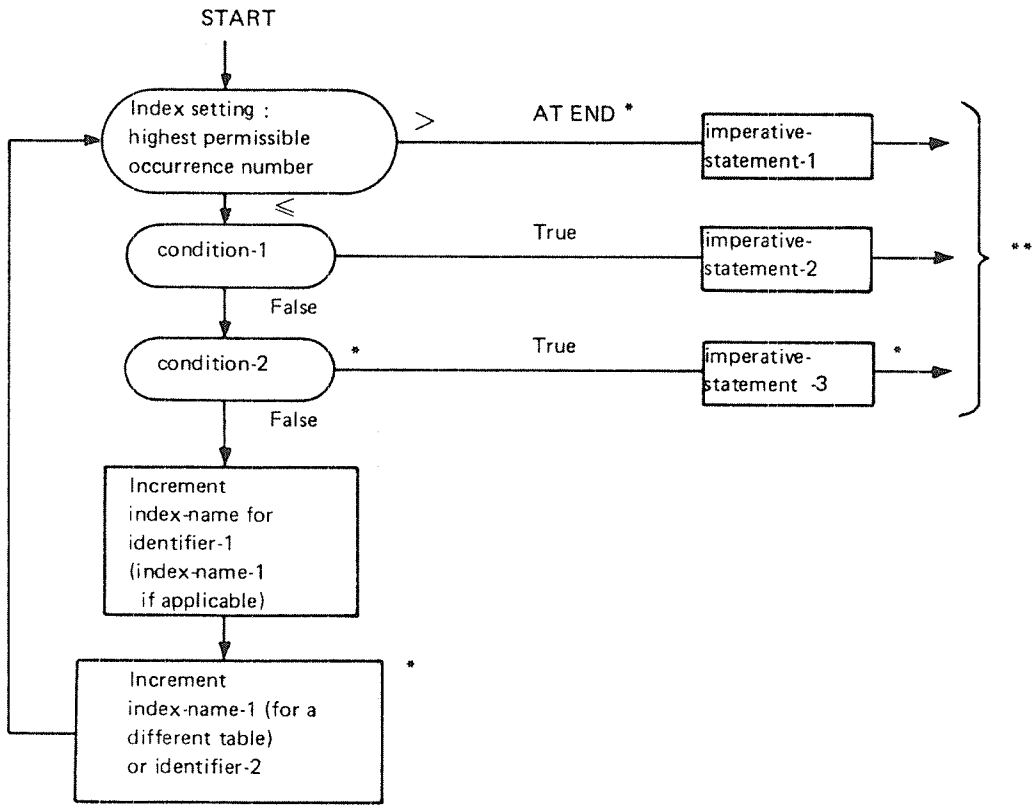


Figure 8.1:

* These operations are options included only when specified in the SEARCH statement.

** Each of these control transfers is to the next executable sentence unless the imperative-statement ends with a GO TO statement

Format 2

If the format 2 SEARCH ALL is used, a non-serial search operation may take place. It is a more simple type of search than for format 1, commencing at the beginning of the table.

The initial setting of the index-name for identifier-1 is ignored (i.e., need not be initialized with the SET statement).

The index is the same as that associated with the first index-name specified in the OCCURS clause.

The following rules apply:

1. If the WHEN option cannot be satisfied by any setting of the index within the permitted range then control is passed to imperative-statement-1 of the AT END phrase if present, or to the next executable sentence if this phrase is not present. In either case, the final setting of the index is not predictable.

2. If the WHEN option can be satisfied, control passes to imperative-statement-2 and the index will indicate an occurrence that allows the conditions to be satisfied.

8.3.1.1 Notes on Multi-Dimensional Tables

Identifier-1 can be a data item subordinate to a data item containing an OCCURS clause. That is, it can be part of a two or three-dimensional table. In this case the data description entry must specify an INDEXED BY option for each dimension.

To search an entire two or three-dimensional table it is necessary to execute a SEARCH statement several times, since this statement execution only modifies the setting of the index-name associated with identifier-1 (and, if present, index-name-1 or identifier-2). Prior to each execution, SET statements must be executed to reinitialize the associated index-names.

One format 3 PERFORM statement can search a whole multi-dimensional table.

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 09.08.46   DATE: 22.10.80
    SOURCE FILE: SEARCH-EX

 1        IDENTIFICATION DIVISION.
 2        PROGRAM-ID.
 3              SEARCH-EX.
 4       ***********************************************************************
 5       *      SHOWS USAGE OF A SIMPLE TABLE "LOOK-UP" VIA SEARCH VERB
 6       ***********************************************************************
 7        ENVIRONMENT DIVISION.
 8        DATA DIVISION.
 9        WORKING-STORAGE SECTION.
10        77  TABLE-LENGTH       COMP VALUE 16.
11        77  FIND-NAME          PIC X(20).
12       ***********************************************************************
13       * SET UP THE TABLE ELEMENTS , NORMALLY ONE WOULD READ DATA FROM
14       * A "REFERENCE" FILE AND PLACE INTO TABLE FOR PROCESSING
15       ***********************************************************************
16        01  NAMES-TABLE.
17              02 FILLER PIC X(20) VALUE "BRABANT            ".
18              02 FILLER PIC 9(5)  VALUE 310.
19              02 FILLER PIC X(20) VALUE "CISALPIN           ".
20              02 FILLER PIC 9(5)  VALUE 822.
21              02 FILLER PIC X(20) VALUE "ERASMUS            ".
22              02 FILLER PIC 9(5)  VALUE 481.
23              02 FILLER PIC X(20) VALUE "ETOILE DU NORD     ".
24              02 FILLER PIC 9(5)  VALUE 554.
25              02 FILLER PIC X(20) VALUE "GOTTARDO           ".
26              02 FILLER PIC 9(5)  VALUE 381.
27              02 FILLER PIC X(20) VALUE "ILE DE FRANCE      ".
28              02 FILLER PIC 9(5)  VALUE 544.
29              02 FILLER PIC X(20) VALUE "IRIS               ".
30              02 FILLER PIC 9(5)  VALUE 666.
31              02 FILLER PIC X(20) VALUE "LE CATALAN TALGO   ".
32              02 FILLER PIC 9(5)  VALUE 870.
33              02 FILLER PIC X(20) VALUE "LE CAPITOLE        ".
34              02 FILLER PIC 9(5)  VALUE 373.
35              02 FILLER PIC X(20) VALUE "LE MISTRAL         ".
36              02 FILLER PIC 9(5)  VALUE 683.
37              02 FILLER PIC X(20) VALUE "LEMANO             ".
38              02 FILLER PIC 9(5)  VALUE 595.
39              02 FILLER PIC X(20) VALUE "LIGURE             ".
40              02 FILLER PIC 9(5)  VALUE 322.
41              02 FILLER PIC X(20) VALUE "MEDIOLANUM         ".
42              02 FILLER PIC 9(5)  VALUE 889.
43              02 FILLER PIC X(20) VALUE "OISEAU-BLEU        ".
44              02 FILLER PIC 9(5)  VALUE 1039.
45              02 FILLER PIC X(20) VALUE "REMBRANT           ".
46              02 FILLER PIC 9(5)  VALUE 713.
47              02 FILLER PIC X(20) VALUE "RHEINGOLD          ".
48              02 FILLER PIC 9(5)  VALUE 1088.
49       ***********************************************************************
50       *  REDEFINE THE ELEMENTS FOR ACCESS WITH THE SEARCH VERB,
51       *  NOTE THAT THE DATA-NAME WITH THE *OCCURS* CLAUSE IS USED IN
52       *  SEARCH AND NOT THE REDEFINES DATA-NAME (WHICH MAY BE FILLER)
53       ***********************************************************************
54        01   FILLER      REDEFINES NAMES-TABLE.
55           02 TRAIN-TABLE  OCCURS  16 TIMES INDEXED BY TABINDEX.
56              03 NAME         PIC X(20).
57              03 DISTANCE     PIC 9(5).
58       /
59        PROCEDURE DIVISION.
60        A000.
61       *                       LIST OUT ALL THE TABLE ENTIES
62             PERFORM LIST-TABLE-ENTRY
63                     VARYING TABINDEX FROM 1 BY 1 UNTIL
64                     TABINDEX = TABLE-LENGTH.
```

```
65        A002.
66       *                        REQUEST A NAME TO FIND
67            DISPLAY "ENTER NAME TO FIND :".
68            ACCEPT FIND-NAME.
69       *                        START AT TOP OF TABLE (1)
70            SET TABINDEX TO 1.
71       *                        LOOK FOR REQUESTED NAME
72            SEARCH TRAIN-TABLE  AT END DISPLAY "NAME NOT FOUND",
73                     WHEN  FIND-NAME = NAME (TABINDEX),
74                           PERFORM LIST-TABLE-ENTRY.
75            GO TO A002.
76       ******************************************************************
77       *      NOTE THE WAY THAT THE LIST ROUTINE IS USED BY EITHER THE
78       *      PERFORM  OR THE SEARCH VERB.
79       ******************************************************************
80        LIST-TABLE-ENTRY.
81            DISPLAY "TRAIN : ", NAME (TABINDEX), " TRAVELS : ",
82                    DISTANCE (TABINDEX), "KM.".
```

*** NO ERROR MESSAGES ***

```
@NRL
RELOCATING LOADER LDR-1935F
*LOAD SEARCH-EX COBLIB
FREE: 037300-177777
*RUN
TRAIN : BRABANT              TRAVELS : 00313KM.
TRAIN : CISALPIN             TRAVELS : 00322KM.
TRAIN : ERASMUS              TRAVELS : 00481KM.
TRAIN : ETOILE DU NORD       TRAVELS : 00554KM.
TRAIN : GOTTARDO             TRAVELS : 00381KM.
TRAIN : ILE DE FRANCE        TRAVELS : 00544KM.
TRAIN : IRIS                 TRAVELS : 00666KM.
TRAIN : LE CATALAN TALGO     TRAVELS : 00870KM.
TRAIN : LE CAPITOLE          TRAVELS : 00373KM.
TRAIN : LE MISTRAL           TRAVELS : 00683KM.
TRAIN : LEMANO               TRAVELS : 00595KM.
TRAIN : LIGURE               TRAVELS : 00322KM.
TRAIN : MEDIOLANUM           TRAVELS : 00889KM.
TRAIN : OISEAU-BLEU          TRAVELS : 01039KM.
TRAIN : REMORANT             TRAVELS : 00713KM.
ENTER NAME TO FIND :IRIS
TRAIN : IRIS                 TRAVELS : 00666KM.
ENTER NAME TO FIND :
NAME NOT FOUND
ENTER NAME TO FIND :LEMANO
TRAIN : LEMANO               TRAVELS : 00595KM.
ENTER NAME TO FIND :
```

8.3.2 The Set Statement

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

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    }
```

Identifier-1 and identifier-3 must name either index data items, or elementary items described as an integer.

Identifier-4 must be described as an elementary numeric integer.

Integer-1 and integer-2 may be signed. Integer-1 must be positive.

Index-names are releated to a given table through the INDEXED BY option of the OCCURS clause which consitutes their definition.

Format 1 - To Option.

When this form of the SET statement is executed, the value of the sending field replaces the current value of the receiving field. If the *receiving field* specifies index-name-1, then, either:

a. If the *sending field* is an index data item then the value of this item is placed in the index name without change.

b. Otherwise, the receiving field is converted to a displacement value corresponding to the occurrence number indicated by the sending field.

If the receiving field specifies an index data item then this is set equal to the contents of the sending field (which must be an index-name or an index data item) no conversion takes place.

If the receiving field specifies an integer data item, then it is set to an occurrence number that corresponds to the occurrence number associated with the sending field (which must be an index name).

The above processes are repeated for indentifier-2, index-name-2, etc.

If index-name-3 is specified the value of the index before execution of the SET statement must correspond to an occurrence number of an element in the associated table.

Any subscripting or indexing associated with identifier-1, etc., is evaluated immediately before the value of the respective data item is changed.

Format 2- UP/DOWN BY Option

When this form of the SET option is executed, the value of the receiving field, index-name-4, is incremented (UP BY) or decremented (DOWN BY) a value corresponding to the value in the sending field. The process is repeated for index-name 5, etc.

Data is the following chart show the validity of the various operand combinations in the SET statement.

Sending Item	Receiving Item		
	Integer Data Item	Index-name	Index Data Item
Integer Literal	No	Valid	No
Integer Data Item	No	Valid	No
Index-name	Valid	Valid	Valid *
Index Data Item	No	Valid *	Valid *

* No conversion takes place.

9 INTER–PROGRAM COMMUNICATION

Complex data processing problems are frequently solved by the use of separately compiled but logically coordinated programs, which, at execution time, form logical and physical subdivisions of a single run unit. This approach lends itself to dividing a large problem into smaller, more manageable segments which can be programmed and debugged independently. At execute time, control is transferred from program to program by the used of CALL and EXIT PROGRAM statements.

9.1 BASIC CONCEPTS

In COBOL terminology, a program is either a source program or an object program depending on context; a *source program* is a syntactically correct set of COBOL statements; an *object program* is the set of instructions, constants, and other machine-oriented data resulting from the operation of a compiler on a source program; and a *run unit* is the total machine language necessary to solve a data processing problem. It includes one or more object programs as defined above, and it may include machine language from sources other than a COBOL compiler.

When the statement of a problem is subdivided into more than one program, the constituent programs must be able to communicate with each other. This communication may take two forms: transfer of control and reference to common data.

9.1.1 Transfer of Control

The CALL statement provides the means whereby control can be passed form one program to another within a run unit. A program that is activated by a CALL statement may itself contain CALL statements. However, results are unpredictable where circulaity of control is initiated; i.e., where program A calls program B, then program B calls program A or another program that calls program A.

When control is passed to a called program, execution proceeds in the normal way from procedure statement to procedure statement beginning with the first nondeclarative statement. If control reaches a STOP RUN statement, this signals the logical end of the run unit. If control reaches an EXIT PROGRAM statement, this signals the logical end of the called program only, and control then reverts to the point immediately following the CALL statement in the calling program. Stated briefly, the EXIT PROGRAM statement terminates only the program in which it occurs, and the STOP RUN statement terminates the entire run unit.

If the called program is not COBOL then the termination of the run unit or the return to the calling program must be programmed in accordance with the language of the called program.

9.1.2 Common Data

Because of program interaction, it may be necessary for one or more programs to have acess to the same data.

In a calling program, the common data items are described together with all other data items in the File, Working-storage, or Linkage Sections. In the called program, common data items are described in the Linkage Section.

At object time, memory is allocated for the whole Data Division in the calling program but not for the Linkage Section of the called program. Communication between the called program and the common data items stored in the calling program is through USING clauses contained in both programs. The USING clause in the calling program is contained in the CALL statement and the operands are common data items described in its Data Division. The USING clause in the called program has operands which are data items appearing in its Linkage Section.

The sequence of appearance of the identifiers in both lists of operands is significant. They must match in both programs. While the called program is being executed, every reference to an operand whose identifier appears in the called program's USING clause is treated as if it were a reference to the corresponding operand in the USING clause of the active CALL statement.

(A calling program may itself be a called program, in this case, common data items can be described in the calling program's Linkage Section. Storage will not be allocated for these items in the calling program itself but in the program which calls the calling program instead).

An example of a called and a calling program is outlined below:

```
CALLING PROGRAM (PROG-A)                CALLED PROGRAM (PROG-B)

WORKING-STORAGE SECTION.                  LINKAGE-SECTION.

01   A-LIST.                            01   B-LIST.
  02 HEADING      PIC X(10).              05 HEADING      PIC X(10).
  02 YEAR         PIC 9( 2).              05 DATE         PIC 9( 4).
  02 MONTH        PIC 9( 2).              05 CODE—ID      PIC X( 4).
  02 CODE-NO      PIC X( 4).                 .
     .                                       .
     .                                       .
     .                                       .
PROCEDURE DIVISION.                     PROCEDURE DIVISION USING B-LIST.
     .
     .
CALL PROG-B USING A-LIST.
```

Note that the names of the data items need not correspond and that parts of data items can be referred to separately (DATE in the called program is subdivided into YEAR and MONTH in the calling program).

9.1.3 Inter-program Communication - Data Division

In the Data Division of a called program, all file description entries must have assigned to them a value of an integral literal (using a VALUE OF FILE-ID IS clause) which is the *same* as that defined in the main program (see Section 5.3.1).

The programmer specifies in the Linkage Section those data items that are common with the calling program.
The format of the Linkage Section is :

LINKAGE SECTION.
level-number { data-name-1 / FILLER }

[REDEFINES Clause]
[BLANK WHEN ZERO Clause]
[JUSTIFIED Clause]
[OCCURS Clause]
[PICTURE Clause]
[SIGN Clause]
[SYNCHRONIZED Clause]
[USAGE Clause]
[IMPORT [COMMON] Clause]

[88 condition-name VALUE Clause]

The Linkage Section in a program is meaningful if and only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

The IMPORT clause must specify the same data item as in the corresponding EXPORT clause (see Section 5.4.2.13 for a description of the rules which apply to both clauses).

The IMPORT COMMON clause is used to specify a FORTRAN common block IMPORT.

The VALUE clause must not be specified in the Linkage Section except in condition-name entries (level 88).

The Linkage Section is used for describing data that is available through the calling program but is to be referred to in both the calling and the called program. No space is allocated in the program for data items referenced by data-names in the Linkage Section of that program. 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.

Data items defined in the Linkage Section of the called program may be referenced within the Procedure Division of the called program only if they are specified as operands of the USING phrase of the Procedure Division header or are subordinate to such operands, and the object program is under the control of a CALL statement that specifies a USING phrase.

The structure of the Linkage Section is the same as that previously described for the Working-Storage Section, beginning with a section header, followed by data description entries for noncontiguous data items and/or record description entries.

Each Linkage Section record name and noncontiguous item name must be unique within the called program since it cannot be qualified.

Of those items defined in the Linkage Section only data-name-1, data-name-2, ... in the USING phrase of the Procedure Division header, data items subordinate to these data-names, and condition-names and/or index-names associated with such data-names and/or subordinate data items, may be referenced in the Procedure Division.

9.1.3.1 Data Item Description Entries

Items in the Linkage Section that bear no hierarchic relationship to one another need not be grouped into records and are classified and defined as noncontiguous elementray items. Each of these data items is defined in a separate data description entry which begins with the special level-number 77.

The Following data clauses are required in each data description entry:

a. level-number 77
b. data-name
c. the PICTURE clause or the USAGE IS INDEX clause.

Other data description clauses are optional and can be used to complete the description of the item if necessary.

9.1.3.2 Record Description Entries

Data elements in the Linkage Section which bear a definite hierarchic relationship to one another must be grouped into 01-level records according to the rules for formation of record descriptions. Any clause which is used in an input or output record description can be used in a Linkage Section.

9.1.4 Inter - Program Communication — Procedure Division

In the Procedure Division, control is transferred between programs by means of the CALL statement.

Reference to common data is provided by the USING option which can appear in the CALL statement and in the called program's Procedure Division header.

The Procedure Division must begin with the following header:

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

The USING phrase is present if and only if the object program is to function under the control of a CALL statement, and the CALL statement in the calling program contains a USING phrase.

For a description of the data-names see the details of the USING option in the CALL statement. The USING option is common to several Inter-Program Communication elements.

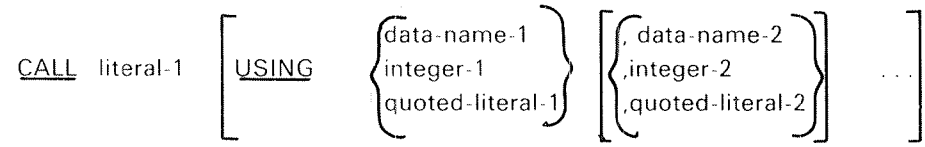
Each of the operands in the USING phrase of the Procedure Divison header must be defined as a data item in the Linkage Section of the program in which this header occurs, and it must have 01 or 77 level-number.

Within a called program, Linkage Section data items are processed according to their data descriptions given in the called program.

9.1.4.1 The Call Statement

The CALL statement causes control to be transferred from one object program to another within the run unit.

Format



Literal-1 must be a non-numeric literal and conform to the rules for formation of a program name (see PROGRAM-ID paragraph in the Identification Division chapter).

Called programs may contain CALL statements.

CALL statement execution causes control to pass to the called subprogram. The first time a called program is entered its state is that of an original 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.

Reinitialization of GO TO statements that have been altered etc., are the responsibility of the programmer.

9.1.4.2 The USING Option

This option makes data items in a calling program available to the called program.

The USING option is specified if, and only if, the called subprogram is to operate under control of a CALL statement and that CALL statement itself contains a USING option. That is, for each CALL USING statement in a calling program there must be a corresponding USING option specified in a called subprogram.

The data-name, or integers, specified by the USING option indicate the data items available to a calling program that may also be referred to in the called program. The order of appearance of these data-names is critical. Corresponding data-names refer to a single set of data equally available to both programs. Their description must define an equal number of character positions but their correspondence is positional and not by name. (In the case of index names no such correspondence is established, separate indices are referred to in the called and calling programs).

9.1.4.3 The EXIT PROGRAM Statement

The EXIT PROGRAM statement marks the logical end of a called program.

Format

EXIT PROGRAM.

The EXIT PROGRAM statement must appear in a sentence by itself.

The EXIT PROGRAM sentence must be the only sentence in the paragraph.

General Rule:

1. An execution of an EXIT PROGRAM statement in a called program causes control to be passed to the calling program. Execution of an EXIT PROGRAM statement in a program which is not called behaves as if the statement were an EXIT statement (see under Procedure Branching Statements in the Procedure Division description).

```
NORD-10/100 COBOL COMPILER - VER  01.10.80      TIME: 09.11.35   DATE: 22.10.80
    SOURCE FILE: (TD)GENSEQ

      1        IDENTIFICATION DIVISION.
      2        PROGRAM-ID.
      3              GENSEQ.
      4       ************************************************************************
      5       *      CREATES SQ-FILE AND LISTS.
      6       ************************************************************************
      7        ENVIRONMENT DIVISION.
      8        INPUT-OUTPUT SECTION.
      9        FILE-CONTROL.
     10              SELECT SQ-FILE ASSIGN "COB1:DATA" ,
     11                           ORGANIZATION IS SEQUENTIAL,
     12                           ACCESS IS SEQUENTIAL.
     13        DATA DIVISION.
     14        FILE SECTION.
     15        FD    SQ-FILE
     16                LABEL RECORDS STANDARD
     17                DATA RECORDS M-REC.
     18        01  M-REC.
     19              02  FILLER PIC X(10).
     20              02  SEQNUM PIC 9(5).
     21              02  FILLER PIC X(5).
     22              02  FILLER PIC X(40).
     23         WORKING-STORAGE SECTION.
     24        01    RANDNO          COMP,  VALUE ZERO.
     25        01    MAXRAND         COMP,  VALUE 1000.
     26        01    NORECS          PIC 9(4).
     27        01    RECCNT          COMP, VALUE 0.
     28
     29        PROCEDURE DIVISION.
     30        INIT-01.
     31              OPEN OUTPUT SQ-FILE.
     32              DISPLAY 'CREATE  RECORDS ?
     33              PERFORM GET-NORECS.
     34
     35              PERFORM CRE-SQ-FILE NORECS TIMES.
     36       *                                  BUILD THE INPUT FILE
     37              CLOSE SQ-FILE.
     38              DISPLAY 'FILE SQ-FILE CREATED.', RECCNT, 'RECORDS.'.
     39              OPEN INPUT SQ-FILE.
     40        LIST-FILE-0.
     41              MOVE 0 TO RECCNT.
     42        LIST-FILE-1.
     43              READ SQ-FILE AT END  GO TO LIST-END.
     44              ADD 1 TO RECCNT.
     45              DISPLAY 'REC ', RECCNT, ' SEQNUM = ', SEQNUM.
     46              GO TO LIST-FILE-1.
     47        LIST-END.
     48              CLOSE SQ-FILE.
     49              DISPLAY "JOB FINISH".
     50              STOP RUN.
     51
     52        CRE-SQ-FILE.
     53              CALL 'RND' USING RANDNO, MAXRAND.
     54              MOVE ALL '*' TO M-REC.
     55              MOVE RANDNO TO SEQNUM.
     56              ADD 1 TO RECCNT.
     57              DISPLAY "UT REC =", RECCNT, " KEY =", SEQNUM.
     58              WRITE M-REC.
     59
     60        GET-NORECS.
     61              ACCEPT NORECS.
     62              IF NORECS NOT NUMERIC,
     63                           DISPLAY "** NOT NUMERIC DATA ",
     64                           GO TO GET-NORECS.

*** NO ERROR MESSAGES ***
```

```
                    )9ASSM XRND,DIABLO,0
                    % RND := A FORTRAN/COBOL CALLABLE RANDOM NUMBER GENERATOR
                    %
                    %       CALL RND(IX,IMAX)
                    %
                    % WHERE IX   = RECEIVING INTEGER VARIABLE
                    %       IMAX = MAX RANGE INTEGER
                    %   NOTE  IF IMAX > 0 THEN RETURN A RANDOM VALUE IN IX
                    %         IF IMAX = 0 THEN RETURN THE SEED VALUE IN IX
                    %         IF IMAX < 0 THEN SET THE SEED VALUE FROM  IX
                    %
                    %
                            )9BEG
                            )9ENT RND
                            )9LIB RND
00001               144053  RND,    SWAP SA DB      %SAVE B REG
00002 004     00350 004033          STA   SAVB
00003 044  IB   001 045401          LDA I 1,B       %GET IMAX
00004 130     00007 131003          JAZ   RNDZ      %XMAX = 0 RETURN SEED VALUE
00005 130     00012 130405          JAN   RNDN      %XMAX < 0 SET NEW SEED VALUE
00006 124     00015 124007          JMP   RNDP      %XMAX > 0 RETURN NEXT RND NR
00007 044     00034 044025  RNDZ,   LDA   SEED      %LOAD SEED VALUE AND RETURN
00010 004  IB   000 005400          STA I 0,B
00011 124     00031 124020          JMP   RNDX
00012 044  IB   000 045400  RNDN,   LDA I 0,B       %LOAD NEW SEED VALUE, SAVE
00013 004     00034 004021          STA   SEED
00014 124     00031 124015          JMP   RNDX      %RETURN
00015               000000  RNDP,
00016 150           151420          NLZ   20        %CONVERT TO FLOATING
00017 030     00040 030021          STF   XMAX
00020 044     00034 044014          LDA   SEED      %INITIAL VALUE
00021 120     00036 120015          MPY   CONS
00022 060     00037 060015          ADD   CONS+1
00023 004     00034 004011          STA   SEED
00024               156577          SHA   ZIN SHR 1
00025 150           151401          NLZ   1
00026 110     00040 110012          FMU   XMAX
00027 150           152360          DNZ   -20       %CONVERT BACK TO INTEGER
00030 004  IB   000 005400          STA 1 0,B       %STORE IN IX
00031 044     00035 044004  RNDX,   LDA   SAVB      %RECOVER B REG
00032               146153          COPY SA DB
00033               146142          EXIT
00034               003614  SEED,   3614
00035               000000  SAVB,   0
00036               012465  CONS,   12465;
00037               033031          33031
00040               000000  XMAX,   0;
00041               000000          0;
00042               000000          0
                            )FILL
                            )9END
                            )9EOF
```

10 DEBUGGING

10.1 THE SYMBOLIC DEBUGGER

With this facility the user is provided with a powerful set of commands to monitor the executing of his COBOL program. He may inspect and change the state of the computation.

Compilation:

To invoke the Symbolic Debugger the compile-mode command DEBUG must be given.

The Symbolic Debugger is a separate system and after recover the Symbolic Debugger (DEBUG) the command PLACE <program> must be given.

On the ND-100, if the DEBUG command has been issued when compiling a program, the program and COBOL library must be loaded on an image file and dumped on a :PROG file by means of the DUMP command, or loaded into a :PROG file by means of the PROG—FILE command.

```
@COBOL

NORD-10/100 COBOL COMPILER - VER 81.01.09

*DEBUG
*COMPILE XXX,0,XXX

*** NO ERROR MESSAGES ***

*EXIT
@NRL
RELOCATING LOADER LDR-1935G
*PROG-FILE XXX
*L XXX
FREE: 000474-177777
*L COBOL-1 BANK
FREE: 045161-177777
*EX
@DEBUG

NORD-100 SYMBOLIC DEBUGGER, 24 NOVEMBER 1980.
*PLAC XXX
COBOL PROGRAM.  XXX
*RUN
```

Figure 10.1:Example of Compilation and Debugging on the ND-100

10.1.1 Commands and Command Arguments

Whenever the symbolic debugger expects the operator to enter a command it outputs an asterisk (*). A command (along with possible arguments) must be typed on the same line as the asterisk. Omitted arguments are asked for by the debugger. Several commands, separated by semicolons (;), can be written on the same line. Command names may be abbreviated and the standard editing characters are available when typing command input.

10.1.1.1 Command Arguments

Several types of command arguments are used in the debugger:

- Decimal and octal numbers.
- Character constants.
- Program, line, routine, paragraph, or variable item specifier.
- Expressions involving the above types and the operators +, —, *, /, and ADDR. Array indexing is also available.
- Format specifier.
- File name.

10.1.1.2 Decimal and Octal Numbers

A decimal constant is denoted by a sequence of digits optionally followed by the letter D. An octal constant is denoted by a sequence of octal digits (0-7) followed by the letter B. Numeric constants can be signed.

10.1.1.3 Character Constants

A character constant is denoted by a number sign (#) followed by a ASCII character. For example:

A has the value 65(101B).

10.1.1.4 Program, Line, Routine, Paragraph and Variable Specifier

A named item is specified by a sequence of names separated by dots (.), corresponding to the static program nesting in a COBOL program.

10.1.1.5 Expressions

Arithmetic expressions can be formed using array indexing, dot notation, and the operators +, —, *, and / on numeric items. The operator ADDR can be used on any item that has an address. The operators must be separated from the variable by a space.

10.1.1.6 Format Specifier

A format specifier is one or more of the letters: O = Octal, D = Decimal, H = Hexadecimal, A = ASCII, S = Floating point (single), F = Floating point (double), and I = Instruction (disassembly).

10.1.1.7 File Name

No syntax checking of file names is performed. A file name is terminated by a carriage return, space, comma or semicolon. If the file is already open, the octal file number can be used in place of the file name (octal number *without* B).

10.1.2 The Available Commands

ACTIVE—ROUTINES

This command writes the current routine call hierarchy on the terminal, starting with the current routine and ending with the main program.

ALIGN—LISTING [{ program / routine }] line

This command is used to adjust the line numbers in the debugger to correspond with those on a listing that is not up-to-date. Several ALIGN—LISTING commands may be given in order to adjust different parts of the listing. If areas overlap the command given last has priority over previous commands.

If no program/routine is specified the innermost routine in the current scape is assumed.

BREAK { routine / paragraph / section / line } [count]

Sets a breakpoint at the specified item. If a routine-name is specified the breakpoint is set at the first line in the routine. If a positive number is specified for the count argument the breakpoint is set at the first line in the routine. If a positive number is specified for the count argument the breakpoint will be passed count-1 times before it is performed.

When the breakpoint is reached, execution terminates and control passes to the debugger. The current scope is updated and a message indicating the current routine and line is output.

BREAK—ADDRESS program address [count]

This command is similar to the BREAK command except that the breakpoint is specified directly as a program address.

BREAK—RETURN

Sets a breakpoint at the return address of the current routine and resumes execution from the current line.

CHECK—OUT—MODE [{ program / routine }]

This command sets breakpoints on all lines in the specified interval. Whenever control returns to the debugger a list of all the lines that have never been executed can be obtained by using the DUMP—LOG command. If the command LOG—CALLS is given before the CHECK—OUT—MODE command only the first line in every routine is checked.
If no program/routine is specified, all lines are checked.

COMPARE—DATA low high [output file]

The data area specified by the lower and upper bounds is compared to the image-file contents. Modified locations are displayed with address, old contents, and new contents.
Default output file is the terminal, default file type is : LIST.

DISPLAY [{ item / value }]

The expression is evaluated and displayed in the formats specified by the FORMAT—DISPLAY command. Several expressions, separated by commas, can be specified on the same line.

DUMP—LOG [*output file*]

The format of this command depends upoin the type of log specified.

If LOG—CALLS was specified last, a list of the last 200 routine calls is displayed on the terminal.

If LOG—LINES was specified last, a list of the last 200 lines that have been executed is displayed on the terminal. If a line is the first line in a routine, the routine name is also displayed.

Default output file is the terminal, default file type is : LIST.

EXIT

Close files and transfer control to monitor.

FIND—SCOPE *program address*

This command finds the program/routine and line number that corresponds to the specified program-address and updates the scope accordingly. The current scope status is displayed.

FORMAT—DISPLAY { *A* *D* *F* *H* *O* *S* }

Set format(s) for the DISPLAY command.

FORMAT—LOOK—AT { *A* *D* *F* *H* *I* *O* *S* }

Set format(s) for the LOOK—AT commands.

GUARD *address* [*low*] : [*high*]

This comnmand specifies a location which is to be checked every time a breakpoint is reached. If the contents of the location are outside the permitted range, a guard violation occurs and control is passed to the debugger. The permitted range is specified by low : high. If low < = high then the permitted range is low < =n < = high. If low> high then the permitted range is the complement of the above.

If only low is specified the high is set equal to low. If no range is specified the permitted range becomes the signal value of the current contents of the specified address.

The frequency with which the location is checked is determinded by using the LOG—CALLS or LOG—LINES command.

If no arguments are given (address is null), the guard violation check is disabled (guard reset).

HELP *command name*

The HELP command lists available commands on the terminal. Only those commands that have <command name> as a subset are listed. If <command name> is null then all available commands are listed. Each command is followed by an argument list (if any). Required and optional arguments are enclosed in angle and square brackets, respectively.

INVOKE *routine* [*parameter* [, *parameter*] ...]

This command can be used to call routines in the user program (the program being debugged). Only variable items that have an address can be used as parameters, i.e., constants are not allowed.

LOG—CALLS [{ *program* / *routine* }]

This command specifies that all routine calls are to be logged in a cyclic buffer. This buffer can be inspected by means of the DUMP-LOG command (see above). The buffer can hold a maximum of ten (10) entries.

If a program/routine is specified only routines in the specified unit are logged.

This command also affects the CHECK—OUT—MODE, GUARD, and STEP commands (break on every routine as opposed to every line).

LOG—LINES [{ program / routine }]

This command specifies that all lines that are executed are to be logged in a cyclic buffer. This buffer can be inspected by means of the DUMP—LOG command (see above). The buffer can hold a maximum of ten (10) entries.

If a program/routine is specified only lines in the specified unit are logged.

This command also affects the CHECK—OUT—MODE, GUARD, and STEP commands (break on every line as opposed to every routine).

LOOK—AT—DATA *data address* [*count*] [output-file]

This command and the related commands LOOK—AT PROGRAM, LOOK—AT—REGISTER, and LOOK—AT—STACK enables the user to inspect and modify data locations, program locations and registers. When first entered, a number of locations starting from the specified address is printed on the terminal. The optional parameter, count, specifies the number of locations to be output. The default value of count is one.

If an output file other than the terminal is specified, control returns to the debugger when the specified number of locations have been output. If no output file, or the terminal is specified control remains within the LOOK—AT command and the sub-commands described below are available.

The contents of each of the locations is printed in the format(s) specified by the FORMAT—LOOK—AT command.

Several sub-commands are available inside the LOOK—AT commands (cr means carriage return):

m,n/ cr	This is the same as LOOK—AT m, n but without having to return to the debugger's command processor in between.
m cr	Deposits the value of the expression m in the current location and advances to the next location.
cr	Advances to the next location without changing the contents of the current location.
;cr or . cr or @ cr	Return to the debugger's processor.

Special notation used with the slash (/) command:

m/ cr	Take value of m as next address and display this location.
/ cr	Take contents of current location as next address and display this location (direction).
m,/ cr	Take value of m as next address and display n locations, where n is the last count entered.
,n/ cr	Take the contents of the current location as the next address and display n locations.
,/ cr	Take the contents of the current location as the next address and display n locations, where n is the last count entered.

LOOK—AT—PROGRAM program address [*count*] [*output file*]

Inspect and modify program locations. This command is similar to the LOOK—AT—DATA command except that I format (symbolic instructions) is enabled as default.

LOOK—AT—REGISTER register [*count*] [*output file*]

Inspect and modify CPU—registers. This command is similar to the LOOK—AT—DATA command. To dump all registers type: LOOK—AT—REGISTER P 9.

LOOK—AT—STACK B-register [*count*] [*output file*]

Inspect and modify locations in the stack. This command is similar to the LOOK—AT—DATA command except that both absolute and relative addresses are displayed. Locations in the stack header are given by name rather than by address. The sub-commands NEXT and PREVIOUS can be used to move up and down the stack according to the current stack links. These subcommands may be abbreviated just as normal commands.

MACRO *name* *body*

This command can be used to build macro commands composed of one or more of the basic commands and other macro commands. The macro name can be any character-string and is terminated by space or comma. Only the first eight characters are significant. The rest of the line following the macro name is taken as the macro body. The macro body is not terminated by semicolon thus several commands can be included in the same macro body.

If the body is empty, the corresponding macro is erased. If the macro name is empty, all the currently defined macros are displayed on the terminal.

A macro parameter is referenced in the macro body as "n", where n is a one-digit number (1-9).

A macro name is used in the same way as a command name and is taken into account in the abbreviation lookup. However, macro parameters are not asked for if missing but are simply taken to be empty strings when the macro is expanded.

RESET—BREAKS [*program address*]

If no program address is specified, all breakpoints set with BREAK, LOG—CALLS, or LOG—LINES are reset. A breakpoint set by means of the BREAK—ADDRESS command is reset only if it is the first instruction of a line.

If a program address is specified the breakpoint in the addressed location is reset.

RUN [*program-address*]

If no program-address is specified execution is resumed from the current line. Otherwise control is transferred directly to the specified program address.

SCOPE [{ *program* / *routine* }]

This command finds the specified program routine and updates the scope accordingly. The current scope status is displayed. If no program/routine is specified the current scope is not affected.

SET *variable* [=] *value*

This command is used to set program variables. Any variable reference which has a defined address can be set.

Example:

```
*SET XX  10

*
```

STEP [*count*]

Sets the debugger in single step mode. One line or one routine call is executed at each step depending upon the log mode (LOG—CALLS, LOG—LINES). The count argument specifies the number of units (lines or routine calls) to be executed at each step. When the debugger stops and outputs the current routine and line, the user can type carriage return (execute n units), or something else (command). The current scope is positioned correctly at each step.

If the count is specified to 0 (zero), then each line/routine is printed out as the program execution is continued (execution trace).

If the debugger is unable to position the scope (P-register is outside the area described by the available debug information), the step unit will be one instruction.

10.1.3 Symbolic Debugger Command Summary (Help Output)

ACTIVE—ROUTINES	
ALIGN—LISTING	[PROGRAM OR ROUTINE] LINE
BREAK	ROUTINE, PARAGRAPH, SECTION OR LINE [COUNT]
BREAK—ADDRESS	PROGRAM ADDRESS [COUNT]
BREAK—RETURN	
CHECK—OUT—MODE	[PROGRAM OR ROUTINE]
COMPARE—DATA	LOW HIGH [OUTPUT FILE]
COMPARE—PROGRAM	LOW HIGH [OUTPUT FILE]
DISPLAY	[ITEM OR VALUE]
DUMP—LOG	[OUTPUT FILE]
EXIT	
FIND—SCOPE	PROGRAM ADDRESS
FORMAT—DISPLAY	FORMATS A, D, F, H, O, OR S
FORMAT—LOOK—AT	FORMATS A, D, F, H, I, O, OR S
GUARD	ADDRESS [LOW] : [HIGH]
HELP	COMMAND NAME
INVOKE	ROUTINE [PARAMETER (S)]
LOG—CALLS	[PROGRAM OR ROUTINE]
LOG—LINES	[PROGRAM OR ROUTINE]
LOOK—AT—DATA	DATA ADDRESS [COUNT] [OUTPUT FILE]
LOOK—AT—PROGRAM	PROGRAM ADDRESS [COUNT] [OUTPUT FILE]
LOOK—AT—REGISTER	REGISTER [COUNT] [OUTPUT FILE]
LOOK—AT—STACK	B REGISTER [COUNT][OUTPUT FILE]
MACRO	NAME BODY
RESET—BREAKS	[PROGRAM ADDRESS]
RUN	[PROGRAM ADDRESS]
SCOPE	[PROGRAM OR ROUTINE]
SET	VARIABLE [=] VALUE
STEP	[COUNT]

For further information on the use of the Symbolic Debugger, refer to the Symbolic Debugger Users Guide, ND-60.158.

APPENDIX A

COMPOSITE LANGUAGE SKELETON

This appendix contains the complete syntax of the ND COBOL. It is intended to display complete and syntactically correct formats used throughout this manual.

A.1 Notation Used in Formats

A.1.1 Definition of a General Format

A general format is the specific arrangement of the elements of a clause or a statement. (1) A clause or a statement consists of elements as defined below. Throughout this manual, a format is shown adjacent to information defining the clause or statement. When more than one specific arrangement is permitted, the General Format is separated into numbered formats. Clauses must be written in the sequence given in the General Format. (Clauses that are optional must, if they are used, appear in the sequence shown.) In certain cases, stated explicitly in the rules associated with a given format, clauses may appear in sequences other than shown. Applications, requirements or restrictions are shown as rules.

A.1.1.1 Elements

Elements which make up a clause or a statement consist of upper-case words, lower-case words, level-numbers, brackets, braces, connectives, and special characters.

(1) These definitions are identical to those of the CODASYL COBOL committee.

A.1.1.2 Words

All underlined upper-case words are called key words and are required when the functions of which they are a part are used. Upper-case words which are not underlined are optional to the user and need not be written in the source program. Upper-case words, whether underlined or not, must be spelled correctly.

Lower-case words, in a General Format, are generic terms used to represent COBOL words, literals, PICTURE character-strings, or a complete syntactical entry that must be supplied by the user. Where generic terms are repeated in a General Format, a number or letter appendage to the term serves to identify that term for explanation or discussion.

A.1.1.3 Level-Numbers

When specific level-numbers appear in Data Description entry formats, those specific level-numbers are required when such entries are used in a COBOL program. In this document, the form 01, 02...09 is used to indicate level-numbers 1 through 9.

A.1.1.4 Brackets, Braces and Choice Indicators

When brackets, [ ], enclose a portion of a General Format, one of the options contained within the brackets may be explicitly specified or that portion of the General Format may be omitted.

When braces, { }, enclose a portion of a General Format, one of the options contained within the braces must either be explicitly specified or implicitly selected. If one and only one of the options contains only reserved words which are not key words, that option is the default option and is implicitly selected unless one of the options is explicitly specified.

When choice indicators, {[ ]}, enclose a portion of the General Format, one or more of the unique options contained within the choice indicators must be specified, but a single option may be specified only once.

Options are indicated in a General Format or a portion of a General Format by vertically stacking alternative possibilities, by a series of brackets, braces or choice indicators or by a combination of both. An option is selected by specifying one of the possibilities, from a stack of alternative possibilities, or by specifying a unique combination of possibilities from a series of brackets, braces or choice indicators.

A.1.1.5 The Ellipsis

In text, other than the General Formats, the ellipsis shows omission of a word or words when such omission does not impair comprehension. This is the conventional meaning of the ellipsis, and this use becomes apparent in context.

In a General Format, the ellipsis represents the position at which the user elects repetition of a portion of a format. The portion of the format that may be repeated is determined as follows:

Given '...' (the ellipsis) in a format, scanning right to left, determine the ']' (rigth bracket) or '}' (right brace) delimiter immediately to the left ot the '...'; continue scanning right to left and determine the logically matching ' [' (left bracket) or '{' (left brace) delimiter; the '...' applies to the portion of the format between the determined pair of delimiters.

A.1.1.6 Format Punctuation

The separator period, when used in formats, has the status of a required word.

A.1.1.7 Use of Special Characters in Formats

Special characters, when appearing in formats, although not underlined, are required when such portions of the formats are used.

GENERAL FORMAT FOR IDENTIFICATION DIVISION

```
IDENTIFICATION DIVISION

PROGRAM-ID.  program-name.

[AUTHOR.  [comment-entry] ...]

[INSTALLATION.  [comment-entry] ...]

[DATE-WRITTEN.  [comment-entry] ...]

[DATE-COMPILED.  [comment-entry] ...]

[SECURITY.  [comment-entry] ...]

[REMARKS.  [comment-entry] ...]
```

GENERAL FORMAT FOR ENVIRONMENT DIVISION

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER.    computer-name  [WITH DEBUGGING MODE] .

OBJECT-COMPUTER.    computer-name
        [, SEGMENT-LIMIT IS  segment-number] .

[SPECIAL-NAMES.

        [, CURRENCY SIGN IS  literal]
        [, DECIMAL-POINT IS COMMA] ] .

[
INPUT-OUTPUT SECTION.

  FILE-CONTROL.
        {file-control-entry} ...

  I-O-CONTROL.
        [SAME AREA FOR  file-name-1 {, file-name-2} ...] ...]
```

GENERAL FORMAT FOR FILE CONTROL ENTRY

FORMAT 1:

```
SELECT  [OPTIONAL]  file-name
        ASSIGN TO  assignment-name-1
        [
        ; RESERVE  integer-1 [AREA ]
                             [AREAS]
        ]
        [; ORGANIZATION IS SEQUENTIAL]
        [; ACCESS MODE IS SEQUENTIAL]
        [; FILE STATUS IS  data-name-1] .
```

FORMAT 2:

```
SELECT  file-name
        ASSIGN TO  assignment-name-1
        [
        ; RESERVE  integer-1 [AREA ]
                             [AREAS]
        ]
        ; ORGANIZATION IS INDEXED
        [
        ; ACCESS MODE IS  {SEQUENTIAL}
                         {RANDOM    }
                         {DYNAMIC   }
        ]
        ; RECORD KEY IS  data-name-1
        [; ALTERNATE RECORD KEY IS  data-name-2
                    [WITH DUPLICATES] ] ...
        [; FILE STATUS IS  data-name-3] .
```

GENERAL FORMAT FOR FILE CONTROL ENTRY

FORMAT 3:

```
SELECT file-name

       ASSIGN TO assignment-name-1

       [                        [AREA ] ]
       [ ; RESERVE integer-1    [AREAS] ]

       ; ORGANIZATION IS RELATIVE

       [ ; ACCESS MODE IS
       [
       [      { SEQUENTIAL [, RELATIVE KEY IS data-name-1]  } ]
       [      { {RANDOM }                                   } ]
       [      { {DYNAMIC}, RELATIVE KEY IS data-name-1      } ]

       [; FILE STATUS IS data-name-2] .
```

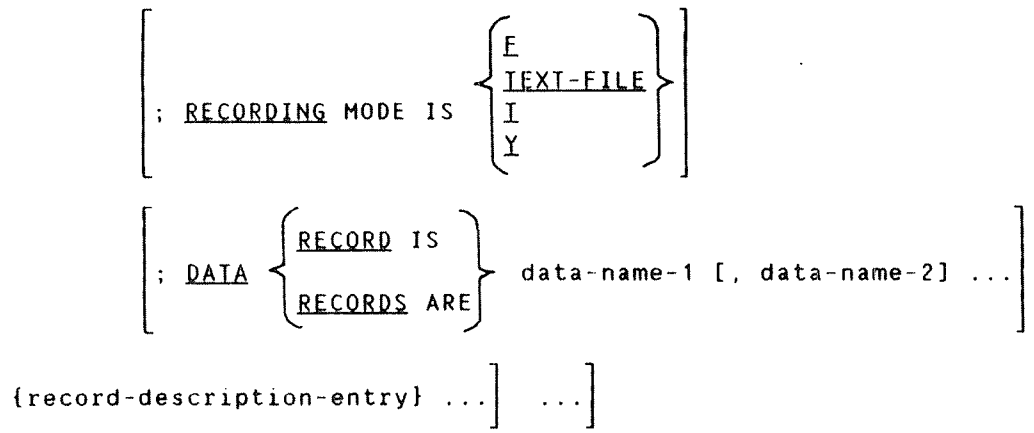
FORMAT 4:

```
SELECT file-name ASSIGN TO assignment-name-1 .
```

GENERAL FORMAT FOR DATA DIVISION

```
DATA DIVISION.
[FILE SECTION.

[FD  file-name

        [; BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS    }]
                                                   {CHARACTERS}

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

                  [DEPENDING ON  identifier] ]

        [; LABEL {RECORD IS  } {STANDARD}]
                 {RECORDS ARE} {OMITTED }

        [VALUE OF FILE-ID IS integer]

        [; RECORDING MODE IS {F          }]
                             {TEXT-FILE  }
                             {I          }
                             {Y          }

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

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

[SD file- name

        [; RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS

                  [DEPENDING ON  identifier] ]
```

GENERAL FORMAT FOR DATA DIVISION

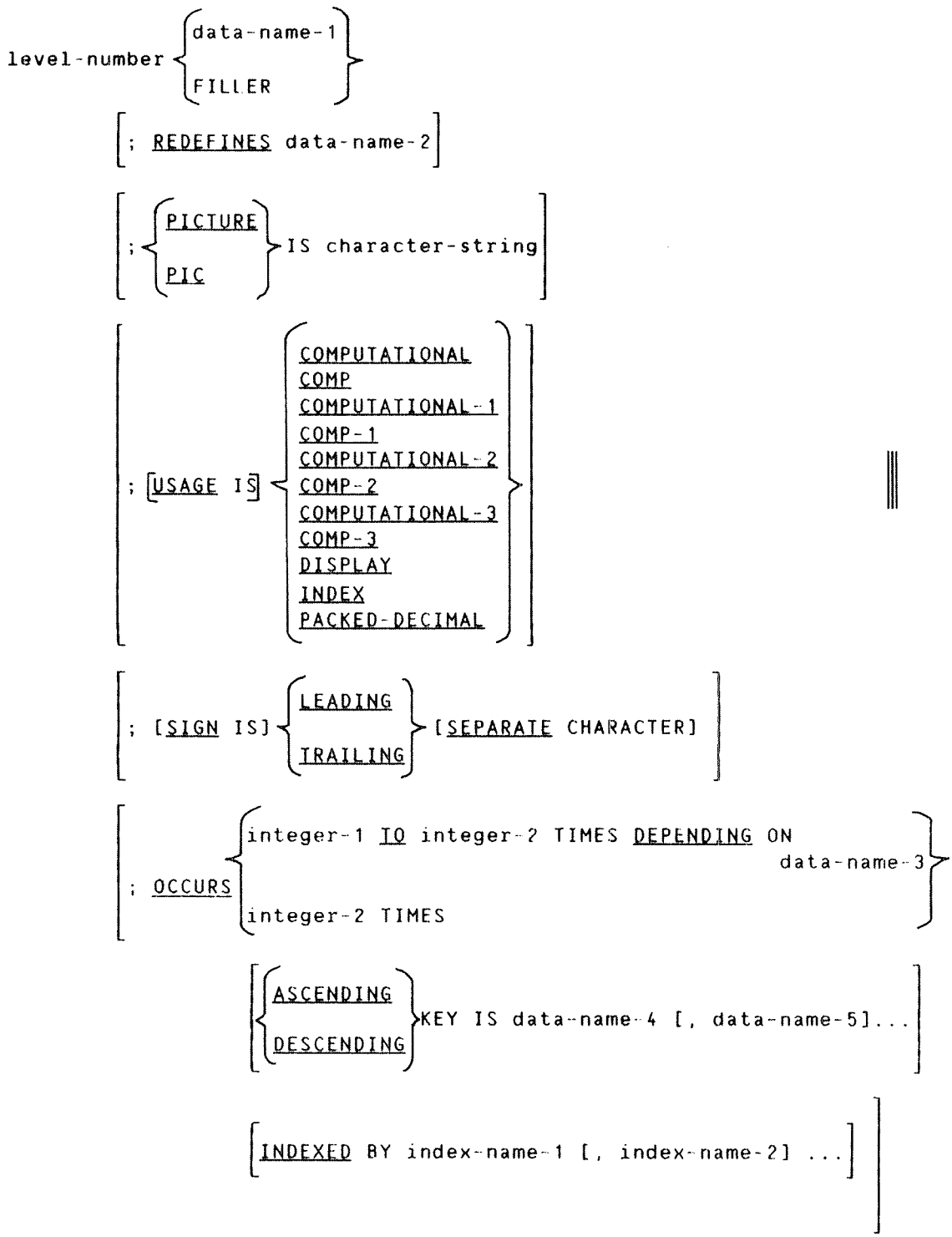


GENERAL FORMAT FOR DATA DIVISION

```
[WORKING-STORAGE SECTION.

 [77-level-description-entry ]
 [record-description-entry   ] ...]

[LINKAGE SECTION.

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

GENERAL FORMAT FOR DATA DESCRIPTION ENTRY

FORMAT 1



GENERAL FORMAT FOR DATA DESCRIPTION ENTRY

```
[; {SYNCHRONIZED} [LEFT ]]
   {SYNC        } [RIGHT]

[; {JUSTIFIED} RIGHT]
   {JUST     }

[; BLANK WHEN ZERO]

[; VALUE IS literal]

[; IMPORT [COMMON]]

[; EXPORT] .
```

FORMAT 2:

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

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

GENERAL FORMAT FOR PROCEDURE DIVISION

FORMAT 1:

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

[DECLARATIVES.

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

GENERAL FORMAT FOR VERBS

```
ACCEPT identifier [FROM  mnemonic-name]

ACCEPT identifier FROM {DATE     }
                       {DAY      }
                       {TIME     }
                       {CPU-TIME }

ACCEPT ( {[identifier [{+} integer]]} {[identifier [{+} integer]]} )
       (  {        integer         }  {        integer         }  )
         identifier [WITH [BEEP]
                          [SPACE-FILL]
                          [LENGTH-CHECK]
                          [AUTO-SKIP]
                          [PROMPT]
                          [UPDATE]
                          [INVISIBLE]
                          [BLANK-WHEN-ZERO]
                          [MUST]
                          [INVERSE-VIDEO]
                          [BLINK]
                          [UNDERLINE]
                          [LOW-INTENSITY]
                          [NORMAL]
                          [UPPER-CASE]
                          [UP procedure-name]
                          [DOWN procedure-name]
                          [HOME procedure-name]
                          [EXIT procedure-name]
                          [LEFT procedure-name]
                          [RIGHT procedure-name]
                          [CONTROL procedure-name] ]

ACCEPT-ERROR
```

GENERAL FORMAT FOR VERBS

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

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

        [ ; ON SIZE ERROR  imperative-statement ]
```

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

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

        [ ; ON SIZE ERROR  imperative-statement ]
```

```
ALTER  procedure-name-1 TO [PROCEED TO] procedure-name-2

       [ , procedure-name-3 TO [PROCEED TO] procedure-name-4 ] ...
```

```
BLANK SCREEN
```

```
BLANK [ LINE  ] n1 [TO n2] [COLUMN n3 TO n4]
      [ LINES ]
```

```
CALL literal-1 [ USING { data-name-1 } [ , data-name-2 ]
                      { quoted-literal } [ , quoted-literal ] ...
                      { integer      } [ , integer      ]     ]
```

GENERAL FORMAT FOR VERBS

```
CLOSE file-name-1 [WITH LOCK] [, file-name-2 [WITH LOCK]] ...

CLOSE file-name-3 [ [ [REEL] [WITH NO REWIND] ]
                  [ [UNIT]                   ]
                  [ WITH { NO REWIND } ]
                  [      { LOCK      } ]
                  [ , file-name-4 [ [ [REEL] [WITH NO REWIND] ] ] ... ]
                  [               [ [UNIT]                   ] ]
                  [               [ WITH { NO REWIND } ]     ]
                  [               [      { LOCK      } ]     ]

COMPUTE identifier-1 [ROUNDED] [, identifier-2 [ROUNDED]] ...
        = arithmetic-expression
        [; ON SIZE ERROR imperative-statement]

CONTINUE

COPY file-name

DELETE file-name RECORD [; INVALID KEY imperative-statement]
```

GENERAL FORMAT FOR VERBS

```
DISPLAY { identifier-1 } [, identifier-2 ] ... [UPON mnemonic-name]
        { literal-1    }  [, literal-2    ]

        [WITH NO ADVANCING]
```

```
DISPLAY (  { identifier-1 [{+} integer-1] }  { identifier-2 [{+} integer-3] }  )
           {              integer-2       }  {              integer-4       }

        { identifier-3 } [, identifier-4 ] ... [WITH [BEEP]
        { literal-1    }  [, literal-2    ]        [SPACE-FILL]
                                                   [INVERSE-VIDEO]
                                                   [BLINK]
                                                   [UNDERLINE]
                                                   [LOW-INTENSITY]
                                                   [NORMAL]
                                                   [AUTO-ERASE]
                                                   [PROMPT]
                                                   [BLANK-WHEN-ZERO] ]
```

```
DISPLAY (  { identifier-1 [{+} integer-1] }  { identifier-2 [{+} integer-3] }  )
           {              integer-2       }  {              integer-4       }

        FRAME  { identifier-3 } * { identifier-4 }
               { literal-1    }   { literal-2    }

               [WITH [SPACE-FILL] [HEADING] ]
```

```
DISPLAY (  { identifier-1 [{+} integer-1] }  { identifier-2 [{+} integer-3] }  )
           {              integer-2       }  {              integer-4       }

        { FULL-BAR   }  { identifier-3 } * { identifier-4 }
        { SPARSE-BAR }  { literal-1    }   { literal-2    }
```

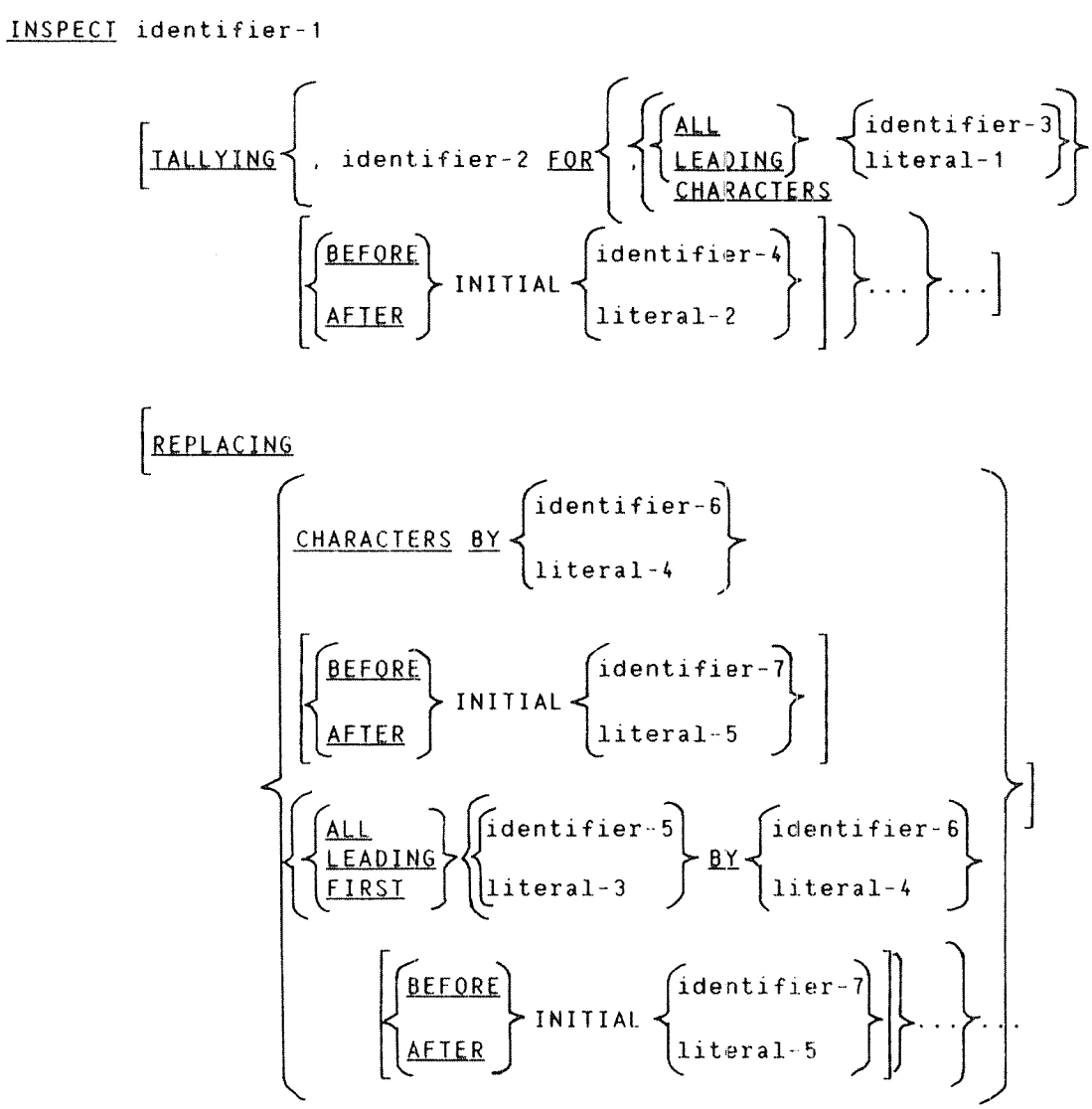
GENERAL FORMAT FOR VERBS

```
DIVIDE {identifier-1} INTO identifier-2  [ROUNDED]
       {literal-1   }
        [, identifier-3  [ROUNDED] ]
        [; ON SIZE ERROR  imperative-statement]

DIVIDE {identifier-1} {INTO} {identifier-2}
       {literal-1   } {BY  } {literal-2   }
        GIVING identifier-3 [ROUNDED] [, identifier-4 [ROUNDED] ] ...
        [; ON SIZE ERROR  imperative-statement]

DIVIDE {identifier-1} {INTO} {identifier-2}
       {literal-1   } {BY  } {literal-2   }
        GIVING identifier-3 [ROUNDED]
        REMAINDER identifier-4
        [; ON SIZE ERROR  imperative-statement]

DO sentence  [ WHILE condition  sentence] ... END-DO

DO FOR identifier-1 FROM {identifier-2 [{+} integer-2]}
                         {             integer-3      }
        [BY integer-4] TO {identifier-5 [{+} integer-5]}
                          {             integer-6      }
        sentence
        [WHILE condition  sentence] ... END-DO
```

GENERAL FORMAT FOR VERBS

```
EXHIBIT NAMED { identifier } ...
              { literal    }

EXIT [PROGRAM]

EXIT-DO

EXIT-ALL-DO

GO TO [procedure-name-1]

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

IF condition { statement-1 } [ELSE { statement-2 }]
             { NEXT SENTENCE }     { NEXT SENTENCE }

IF condition THEN { statement-3 } [ELSE { statement-2 }] [END-IF]
                  { NEXT SENTENCE }     { NEXT SENTENCE }

IF condition THEN { statement-5 }
                  { NEXT SENTENCE }

   [ELSE-IF condition-2 THEN { statement-6 }] ...
                             { NEXT SENTENCE }

   [ELSE { statement-7 }] [END-IF]
         { NEXT SENTENCE }
```

GENERAL FORMAT FOR VERBS



GENERAL FORMAT FOR VERBS

```
MERGE file-name-1 ON { ASCENDING  } KEY data-name-1
                     { DESCENDING }
     [, data-name-2] ...

     [ ON { ASCENDING  } KEY data-name-3 [, data-name-4] ... ]
          { DESCENDING }

     USING file-name-2, file-name-3

     { OUTPUT PROCEDURE IS section-name-3 [ { THROUGH } section-name-4 ] }
     {                                     { THRU    }                 }
     { GIVING file-name-4                                              }
```

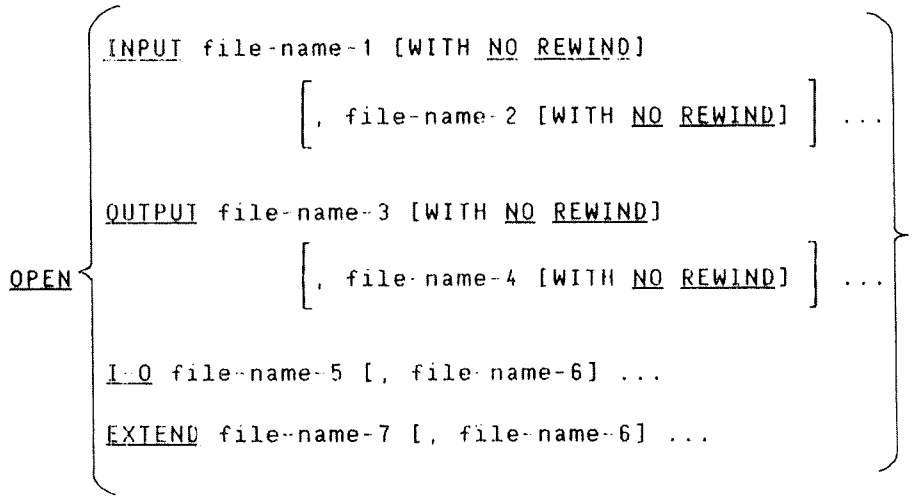
```
MOVE { identifier-1 } TO identifier-2 [, identifier-3] ...
     { literal      }
```

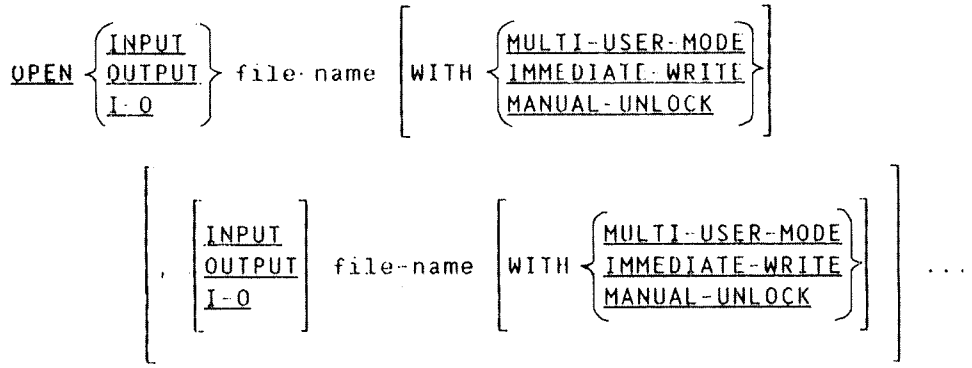
```
MOVE { CORRESPONDING } identifier-1 TO identifier-2
     { CORR          }
```

GENERAL FORMAT FOR VERBS

```
MULTIPLY { identifier-1 } BY identifier-2 [ROUNDED]
         { literal-1    }
       [ , identifier-3 [ROUNDED] ] ...
       [ ; ON SIZE ERROR imperative-statement ]

MULTIPLY { identifier-1 } BY { identifier-2 }
         { literal-1    }   { literal-2    }
       GIVING identifier-3 [ROUNDED]
       [ , identifier-4  [ROUNDED] ] ...
       [ ; ON SIZE ERROR imperative-statement ]
```

GENERAL FORMAT FOR VERBS





GENERAL FORMAT FOR VERBS

```
PERFORM range

PERFORM range { identifier-1 } TIMES
              { integer      }

PERFORM range UNTIL condition-1

PERFORM range VARYING { identifier-5 } FROM { identifier-6 }
                      { index-name-3 }      { index-name-4 }
                                            { literal-3    }
                BY { identifier-7 }
                   { literal-4    } UNTIL condition-1
     [ AFTER { identifier-8 } FROM { identifier-9 }
             { index-name-5 }      { index-name-6 }
                                   { literal-5    }
                BY { identifier-10 }
                   { literal-6     } UNTIL condition-2
     [ AFTER { identifier-11 } FROM { identifier-12 }
             { index-name-8 }      { index-name-8  }
                                   { literal-6     }
                BY { identifier-13 }
                   { literal-7     } UNTIL condition-3 ] ]
```

where range is the construct:

```
procedure-name-1 [ { THROUGH } procedure-name-2 ]
                   { THRU    }
```

GENERAL FORMAT FOR VERBS

```
READ file-name [NEXT RECORD] [INTO identifier] [WITH LOCK]
     [; AT END imperative statement]

READ file-name RECORD [INTO identifier] [WITH LOCK]
     [; KEY IS data-name] [; INVALID KEY imperative-statement]

READ file-name RECORD [INTO identifier] [WITH LOCK]
     [; INVALID KEY imperative-statement]

RELEASE record-name [FROM identifier]

RETURN file-name RECORD [INTO identifier]
     [; AT END imperative statement]

REWRITE record-name [FROM identifier]

REWRITE record-name [FROM identifier]
     [; INVALID KEY imperative-statement]
```

GENERAL FORMAT FOR VERBS

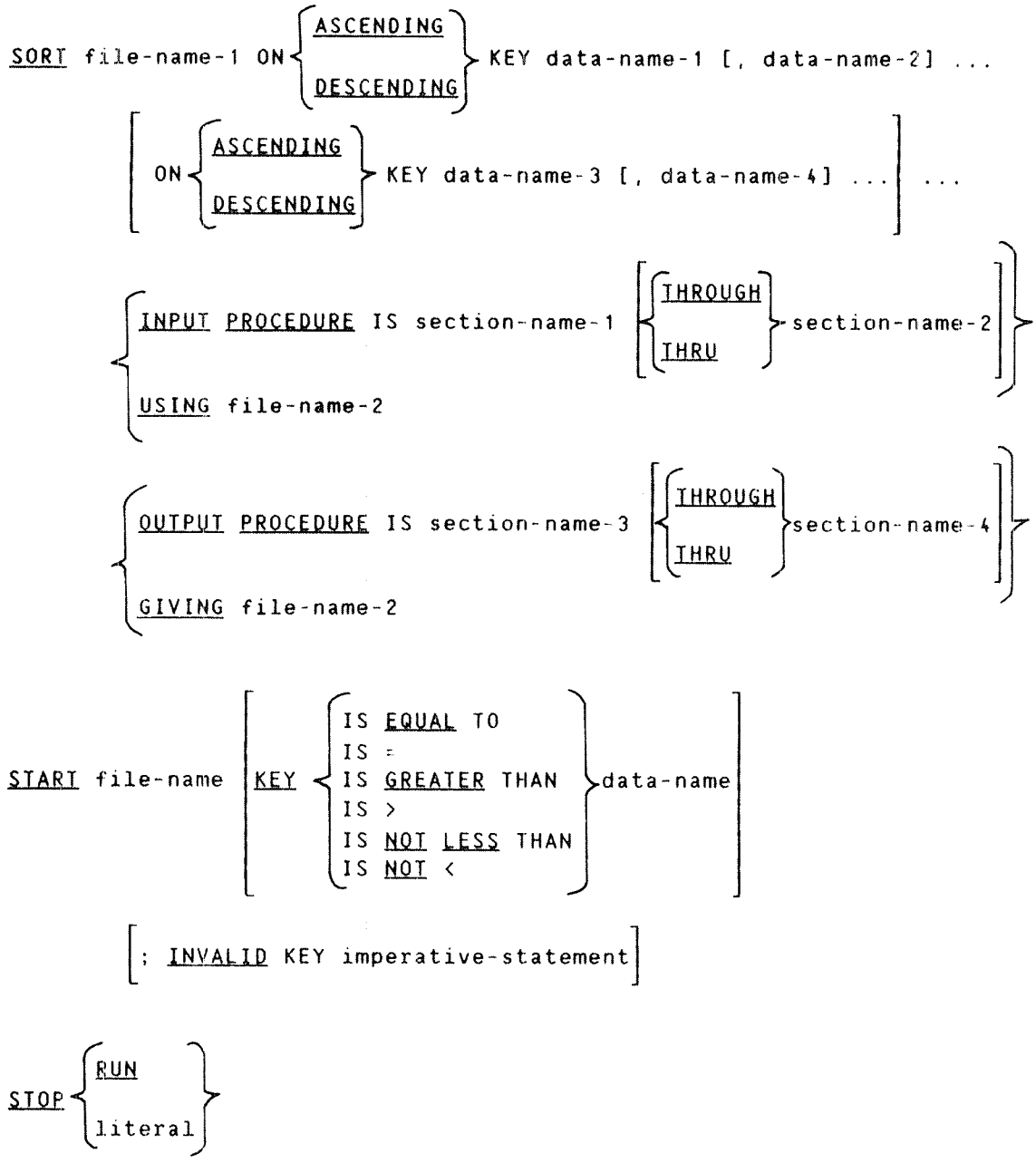
```
SEARCH identifier-1 [ VARYING { identifier-2 } ]
                              { index-name-1 }

     [ ; AT END imperative statement-1 ]

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

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

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

     ; WHEN { data-name-1 { IS EQUAL TO } { identifier-3 }
            {             { IS =       } { literal-1    }
            {                            { arithmetic-expression } }
            { condition-name-1 }

     [ AND { data-name-2 { IS EQUAL TO } { identifier-4 }
           {             { IS =       } { literal-2    }
           {                            { arithmetic-expression } }
           { condition-name-2 } ] ...

     { imperative-statement-2 }
     { NEXT SENTENCE          }
```

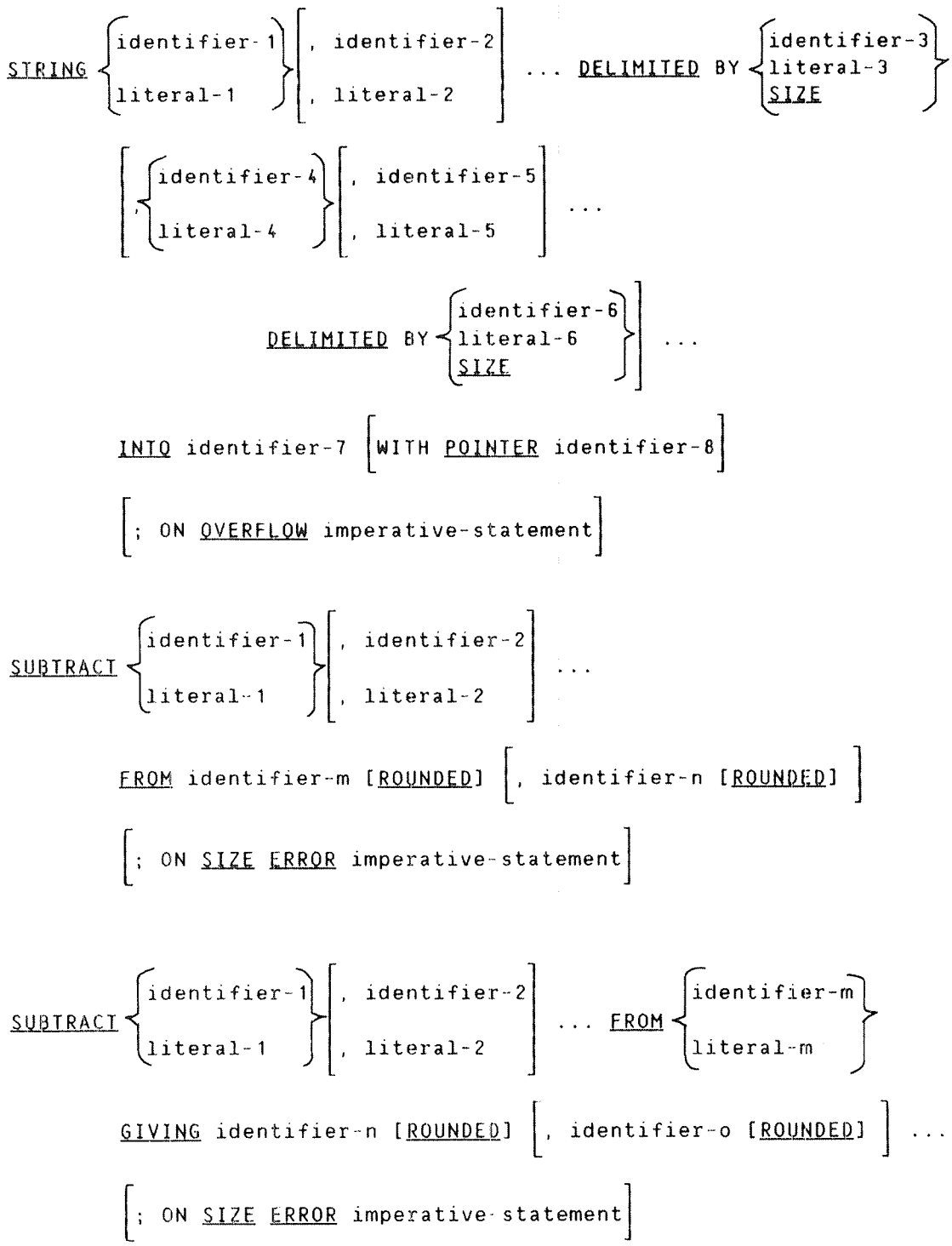
```
SET { identifier-1 [, identifier-2] ... } TO { identifier-3 }
    { index-name-1 [, index-name-2] ...  }    { index-name-3 }
                                              { integer-1    }
```

```
SET index-name-4 [, index-name-5] ... { UP BY   } { identifier-4 }
                                      { DOWN BY } { integer-2    }
```

GENERAL FORMAT FOR VERBS



GENERAL FORMAT FOR VERBS



GENERAL FORMAT FOR VERBS

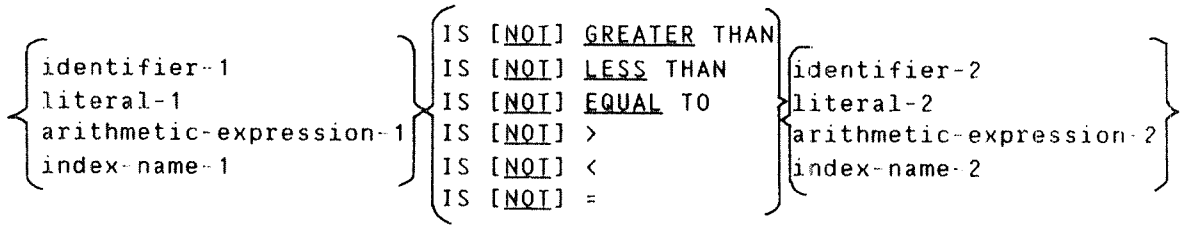
```
UNLOCK file-name

UNSTRING identifier-1  [ DELIMITED BY [ALL] { identifier-2 }
                                            { literal-1    }
                       [ , OR [ALL] { identifier-3 } ] ... ]
                                    { 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 ]

USE AFTER STANDARD { EXCEPTION } PROCEDURE ON { file-name-1                  }
                   { ERROR     }              {    [, filename-2] ...       }
                                              { INPUT                      }
                                              { OUTPUT                     }
                                              { I-O                        }
                                              { EXTEND                     }
```

GENERAL FORMAT FOR VERBS

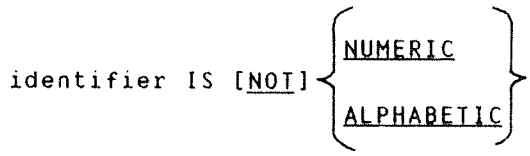
```
WRITE record-name FROM identifier-1
      [ { BEFORE } ADVANCING { { { identifier-3 } [ LINE ] } }
        { AFTER  }           {   { { integer    } [ LINES ] } }
                             {   { PAGE                     } }
                             { }
      ]

WRITE record-name [FROM identifier-1] [WITH LOCK]

      [; INVALID KEY imperative-statement]
```

GENERAL FORMAT FOR CONDITIONS

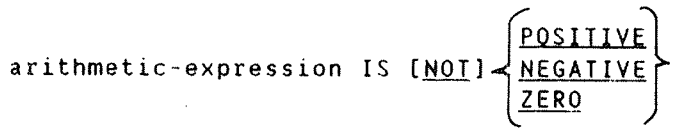
RELATION CONDITION:



CLASS CONDITION:



SIGN CONDITION:



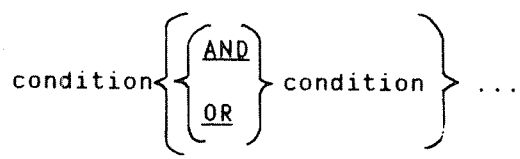
CONDITION-NAME CONDITION:

condition-name

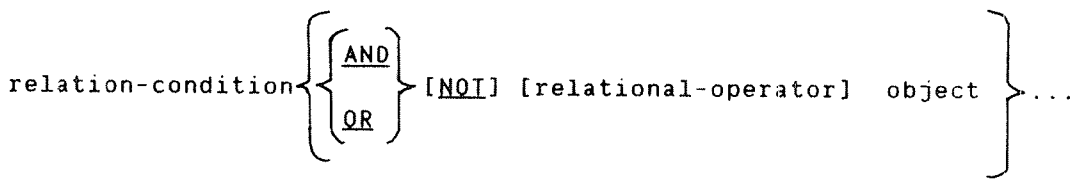
NEGATED SIMPLE CONDITION:

NOT simple condition

COMBINED CONDITION:



ABBREVIATED COMBINED RELATION CONDITION:



MISCELLANEOUS FORMATS

QUALIFICATION:

```
{data-name-1     }  [ {OF} data-name-2 ] ...
{condition-name  }    {IN}
```

```
paragraph-name  [ {OF}  section-name ]
                  {IN}
```

```
file-name
```

SUBSCRIPTING:

```
{data-name     }  ( subscript-1 [, subscript-2 [, subscript-3]] )
{condition-name}
```

INDEXING:

```
{data-name     }  ( {index-name-1  [{+} literal-2]}
{condition-name}    {literal-1                   }

      [, {index-name-2  [{+} literal-4]}
         {literal-3                   }

            [, {index-name-3  [{+} literal-6]}]  ]  )
               {literal-5                   }
```

MISCELLANEOUS FORMATS

IDENTIFIER: FORMAT 1

```
data-name-1 [ { OF } data-name-2 ] ...
              { IN }

    [ ( subscript-1 [, subscript-2 [, subscript-3] ] ) ]
```

IDENTIFIER: FORMAT 2

```
data-name-1 [ { OF } data-name-2 ] ...
              { IN }

    [ ( { index-name-1 [ {±} literal-2 ] }
        { literal-1                     }

          [ , { index-name-2 [ {±} literal-4 ] } ]
              { literal-3                     }

          [ , { index-name-3 [ {±} literal-6 ] } ] ] ) ]
              { literal-5                     }
```

APPENDIX B

ASCII CHARACTER SET

CHAR	Byte Position Left	Right	Dec.	CHAR	Byte Position Left	Right	Dec.
NUL	000000	000000	0	0	030000	000060	48
SOH	000400	000001	1	1	030400	000061	49
STX	001000	000002	2	2	031000	000062	50
ETX	001400	000003	3	3	031400	000063	51
EOT	002000	000004	4	4	032000	000064	52
ENQ	002400	000005	5	5	032400	000065	53
ACK	003000	000006	6	6	033000	000066	54
BEL	003400	000007	7	7	033400	000067	55
BS	004000	000010	8	8	034000	000070	56
HT	004400	000011	9	9	034400	000071	57
LF	005000	000012	10	:	035000	000072	58
VT	005400	000013	11	;	035400	000073	59
FF	006000	000014	12	<	036000	000074	60
CR	006400	000015	13	=	036400	000075	61
SO	007000	000016	14	>	037000	000076	62
SI	007400	000017	15	?	037400	000077	63
DLE	010000	000020	16	@	040000	000100	64
DC1	010400	000021	17	A	040400	000101	65
DC2	011000	000022	18	B	041000	000102	66
DC3	011400	000023	19	C	041400	000103	67
DC4	012000	000024	20	D	042000	000104	68
NAK	012400	000025	21	E	042400	000105	69
SYN	013000	000026	22	F	043000	000106	70
ETB	013400	000027	23	G	043400	000107	71
CAN	014000	000030	24	H	044000	000110	72
EM	014400	000031	25	I	044400	000111	73
SUB	015000	000032	26	J	045000	000112	74
ESC	015400	000033	27	K	045400	000113	75
FS	016000	000034	28	L	046000	000114	76
GS	016400	000035	29	M	046400	000115	77
RS	017000	000036	30	N	047000	000116	78
US	017400	000037	31	O	047400	000117	79
SPACE	020000	000040	32	P	050000	000120	80
!	020400	000041	33	Q	050400	000121	81
"	021000	000042	34	R	051000	000122	82
#	021400	000043	35	S	051400	000123	83
S	022000	000044	36	T	052000	000124	84
%	022400	000045	37	U	052400	000125	85
&	023000	000046	38	V	053000	000126	86
'	023400	000047	39	W	053400	000127	87
(	024000	000050	40	X	054000	000130	88
)	024400	000051	41	Y	054400	000131	89
*	025000	000052	42	Z	055000	000132	90
+	025400	000053	43	[	055400	000133	91
,	026000	000054	44	\	056000	000134	92
—	026400	000055	45	]	056400	000135	93
.	027000	000056	46	^	057000	000136	94
/	027400	000057	47				

CHAR	Byte Position Left	Right	Dec.	CHAR	Byte Position Left	Right	Dec.
_	057400	000137	95	o	067400	000157	111
`	060000	000140	96	p	070000	000160	112
a	060400	000141	97	q	070400	000161	113
b	061000	000142	98	r	071000	000162	114
c	061400	000143	99	s	071400	000163	115
d	062000	000144	100	t	072000	000164	116
e	062400	000145	101	u	072400	000165	117
f	063000	000146	102	v	073000	000166	118
g	063400	000147	103	w	073400	000167	119
h	064000	000150	104	x	074000	000170	120
i	064400	000151	105	y	074400	000171	121
j	065000	000152	106	z	075000	000172	122
k	065400	000153	107	{	075400	000173	123
l	066000	000154	108	|	076000	000174	124
m	066400	000155	109	}	076400	000175	125
n	067000	000156	110	~	077000	000176	126
				DEL	077400	000177	127

APPENDIX C

RUN-TIME MESSAGES

	"Reason"	*Explanation*
600B	I-O — ERR nnn where nnn is a standard SINTRAN III File System Error Code	An error in an I-O operation has arisen without the possibility of user reaction due to omission of any of the following applicable to the file in question: AT END clause INVALID KEY clause USE AFTER STANDARD ERROR (Declarative) FILE STATUS If any *relevant* element above is available, the run-time liabrary routines allow that element to process the data, and THIS ERROR DOES NOT ARISE.
601B	INDEX FILE ERROR nn where nn is a status returned from ISAM	An error has arisen in using the Indexed file system, without the possibility of user reaction due to omission of any of the following clause or to other index file errors: AT END clause INVALID KEY clause
602B	FILE NOT OPEN	Non-addressable data due to an attempt to use data in a file that is not open.
603B	FILE NOT OPEN IN CORRECT MODE	Attempt to use data in a file that is not open in correct mode.
604B	IMPROPER RECORD LENGTH	Incoming record size incorrect when using REWRITE statement.

	"Reason"	*Explanation*

605B	ILLEGAL USE OF REWRITE	Sequence error when using REWRITE. Previous I-O statement not a READ.
606B	OPEN MODE I-O NOT BE USED FOR MAGNETIC TAPE	Magnetic tape files cannot be opened for I-O.
607B	SORT ERROR	Issued by the SORT system.
610B	SORT ERROR — FILE TOO BIG	Issued by the SORT system.
611B	SORT ERROR — TOTAL KEY TOO LONG	Issued by the SORT system.
612B	SORT ERROR IN RECORD SIZE	Issued by the SORT system.
613B	COMPILER/LIBRARY INCOMPATIBILITY	Different versions of the compiler and run-time library cannot be used simultaneously.
* 614B	64/128kw version of COBLIB/ SEPARATE—COBLIB not applicable	If the SEPARATE—CODE—DATA command has been issued, SEPARATE—COBLIB must be loaded, else COBLIB.

* This message is only for the ND-100.

COMPILER ERROR MESSAGES

(ALTERNATE) RECORD KEY MAY ONLY APPLY TO INDEXED FILE
A PARAGRAPH DECLARATION IS REQUIRED HERE
AREA-A VIOLATED; RESTART AT NEXT PARAGRAPH/SECTION/DIVISION/VERB
ASSIGNED WORD MISSING
AT END ONLY LEGAL ON SEQUENTIAL READING.
BAD NESTING OF DO END-DO.
BADLY NESTING OF PARENTHESIS
BLANK WHEN ZERO IS DISALLOWED.
BLOCK SIZE SET TO
BLOCK/RECORD SIZE ILLEGAL.
CLAUSES OTHER THAN VALUE DELETED.
COMP IGNORED FOR DECIMAL ITEM.
CONDITIONAL EXPRESSION IS TOO BIG.
CONFIGURATION SECTION ASSUMED HERE.
CONTINUATION LINE, THEREFORE COL 8-11 MUST BE SPACES.
COPY FILE CANNOT BE FOUND
DATA DIVISION ASSUMED HERE.
DATA RECORDS CLAUSE WAS INACCURATE.
DATA-NAME IN ASSIGN CLAUSE IS UNDEFINED/WRONG TYPE/NOT UNIQUE
DELETE NOT VALID FOR NON-ORGANIZED FILE.
DEPENDING ON DATA-NAME IS UNDEFINED/WRONG TYPE/NOT UNIQUE
ELEMENT IS MALFORMED
ELEMENT NOT DEFINED
ERRONEOUS FILE-NAME IS IGNORED
ERRONEOUS QUALIFICATION; LAST DECLARATION USED.
ERRONEOUS SELECT-SENTENCE; RESUMPTION AT NEXT SELECT OR AREA-A.
ERRONEOUS SUBSCRIPTING; STATEMENT DELETED
EXCESSIVE GROUP SIZE - MAX 32767.
EXCESSIVE OCCURS CLAUSE NESTING IS IGNORED.
EXIT SHOULD BE IN ITS OWN PARAGRAPH.
EXPONENTIATION (**) NOT IMPLEMENTED YET.
EXTERNAL DECIMAL ITEM IS UNSIGNED.
FAULTY QUOTED LITERAL.
FILE NOT SELECTED, ENTRY BYPASSED.
FILE OPENED BUT NEVER CLOSED
FILE SECTION ASSUMED HERE.
GO WITH NO PARAGRAPH-NAME MUST BE IN ITS OWN PARAGRAPH.
GROUP ITEM, THEREFORE PIC/JUST/BLANK/SYNC IS IGNORED.
IDENTIFICATION DIVISION ASSUMED HERE.
ILLEGAL ARITHMETICAL EXPRESSION. STATEMENT DELETED
ILLEGAL BEFORE/AFTER-ADVANCING CLAUSE
ILLEGAL CHARACTER IN COLUMN 7.
ILLEGAL CHARACTER; IGNORED.
ILLEGAL FIGURATIVE CONSTANT
ILLEGAL KEY RELATION
ILLEGAL MOVE OR COMPARISON IS DELETED.
ILLEGAL PERFORM-RANGE.
ILLEGAL USE OF SAME AREA.
IMPERATIVE VERB REQUIRED HERE (NOT ELSE/USE/WHEN/)
IMPROPER REDEFINITION IGNORED.
IMPROPER USE OF 88-LEVEL.
INCOMPLETE (OR TOO LONG) STATEMENT DELETED.
INTEGER LITERAL MUST BE GREATER THAN 0
INVALID BLOCKING IS IGNORED.
INVALID KEY ILLEGAL ON SEQUENTIAL ORGANIZED FILES.
INVALID RECORD SIZE(S) IGNORED.
INVALID VALUE IGNORED.
ITEM ASSUMED TO BE BINARY.

```
KEY DECLARATION OF THIS FILE IS NOT CORRECT.
KEY MUST APPLY ON INDEXED/RELATIVE FILES.
KEY MUST BE DECIMAL OR CHARACTER ITEM.
KEY ONLY LEGAL WITH RANDOM/DYNAMIC ACCESS ON RELATIVE/INDEX FILE.
KEYS MAY ONLY APPLY TO AN INDEXED/RELATIVE FILE
LEVEL 01 ASSUMED.
LEVEL 66 NOT SUPORTED.
LITERAL TRUNCATED TO SIZE OF ITEM.
MAX 10 SORT KEYS ALLOWED
MISORDERED/REDUNDANT SECTION PROCESSED AS IS.
MISSING OPERAND
MISSING PROGRAM-ID/PROGRAM-NAME; DEFAULT PROGRAM NAME = MAINCB.
MNEMONIC-NAME MISSING
NAME OMITTED/DOUBLY DEFINED; ENTRY BYPASSED.
NOT IMPLEMENTED IN THIS COMPILER
NUMBER OF PARAGRAPH-NAMES IN GO-DEPENDING MUST BE 1 TO 100.
OCCURS DISALLOWED AT LEVEL 01/77, OR COUNT TOO HIGH.
OMITTED WORD SECTION IS ASSUMED HERE.
ONLY ACCESS MODE SEQUENTIAL LEGAL ON SEQUENTIAL ORGANIZED FILES.
ONLY GROUP LEVELS ACCEPTED IN MOVE CORRESPONDING
ONLY LEGAL FOR RECORDING MODE V
ONLY RECORDING MODE IS F/TEXT-FILE ALLOWED.
ONLY SEQUENTIAL FILES CAN BE OPENED AS EXTEND.
ONLY SEQUENTIAL FILES CAN BE OPENED/CLOSED WITH NO REWIND.
OPEN MODE (INPUT/OUTPUT/I-O/EXTEND) MISSING
PARAGRAPH REFERENCED BY ALTER MUST CONTAIN SINGLE/SIMPLE GO TO.
PARAGRAPH-NAME REFERENCE MISSING
PERIOD ASSUMED AFTER PROCEDURE-NAME DEFINITION
PICTURE CLAUSE IS BADLY FORMED; PIC X ASSUMED.
PICTURE IGNORED FOR INDEX ITEM.
PROCEDURE DIVISION ASSUMED HERE
PROCEDURE-NAME IS UNRESOLVABLE
QUOTED LITERAL/NUMERIC ELEMENT/NAME IS TOO LONG.
RECORD MIN/MAX DISAGREES WITH RECORD CONTAINS, LATTER SIZE USED.
RECORD SIZE DIFFERENT FROM SD-FILE RECORD SIZE
RECORD SIZE ON INDEXED FILES - MIN 4.
REDUNDANT CLAUSE IGNORED
REDUNDANT FD PROCESSED AS IS.
RELATIVE KEY MAY ONLY APPLY TO RELATIVE FILE
REQUIRED DATA SPACE EXCEEDS MAX AVAILABLE.
RESUMPTION AT NEXT PARAGRAPH/VERB.
RIGHT PARENTHESIS REQUIRED AFTER SUBSCRIPTS
SEQUENTIAL READING IMPOSSIBLE WITH RANDOM ACCESS.
SORRY, NOT ALLOWED INSIDE SEARCH-STATEMENT
SOURCE BYPASSED UNTIL NEXT FD/SECTION
START ONLY ALLOWED ON RELATIVE/INDEX FILES.
STATEMENT DELETED BECAUSE INTEGRAL ITEM IS REQUIRED
STATEMENT DELETED BECAUSE OPERAND IS NOT A FILE-NAME
STATEMENT DELETED DUE TO ERRONEOUS SYNTAX.
STATEMENT DELETED DUE TO MISSING SORT FILE-NAME
STATEMENT DELETED DUE TO NON-NUMERIC OR ILLEGAL TYPE OPERAND
STATUS NAME IS UNDEFINED/WRONG TYPE/NOT UNIQUE
SUBSCRIPT OR INDEX-NAME IS NOT UNIQUE
SYNTAX ERROR (RESUMPTION AT NEXT PARAGRAPH/VERB)
TERMINAL PERIOD ASSUMED ABOVE
TERMINAL PERIOD MISSING
THERE IS NO IF-STATEMENT TO MATCH THIS ELSE/END-IF
THERE IS NO SEARCH TO MATCH THIS WHEN.
USAGE OTHER THAN DISPLAY IGNORED
USING-LIST ITEM LEVEL MUST BE ASSIGNED WORD 01/77/WORD-ALIGNED
VALUE DISALLOWED DUE TO OCCURS/REDEFINES/TYPE CONFLICT.
```

```
WHEN TO MATCH SEARCH IS MISSING.
WHILE NOT INSIDE A DO-LOOP.
WITH NO REWIND ONLY ON FILES OPENED AS INPUT OR OUTPUT.
WORD DECLARATIVES MISSING
WORD MUST NOT START BEFORE COLUMN 12
WORKING-STORAGE ASSUMED HERE.
WRITE BEFORE/AFTER ONLY LEGAL ON SEQUENTIAL FILES.
```

APPENDIX D

RESERVED WORD LIST

ACCEPT
ACCEPT-ERROR
ACCESS
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
AUTO-ERASE
AUTO-SKIP
BEEP
BEFORE
BLANK
BLANK-WHEN-ZERO
BLINK
BLOCK
BOX
BY
CALL
CHARACTER
CHARACTERS
CLOSE
COLUMN
COMMA
COMMON
COMP
COMP-1
COMP-2
COMP-3
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONTAINS
CONTINUE
CONTROL
COPY
CORR
CORRESPONDING
COUNT
CPU-TIME
CURRENCY
DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCEDING
DISPLAY
DIVIDE
DIVISION
DO
DOWN
DUPLICATES
DYNAMIC
ELSE
ELSE-IF
END
END-DO
END-IF
ENVIRONMENT
EQUAL
ERASE
ERROR
EXCEPTION
EXHIBIT
EXIT
EXIT-ALL-DO
EXIT-DO
EXPORT
EXTEND
FD
FILE
FILE-CONTROL
FILE-ID
FILLER
FIRST
FOR
FRAME
FROM
FULL-BAR
GIVING
GO
GREATER
HEADING
HELP
HIGH-VALUE
HIGH-VALUES
HOME
I-O
I-O-CONTROL
IDENTIFICATION
IF
IMMEDIATE-WRITE
IMPORT
IN
INDEX
INDEXED
INITIAL
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
INVERSE-VIDEO
INVISIBLE
IS
JUST
JUSTIFIED
KEY
LABEL
LEADING
LEFT
LENGTH-CHECK
LESS
LINE
LINES
LINKAGE
LOCK
LOW-INTENSITY
LOW-VALUE
LOW-VALUES
MANUAL-UNLOCK
MERGE
MODE
MOVE
MULTI-USER-MODE
MULTIPLY
MUST
NAMED
NEGATIVE
NEXT
NO
NORMAL
NOT
NUMERIC
OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORGANIZATION
OUTPUT
OVERFLOW
PACKED-DECIMAL
PAGE
PERFORM
PIC
PICTURE
POINTER
POSITIVE
PREVIOUS
PROCEDURE
PROCEED
PROGRAM
PROGRAM-ID
PROMPT
QUOTE
QUOTES
RANDOM
RE-DISPLAY
READ
RECORD
RECORDING
RECORDS
REDEFINES
REEL
RELATIVE
RELEASE
REMAINDER
REMARKS
REMOVAL
RENAMES
REPLACING
REPORT
RESERVE
RETURN
REWIND
REWRITE
RIGHT
ROUNDED
RUN
SAME
SCREEN
SD
SEARCH
SECTION
SECURITY
SELECT
SENTENCE
SEPARATE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SOURCE-COMPUTER
SPACE
SPACE-FILL
SPACES
SPARSE-BAR
SPECIAL-NAMES
STANDARD
START
STATUS
STOP
STRING
SUBTRACT
SYNC
SYNCHRONIZED
TALLYING
TEXT-FILE
THAN
THEN
THROUGH
THRU
TIME
TIMES
TO
TRAILING
UNDERLINE
UNIT
UNLOCK
UNSTRING
UNTIL
UP
UPDATE
UPON
UPPER-CASE
USAGE
USE
USING
VALUE
VALUES
VARYING
WHEN
WHILE
WITH
WORKING-STORAGE
WRITE
ZERO
ZEROES
ZEROS

APPENDIX E

CROSS REFERENCE EXAMPLE

To obtain a cross reference listing with a compilation the command

XREF file-name

must be issued at compile-time where 'file-name' is the name of a work file.

```
NORD-10/100 COBOL COMPILER - 11.07.80          TIME: ..     DATE: ..

      1        IDENTIFICATION DIVISION.
      2        PROGRAM-ID.
      3            CROSS-REFERENCE-EXAMPLE.
      4        DATA DIVISION.
      5        WORKING-STORAGE SECTION.
      6        01  PERSON.
      7            03  NAME          PIC X(30)     VALUE "NORSK DATA A/S".
      8            03  ADDRESS       PIC X(30)     VALUE "OSLO, NORWAY".
      9            03  TELEPHONE     PIC 9(11)     VALUE 02309030.
     10            03  INCOME        PIC S9(9)V99 COMP-3.
     11            03  COUNTRY       PIC S9(2)     COMP VALUE 1.
     12                               88  NORWAY    VALUE 1.
     13                               88  SWEDEN    VALUE 2.
     14                               88  DENMARK   VALUE 3.
     15                               88  ENGLAND   VALUE 4.
     16        PROCEDURE DIVISION.
     17        TEST SECTION.
     18        0000.
     19            IF NORWAY          PERFORM 1000.
     20            IF SWEDEN          PERFORM 2000.
     21            IF DENMARK         PERFORM 3000.
     22            IF ENGLAND         PERFORM 4000.
     23            STOP RUN.
     24        1000.
     25            DISPLAY NAME "IS NORWEGIAN".
     26        2000.
     27            DISPLAY NAME "IS SWEDISH".
     28        3000.
     29            DISPLAY NAME "IS DANISH".
     30        4000.
     31            DISPLAY NAME "IS ENGLISH".


*** NO ERROR MESSAGES ***
```

N O R D C O B O L C R O S S R E F E R E N C E L I S T

PROGRAM-ID: CROSS-REFERENCE-EXAMPLE

0000 . . .	(PARAGRAPH)	18				
1000 . . .	(PARAGRAPH)	19	24			
2000 . . .	(PARAGRAPH)	20	26			
3000 . . .	(PARAGRAPH)	21	28			
4000 . . .	(PARAGRAPH)	22	30			
ADDRESS. .	(X 30)	8				
COUNTRYY .	(COMP 2)	11				
DENMARK. .	(88 2)	14	21			
ENGLAND. .	(88 2)	15	22			
INCOME . .	(COMP-3 6)	10				
NAME . . .	(X 30)	7	25	27	29	31
NORWAY . .	(88 2)	12	19			
PERSON . .	(X 80)	6				
SWEDEN . .	(88 2)	13	20			
TELEPHONE.	(NUM 11)	9				
TEST . . .	(SECTION)	17				

APPENDIX F

COMPILER COMMANDS: ND-100

HELP

Lists available commands.

EXIT

Exit to SINTRAN III.

COMPILE <source-file> <list-file> <object-file>

Defines I/O files for the COBOL compiler.

XREF-LIST <work-file>

A cross reference list will be output to the list file, not on the work-file. The parameter <work-file> provides a working file for XREF. Default file type is :XREF.

Example: See Appendix E.

DEBUG-MODE

Debug information will be generated and the Symbolic Debugger can be used. See Symbolic Debugger Users Guide, ND-60.158.

LIBRARY-MODE

The object file will be a library file.

ND100-EXTENDED-MODE

Turns ON the use of the commercial instruction set (COM) in the compiler. If the computer has a commercial instruction set, the speed of execution can be increased by using this command.

Note: on ND-10 the commercial instruction set *must* be installed in order to run COBOL programs.

1-BANK-MODE

If this command is not present the default is 2-bank mode. Normally the code and data are separated, but the use of this command ensures that they are together. The run-time library COBOL-1BANK must be loaded.

TPS-MODE

The compilation will take place under the TPS system.

LOAD file-name [,file-name] ...

To complete the executable program, libraries or other object files may be added by using the above command, where file name is the name of an object file or library.

The default type of the file loaded will be BRF on the ND-100.

LOAD commands will be ignored if they are placed in the source file, or if no PROG-FILE command has been given.

Any error messages which appear while the LOAD command is being executed can be found in the ND Relocating Loader manual (ND-60.066).

COMPILER COMMANDS: ND-500

HELP

Lists available commands.

EXIT

Exit to SINTRAN III.

COMPILE <source-file> <list-file> <object-file>

Defines I/O files for the COBOL compiler.

The default type for the source-file is :SYMB or :COB. For list-file it is :SYMB, and for the object-file it is :NRF.

XREF-LIST <work-file>

A cross reference list will be output to the list file, not on the work-file. The parameter <work-file> provides a working file for XREF. Default file type is :XREF.

DEBUG-MODE

Debug information will be generated and the Symbolic Debugger can be used. See Symbolic Debugger Users Guide, ND-60.158.

LIBRARY-MODE

The object file will be a library file.

APPENDIX G

GLOSSARY

Abbreviated Combined Relation Condition

The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

Access Mode

The manner in which records are to be operated upon within a file.

Actual Decimal Point

The physical representation, using either of the decimal point characters period (.) or comma (,), of the decimal point position in a data item.

Alphabetic Character

A character that belongs to the following set of letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z and the space.

Alphanumeric Character

Any character in the computer's character set.

Alternate Record Key

A key, other than the prime record key, whose contents identify a record within an indexed file.

Arithmetic Expression

An arithmetic expression can be an identifier or a numeric elementary item, a numeric literal, such identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

Arithmetic Operator

A single character, or a fixed two character combination, that belongs to the following set:

Character	Meaning
+	addition
—	subtraction
*	multiplication
/	division
**	exponentiation

Ascending Key

A key upon the values of which data is ordered starting with the lowest value of key up to the highest value of key in accordance with the rules for comparing data items.

Assumed Decimal Point

A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

At End Condition

A condition caused:

1. During the execution of a READ statement for a sequentially accessed file.

2. During the execution of a RETURN statement, when no next logical record exists for the associated sort or merge file.

3. During the execution of a SEARCH statement, when the search operation terminates without satisfying the condition specified in any of the associated WHEN phrases.

Block

A physical unit of data that is normally composed of one or more logical records. For mass storage files, a block may contain a portion of a logical record. The size of a block has no direct relationship to the size of the file within which the block has not direct relationship to the size of the file with which the block is contained or to the size of the logical record(s) that are either continued within the block or that overlap the block. The term is synonymous with physical record.

Called Program

A program which is the object of a CALL statement combined at object time with the calling program to produce a run unit.

Calling Program

A program which executes a CALL to another program.

Character

The basic indivisible unit of the language.

Character Position

A character position is the amount of physical storage required to store a single standard data format character described as usage is DISPLAY.

Character-String

A sequence of contiguous characters which form a COBOL word, a literal, a PICTURE character-string, or a comment entry.

Class Condition

The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic or is wholly numeric.

Clause

A clause is an ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry.

COBOL Word

See word.

Collating Sequence

The sequence in which the characters that are acceptable in a computer are ordered for purposes of sorting, merging and 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 Line

A source program line represented by an asterisk in the indicator area of the line and any characters from the computer's character set in area A and area B of that line. The comment line serves only for documentation in a program. A special form of comment line represented by a stroke (/) in the indicator area of the line and any characters from the computer's character set in area A and area B of that line causes page ejection prior to printing the comment.

Compile Time

The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL object program.

Compiler Directing Statement

A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation.

Complex Condition

A condition in which one or more logical operators act upon one or more conditions. (See Negated Simple Condition, Combined Condition and Negated Combined Condition.)

Computer-Name

A system-name that identifies the computer upon which the program is to be compiled or run.

Condition

A status of a program at execution time for which a truth value can be determined. Where the term 'condition' (condition-1, condition-2, ...) appears in these language specifications in or in reference to 'condition' (condition-1, condition-2, ...) of a general format, it is a conditional expression consisting of either a simple condition optionally parenthesized, or a combined condition consisting of the syntactically correct combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined.

Condition-Name

A user defined word assigned to a specific value, set of values, or range of values, within the complete set of values that a conditional variable may proccess.

Condition-Name Condition

The proposition, for which a truth value can be determined, that the value of a conditional variable is a member of the set of values attributed to a condition-name associated with the conditional variable.

Conditional Expression

A simple condition or a complex condition specified in an IF, PERFORM or SEARCH statement. (See Simple Condition and Complex Condition.)

Conditional Statement

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.

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 that describes overall specifications of source and object computers.

Connective

A reserved word that is used to:

1. Associate a data-name, paragraph-name, condition-name or text-name with its qualifier.

2. Link two or more operands written in a series.

3. Form conditions (logical connectives) (see Logical Operator).

Contiguous Items

Items that are described by consecutive entries in the Data Division and that bear a definite hierarchic relationship to each other.

Currency Sign

The character '$' of the COBOL character set.

Currency Symbol

The character defined by the CURRENCY SIGN clause in the SPECIAL NAMES paragraph. If no CURRENCY SIGN clause is present in a COBOL source program, the currency symbol is identical to the currency sign.

Current Record

The record which is available in the record area associated with the file.

Current Record Pointer

A conceptual entity that is used in the selection of the next record.

Data Clause

A clause that appears in a data description entry in the Data Division and provides information describing a particular attribute of a data item.

Data Description Entry

An entry in the Data Division that is composed of a level number followed by a data name, if required, and then followed by a set of data clauses, as required.

Data Item

A character or a set of contiguous characters (excluding in either case literals) defined as a unit of data by the COBOL program.

Data-Name

A user defined word that names a data item described in a data description entry in the data division. When used in the general formats, 'data-name' represents a word which can neither be subscripted, indexed, nor qualified unless specifically permitted by the rules for that format.

Debugging Line

A debugging line is any line with 'D' in column 7.

Declaratives

A set of one or more special purpose sections, written at the beginning of the Procedure Division, the first of which is preceded by the key word DECLARATIVES and the last of which is followed by the key words END DECLARATIVES. A declarative is composed of a section header, followed by a USE compiler directing sentence, followed by a set of zero, one or more associated paragraphs.

Declarative Sentence

A compiler directing sentence consisting of a single USE statement terminated by the separator period.

Delimiter

A character or a sequence of contiguous characters that identify the end of a string of characters and separates that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

Descending Key

A key upon the values of which data is ordered starting with the highest value of key down to the lowest value of key, in accordance with the rules for comparing data items.

Digit Position

A digit position is the amount of physical storage required to store a single digit. This amount may vary depending on the usage of the physical storage are defined by the implementor.

Division

A set of zero, one or more sections of paragraphs, called the division body, that are formed and combined in accordance with a specific set of rules. There are four (4) divisions in a COBOL program: Identification, Environment, Data and Procedure.

Division Header

A combination of words followed by a period and a space that indicates that beginning of a division. The division headers are:

IDENTIFICATION DIVISION
ENVIRONMENT DIVISION
DATA DIVISION
PROCEDURE DIVISION [USE sentence]

Dynamic Access

An access mode in which specific logic records can be obtained from or placed into a mass storage file in a nonsequential manner (see Random Access) and obtained from a file in a sequential manner (see Sequential Access), during the scope of the same OPEN statement.

Editing Character

A single character or a fixed two character combination belonging to the following set:

Character:	Meaning:
B	space
0	zero
+	plus
—	minus
CR	credit
DB	debit
Z	zero suppress
*	check protect
$	currency sign
,	comma (decimal point)
.	period (decimal point)
/	stroke (virgule, slash)

Elementary Item

A data item that is described as not being further logically subdivided.

End of Procedure Division

The physical position in a COBOL source program after which no further procedures appear.

Entry

Any descriptive set of consecutive clauses terminated by a period and written in the Identification Division, Environment Division or Data Division of a COBOL source program.

Environment Clause

A clause that appears as part of an Environment Division entry.

Execution Time

See Object Time.

Extend Mode

The state of a sequential file after execution of an OPEN statement, with the EXTEND phrase specified, for that file and before the execution of a CLOSE statement for that file.

Figurative Constant

A compiler generated value referenced through the use of certain reserved words.

File

A collection of records.

FILE-CONTROL

The name of an environment division paragraph in which the data files for a given source program are declared.

File Description Entry

An entry in the file sectio of the data division that is composed of the level indicator FD, followed by a file name, and then followed by a set of file clauses as required.

File Name

A user defined word that names a file described in a file description entry or a sort/merge file description entry within the File Section of the Data Division.

File Organization

The permanent logical file structure established at the time that a file is created.

File Section

The section of the Data Division that contains file description entries and sort merge file description entries together with their associated record descriptions.

Format

A specific arrangement of a set of data.

Group Item

A named contiguous set of elementary or group items.

High Order End

The leftmost character of a string of characters.

I-O-Control

The name of an Environment Division paragraph in which object program requirements for specific input-output techniques, rerun points, sharing of same areas of several data files and multiple file storage on a single input-output device are specified.

I-O Mode

The state of a file after execution of an OPEN statement, with the I-O phrase specified, for that file and before the execution of a CLOSE statement for that file.

Identifier

A data name, followed as required, by the syntactically correct combination of qualifiers, subscripts and indices necessary to make unique reference to a data item.

Imperative Statement

A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements.

Index

A computer storage position or register, the contents of which represent the identification of a particular element in a table.

Index Data Item

A data item in which the value associated with an index name can be stored in a form specified by the implementor.

Index Name

A user defined word that names an index associated with a specific table.

Indexed Data Name

An identifier that is composed of a data name, followed by one or more index names enclosed in parentheses.

Indexed File

A file with indexed organization.

Indexed Organization

The permanent logical file structure in which each record is identified by a value of one or more keys within that record.

Input File

A file that is opened in the input mode.

Input Mode

The state of a file after execution of an OPEN statement, with the INPUT phrase specified, for that file and before the execution of a CLOSE statement for that file.

Input-Output File

A file that is opened in the I-O mode.

Input-Output Section

The section of the Environment Division that names the files and the external media required by an object program and which provides information required for transmission and handling of data during execution of the object program.

Input Procedure

A set of statements that is executed each time a record is released to the sort file.

Integer

A numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point. Where the term 'integer' appears in general formats, integer must not be a numeric data item, and must not be signed, nor zero unless explicitly allowed by the rules of that format.

Invalid Key Condition

A condition, at object time, caused when a specific value of the key associated with an indexed or relative file is determined to be invalid.

Key

A data item which identifies the location of a record, or a set of data items which serve to identify the ordering of data.

Key Word

A reserved word whose presence is required when the format in which the word appears is used in a source program.

Level Indicator

Two alphabetic characters that identify a specific type of file or a position in hierarchy.

Level Number

A user defined word which indicates the position of a data item in the hierarchical structure of a logical record or which indicates special properties of a data description entry. A level number is expressed as a one or two digit number. Level numbers in the range 1 through 49 indicate the position of a data item in the hierarchical structure of a logical record. Levels numbers in the range 1 through 9 may be written either as a single digit or as a zero followed by a significant digit. Level numbers 77 and 88 identify special properties of a data description entry.

Library Name

A user defined word that names a COBOL library that is to be used by the compiler for a given source program compilation.

Library Text
A sequence of character strings and/or separators in a COBOL library.

Linkage Section
The section in the Data Division of the called program that describes data items available from the calling program. These data items may be referred to by both the calling and called program.

Literal
A character-string whose value is implied by the ordered set of characters comprising the string.

Logical Operator
One of the reserved words AND, OR or NOT. In the formation of a condition, both or either of AND and OR can be used as logical connectives. NOT can be used for logical negation.

Logical Record
The most inclusive data item. The level number for a record is 01.

Low Order End
The rightmost character of a string of characters.

Mass Storage
A storage medium on which data may be organized and maintained in both a sequential and nonsequential manner.

Mass Storage Control System (MSCS)
An input-output control system that directs, or controls, the processing of mass storage files.

Mass Storage File
A collection of records that is assigned to a mass storage medium.

Merge File
A collection of records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

Mnemonic Name
A user defined word that is associated in the environment division with a specified implementor name.

MSCS
See Mass Storage Control System.

Negated Combined Condition
The 'NOT' logical operator immediately followed by a parenthesized combined condition.

Negated Simple Condition
The 'NOT' logical operator immediately followed by a simple condition.

Next Executable Statement

The next statement to which control will be transferred after execution of the current statement is complete.

Next Record

The record which logically follows the current record of a file.

Noncontiguous Items

Elementary data items, in the Working-Storage and Linkage Sections, which bear no hierarchic relationship to other data items.

Nonnumeric Item

A data item whose description permits its contents to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

Nonnumeric Literal

A character-string bounded by quotation marks. The string of characters may include any character in the computer's character set. To represent a single quotation mark character within a nonnumeric literal, two contiguous quotation marks must be used.

Numeric Character

A character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Numeric Item

A data item whose description restricts its contents to a value represented by characters chosen from the digits '0' through '9'; if signed, the item may also contain a '+', '—', or other representation of an operational sign.

Numeric Literal

A literal composed of one or more numeric characters that also may contain either a decimal point, or an algebraic sign, or both. The decimal point must not be the rightmost character. The algebraic sign, if present, must be the leftmost character.

Object Computer

The name of an Environment Division paragraph in which the computer environment, within which the object program is executed, is described.

Object Program

A set or group of executable machine language instructions and other material designed to interact with data to provide problem solutions. In this context, an object program is generally the machine language result of the operation of a COBOL compiler on a source program. Where there is no danger of ambiguity, the word 'program' alone may be used in place of the phrase 'object program'.

Object Time

The time at which an object program is executed.

Open Mode

The state of a file after execution of an OPEN statement for that file and before the execution of 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

Whereas the general definition of operand is 'that component which is operated upon', for the purposes of this publication, any lower case word (or words) that appears in a statement or entry format may be considered to be an operand and, as such, is an implied reference to the data indicated by the operand.

Operational Sign

An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

Operational Sign

An algebraic sign, associated with a numeric data item or a numeric literal, to indicate whether its value is positive or negative.

Option

A phrase in which a choice can be made between alternate wordings.

Optional Word

A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

Output File

A file that is opened in either the output mode or extended mode.

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.

Phrase

A phrase is an ordered set of one or more consecutive COBOL character strings that form a portion of a COBOL procedural statement or of a COBOL clause.

Physical Record

See Block.

Prime Record Key

A key whose contents uniquely identify a record within an indexed file.

Procedure

A paragraph or group of logically successive paragraphs or a section or group of logically successive sections, within the Procedure Division.

Procedure Name

A user defined word which is used to name a paragraph or section in the Procedure Division. It consists of a paragraph name (which may be qualified), or a section name.

Program Name

A user defined word that identifies a COBOL source program.

Punctuation Character

A character that belongs to the following set:

Character:	Meaning:
,	comma
;	semicolon
.	period
"	quotation mark (double)
(	left parenthesis
)	right parenthesis
	space
=	equal sign

Qualified Data-Name

An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF and IN followed by a data-name qualifier.

Qualifier

1. A data-name which is used in a reference together with another data name at a lower level in the same hierarchy.

2. A section name which is used in a reference together with a paragraph name specified in that section.

3. A library name which is used in a reference together with a text name associated with that library.

Random Access

An access mode in which the program specified value of a key data item identifies the logical record that is obtained from, deleted from or placed into a relative or indexed file.

Record

See Logical Record.

Record Area

A storage area allocated for the purpose of processing the record described in a record description entry in the file section.

Record Description

See Record Description Entry.

Record Description Entry

The total set of data description entries associated with a particular record.

Record Key

A key, either the prime record key or an alternate record key, whose contents identify a record within an indexed file.

Record Name

A user defined word that names a record described in a record description entry in the data division.

Reference Format

A format that provides a standard method for describing COBOL source programs.

Relation

See Relational Operator.

Relation Character

A character that belongs to the following set.

Character:	Meaning:
>	greater than
<	less than
=	equal to

Relation Condition

The proposition, for which a truth value can be determined, that the value of an arithmetic expression or data item has a specific relationship to the value of another arithmetic expression or data item (see Relational Operator).

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 in the construction of a relation condition. The permissible operators and their meanings are:

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

Relative File

A key whose contents identify a logical record in a relative file.

Relative Organization

The permanent logical file structure in which each record is uniquely identified by an integer value greater than zero, which specifies the record's logical ordinal position in the file.

Reserved Word

A COBOL word specified in the list of words which may be used in COBOL source programs, but which must not appear in the programs as user defined words or system names.

Routine Name

A user defined word that identifies a procedure written in a language other than COBOL.

Run Unit

A set of one or more object programs which function, at object time, as a unit to provide problem solutions.

Section

A set of zero, one, or more paragraphs or entries, called a section body, the first of which is preceded by a section header. 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 and Procedure Division.

In the Environment and Data Divisions, a section header is composed of reserved words followed by a period and a space. The permissible section headers are:

In the Environment Division:

CONFIGURATION SECTION
INPUT-OUTPUT SECTION

In the Data Division:

FILE SECTION
WORKING-STORAGE SECTION
LINKAGE SECTION

In the Procedure Division, a section header is composed of a section name, followed by the reserved word SECTION, followed by a period and a space.

Section Name
A user defined word which names a section in the Procedure Division.

Sentence
A sequence of one or more statements, the last of which is terminated by a period followed by a space.

Separator
A punctuation character used to delimit character strings.

Sequential Access
An access mode in which logical records are obtained from or placed into a file in a consecutive predecessor to successor logical record sequence determined by the order of records in the file.

Sequential File
A file with sequential organization.

Sequential Organization
The permanent logical file structure in which a record is identified by a predecessor successor relationship established when the record is placed into the file.

Sign Condition
The proposition, for which a truth value can be determined, that the algebraic value of a data item or an arithmetic expression is either less than, greater than, or equal to zero.

Simple Condition

Any single condition chosen from the set:

relation condition
class condition
condition-name condition
switch status condition
sign condition
(simple condition)

Sort File

A collection of records to be sorted by a SORT statement. The sort file is created and can be used by the sort function only.

Sort/Merge File Description Entry

An entry in the file section of the Data Division that is composed of the level indicator SD, followed by a file name, and then followed by a set of file clauses as required.

SOURCE COMPUTER

The name of an Environment Division paragraph in which the computer environment, within which the source program is compiled, is described.

Source Program

Although it is recognized that a source program may be represented by other forms and symbols, in this document it always refers to a syntactically correct set of COBOL statements beginning with an Identification Division and ending with the end of a Procedure Division. In contexts where there is no danger of ambiguity the word "program" alone may be used in place of the phrase "source program".

Special Character

A character that belongs to the following set:

Character:	Meaning:
+	plus sign
–	minus sign
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma (decimal point)
;	semicolon
.	period (decimal point)
"	quotation mark (double)
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol
'	quotation mark (single)

Special Character Word
A reserved word which is an arithmetic operator or a relation character.

SPECIAL NAMES
The name of an Environment Division paragraph in which implementor names are related to user specified mnemonic names.

Special Registers
Compiler generated storage areas whose primary use is to store information produced in conjunction with the use of specific COBOL features.

Statement
A syntactically valid combination of words and symbols written in the Procedure Division beginning with a verb.

Subprogram
See Called Program.

Subscript
An integer whose value identifies a particular element in a table.

Subscripted Data Name
An identifier that is composed of a data name followed by one or more subscripts enclosed in parentheses.

System Name
A COBOL word which is used to communiate with the operating environment.

Table
A set of logically consecutive items of data that are defined in the Data Division by means of the OCCURS clause.

Table Element
A data item that belongs to the set of repeated items comprising a table.

Truth Value
The representation of the result of the evaluation of a condition in terms of one or two values

true
false

Unary operator
A plus (+) 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 of +1 or –1 respectively.

Unit
A module of mass storage the dimensions of which are determined by each implementor.

User Defined Word

A COBOL word that must be supplied by the user to satisfy the format of a clause or statement.

Variable

A data item whose value may be changed by execution of the object program. A variable used in an arithmetic expression must be a numeric elementary item.

Verb

A word that expresses an action to be taken by a COBOL compiler or object program.

Word

A character-string of not more than 30 characters which forms a user defined word, a system name or a reserved word.

Working-Storage Section

The section of the Data Division that describes Working-Storage data items, composed either of noncontiguous items or of Working-Storage records or of both.

77 Level Description Entry

A data description entry that describes a noncontiguous data item with the level number 77.

APPENDIX H

SIZE OF TEMPORARY FIELDS

Execution by the compiler of certain arithmetic statements or operations can generate intermediate results which will be held in *temporary fields.*

Intermediate results can be obtained when:

1. A COMPUTE statement assigns the value of an arithmetic expression to more than one data item.

2. ADD or SUBTRACT statements are encountered which have multiple operands immediately following the verb.

3. IF or PERFORM statements containing arithmetic expressions are executed.

Using the COMPUTE statement as an example, the size of temporary fields can be ascertained as follows.

Each numeric item within the arithmetic expression is examined and a temporary field formed which can contain the maximum number of digit positions found in any examined item before the decimal point, concatenated with the maximum number of digit positions of any examined item following a decimal point.

EXAMPLE:

If we have:

```
COMPUTE X = A * B
```

where A is declared as PIC S9(5)V99999
and B is declared as PIC S9(7)V9999

Then the temporary field will have a size of:

```
S9(7)V99999
```

If the total number of positions when the concatenation is carried out is greater than 18, then the number of digit positions will be truncated from the right.

COMPUTATIONAL DATA ITEMS

The size of a field for a COMPUTATIONAL item is a single word where there are four or less integer positions before the decimal point, and a double word where there are five or more such positions.

However, for the purposes of calculating the sizes of temporary fields, a COMPUTATIONAL item occupying a single word and *not* having a picture definition, is treated as though it has *five* places before the decimal point, and a double word item as though it has *ten*. COMPUTATIONAL items are integers and have no places after the decimal point.

A COMPUTATIONAL item with a picture definition will have a temporary field formed containing the maximum number of digit positions, concatenated with a sign position.

For example, An item declared as:

```
PIC  S9(2)  COMPUTATIONAL
```

with a value of 11, will have a temporary field of 3 positions.

APPENDIX I

INDEXED/RELATIVE I-O STATUS SUMMARY

The following table summarizes possible statuses from the Indexed/Relative I-O verbs:

STATUS		VERB							
Code (PICTURE XX)	Meaning	OPEN	CLOSE	READ NEXT	READ	WRITE	REWRITE	DELETE	START
'00'	OK	X	X	X	X	X	X	X	X
'10'	End of file			X					
'21'	Wrong sequence of words			X			X	X	
'22'	Duplicates not allowed					X	X		
'23'	Record not found				X			X	X
'24'	No more space on file					X			
'68'	Record locked by another program			X[1]	X[1]		X	X	
'78'	Record modified by another program						X	X	
'94'	Error flag set	X							
'95'	File not initialized	X							
'97'	File access violation	X		X	X	X	X	X	X
'98'	(See note)*	X[2]	X[3]						
'99'	SINTRAN file system error	X	X						

*Note: 1 : Read with lock
2 : File does not correspond to given description
3 : File already closed

APPENDIX J

EXECUTING A SIMPLE PROGRAM

In this appendix, examples are given of how to compile, load and run a simple program on the ND-100 and ND-500 computers. The program also demonstrates some of the features of the ND COBOL.

To try out the example, you must first write this program onto the file "X-001:symb", using one of the ND editors:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    X-001.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  NAME          PIC X(30).
01  I             COMP.

PROCEDURE DIVISION.
1000.  BLANK SCREEN.
       DISPLAY (10,  1) "Your name:"
       ACCEPT  (10, 12) NAME WITH PROMPT.
       BLANK LINE 10.
       DISPLAY ( 1,  1) FRAME 18 * 75 WITH HEADING.
       DIAPLAY ( 2, 28) "M y   n a m e   i s".
       DO FOR I FROM 4 to 17
          DISPLAY (I,  3) NAME WITH BLINK
          DISPLAY (I, 42) NAME WITH INVERSE-VIDEO
       END-DO.
       DISPLAY (22, 10) "You have now used the ND COBOL System"
                        WITH UNDERLINE.
       STOP RUN.
```

J.1 Running the Example on an ND-100 Computer

The following listing of a terminal session shows how to compile, load and execute the program on an ND-100:

In this example, the user's inputs are shown underlined, with a "←" indicating a carriage return from the user.

```
@COBOL←

ND-100 COBOL Compiler ... 203075-F

*COMPILE X-001, 1, "X-001"←

ND-100  COBOL  Compiler  ...  203075-F    TIME:  14.49.18  DATE:
                                                        84.01.11

  SOURCE FILE: X-001
  OBJECT FILE: "X-001"
  MODES:       2-BANK

     1      IDENTIFICATION DIVISION.
     2      PROGRAM-ID.  X-001.
     3
     4      DATA DIVISION.
     5      WORKING-STORAGE SECTION.
     6      01  NAME          PIC X(30).
     7      01  I             COMP.
     8
     9      PROCEDURE DIVISION.
    10      1000.  BLANK SCREEN.
    11             DISPLAY (10,  1) "Your name:".
    12             ACCEPT  (10, 12) NAME WITH PROMPT.
    13             BLANK LINE 10.
    14             DISPLAY ( 1,  1) FRAME 18 * 75 WITH HEADING.
    15             DISPLAY ( 2, 28) "M y  n a m e  i s".
    16             DO FOR I FROM 4 TO 17
    17                DISPLAY (I,  3) NAME WITH BLINK
    18                DISPLAY (I, 42) NAME WITH INVERSE-VIDEO
    19             END-DO.
    20             DISPLAY (22, 10)
                       "You have now used the ND COBOL System"
    21                                 WITH UNDERLINE.
    22             STOP RUN.

*** NO ERRORS FOUND ***
```

```
*EXIT←

@NRL←
RELOCATING LOADER LDR-1935I
*PROG-FILE "X-001"←
*LOAD X-001←
FREE: 000211-177777 .... FREE DATA AREA: 000464-177777
*LOAD COBOL-2BANK←
FREE: 031635-177777 .... FREE DATA AREA: 006717-177777
*EXIT←

@X-001←
Your name: <YOUR NAME>← ..................
```

.. and watch the screen.

If you know how to use MODE-files, then try executing a :MODE-file with the following contents:

```
@COBOL
COMPILE X-001, 1, "X-001"
EXIT

@NRL
PROG-FILE "X-001"
LOAD X-001
LOAD COBOL-2BANK
EXIT
```

Then, try executing X-001.

J.2 Running the Example on an ND-500 Computer

The following listing of a terminal session shows how to compile, load and execute the program on an ND-500:

In this example, the user's inputs are shown underlined, with a ''←'' indicating a carriage return from the user.

```
@ND←
ND-500 MONITOR VERSION E      83.12. 9 / 83.12.13
N500: COBOL←

ND-500 COBOL Compiler - 11. Jan 1984

*COMPILE X-001,1,"X-001"←
ND-500 COBOL Compiler - 11. Jan 1984     TIME 10.15.44     DATE:
                                                          84.01.18

  SOURCE FILE: X-001
  OBJECT FILE: "X-001"

        1         IDENTIFICATION DIVISION.
        2         PROGRAM-ID.   X-001.
        3
        4         DATA DIVISION.
        5         WORKING-STORAGE SECTION.
        6         01 NAME           PIC X(30).
        7         01 I              COMP.
        8
        9         PROCEDURE DIVISION.
       10         1000.  BLANK SCREEN.
       11                DISPLAY (10,  1) "Your name:".
       12                ACCEPT  (10, 12) NAME WITH PROMPT.
       13                BLANK LINE 10.
       14                DISPLAY ( 1,  1) FRAME 18 * 75 WITH
                  HEADING.
       15                DISPLAY ( 2, 28) "My  name  is".
       16                DO FOR I FROM 4 TO 17
       17                   DISPLAY (I,  3) NAME WITH BLINK
       18                      DISPLAY   (I,   42)   NAME   WITH
                  INVERSE-VIDEO
       19                END-DO.
       20                DISPLAY (22, 10) "You have now used the
                                                  ND COBOL System"
       21                                WITH UNDERLINE.
       22                STOP RUN.

*** NO ERRORS FOUND ***
```

```
*EXIT←
N500: LINK-LOAD←

ND-Linkage-Loader - F                    10. September 1983 Time: 00.07

Nll entered:                             18. January   1984 Time: 10:15
Nll: SET-DOMAIN "X-001"←

The "DESCRIPTION-FILE" will now be initialized
Nll: LOAD X-001←
Program:.....561 P          Data: .....1124 D01
Nll: LOAD COBOL-LIB←
COBOL-LIB-F-840109
Program:.....33170 P01      Data: .....21034 D01
Nll: LOAD EXCEPTION-LIB←
EXCEPTION-LIB-204157A
Porgram:.....33170 P01      Data: .....21034 D01
Nll: EXIT←

N500: X-001←
```

Then, see what happens.

If you know how to use MODE-files, then try executing a :MODE-file with the following contents:

```
@ND
COBOL
COMPILE X-001, 1, "X-001"
EXIT

LINK-LOAD
SET-DOMAIN "X-001"
LOAD X-001
LOAD COBOL-LIB
LOAD EXCEPTION-LIB
EXIT
```

Then, try executing X-001:

```
@ND←
ND-500: X-001←
```

etc.

INDEX

********** SEND US YOUR COMMENTS!!! **********

Are you frustrated because of unclear information in this manual? Do you have trouble finding things? Why don't you join the Reader's Club and send us a note? You will receive a membership card — and an answer to your comments.

Please let us know if you

* find errors
* cannot understand information
* cannot find information
* find needless information

Do you think we could improve the manual by rearranging the contents? You could also tell us if you like the manual!

********** HELP YOURSELF BY HELPING US!! **********

Manual name: ND COBOL Reference Manual

Manual number: ND-60.144.02 Rev. B

What problems do you have? (use extra pages if needed) _____

_____

_____

_____

_____

Do you have suggestions for improving this manual ? _____

_____

_____

_____

_____

_____

Your name: _____ Date: _____

Company: _____ Position: _____

Address: _____

_____

What are you using this manual for ? _____

_____

NOTE!

This form is primarily for documentation errors. Software and system errors should be reported on Customer System Reports.

Send to:

Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
Oslo 6, Norway

Norsk Data's answer will be found on reverse side

Answer from Norsk Data _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Answered by _____ Date _____

Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
Oslo 6, Norway

Systems that put people first

NORSK DATA A.S OLAF HELSETS VEI 5 P.O. BOX 25 BOGERUD 0621 OSLO 6 NORWAY
TEL.: 02 - 29 54 00 - TELEX: 18284 NDN