60450100

CONTROL DATA CORPORATION

NOS VERSION 1 MODIFY REFERENCE MANUAL

CDC® COMPUTER SYSTEMS:
- CYBER 170
  - MODELS 171, 172, 173, 174, 175
- CYBER 70
  - MODELS 71, 72, 73, 74
- 6000 SERIES

ALPHABETIZED DIRECTIVES INDEX

Name	Parameters	Page Number
*BKSP	file	5-2
*BKSP	file, n	5-2
*CALL	deckname	6-1
*CALLALL	string	6-2
*COMMENT	comment	6-2
*COPY	file, deckname	3-3
*COPYPL	file, deckname	3-2
*CREATE	file	3-2
*CWEOR		6-3
*DECK	deckname	4-2
*DEFINE	name	7-1
*DELETE	c	4-2
*D	c	4-2
*DELETE	c_1, c_2	4-2
*D	c_1, c_2	4-2
*EDIT	$p_1, p_2, \ldots, p_n$	4-4
*ELSE		6-2
*ENDIF		6-2
*IDENT	modname	4-1
*IF	atr, name, value	6-2
*IFCALL	name, deckname	6-1
*IGNORE	dname	4-3
*INSERT	c	4-3
*I	c	4-3
*INWIDTH	n	7-1
*MODNAME	modname	4-2
*MOVE	$\text{dname}_r, \text{dname}_1, \text{dname}_2, \text{dname}_3, \ldots, \text{dname}_n$	7-2
*NIFCALL	name, deckname	6-1
*NOSEQ		3-3, 6-3
*OPLFILE	$\text{file}_1, \text{file}_2, \ldots, \text{file}_n$	3-2
*PREFIX	x	7-1
*PREFIXC	x	7-1
*PURDECK	$\text{dname}_1, \text{dname}_2, \ldots, \text{dname}_n$	4-3
*PURDECK	$\text{dname}_a. \text{dname}_b$	4-3
*READ	file	5-1
*READ	file, deckname	5-1
*READ	file, *	5-1
*READPL	dname	5-1
*READPL	dname, c_1, c_2	5-1
*RESTORE	c	4-2
*RESTORE	c_1, c_2	4-2
*RETURN	$\text{file}_1, \text{file}_2, \ldots, \text{file}_n$	5-2
*REWIND	$\text{file}_1, \text{file}_2, \ldots, \text{file}_n$	5-2
*SEQ		6-3
*SKIP	file	5-2
*SKIP	file, name	5-2
*SKIPR	file, rname	5-2
*UNYANK	modname	4-3
*UNYANK	modname, *	4-3
*UPDATE		7-2
*WEOF		6-3
*WEOR		6-3
*WIDTH	n	3-3, 6-2
*YANK	modname	4-3
*YANK	modname, *	4-3
*/	comment	7-1

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released.
(3-8-76)	
B	Revised to update the manual to NOS 1.2 at PSR level 439, and to make typographical and
(12-3-76)	technical corrections. New directives IF, ELSE, ENDIF, and NIFCALL are added. The
	previous DEFINE directive has a new parameter added that allows a value to be associated
	with a defined name. This edition obsoletes the previous edition.
C	Revised to update the manual to NOS 1.2 at PSR level 452, to reformat error messages, and to
(7-15-77)	make typographical and technical corrections. Support of CDC CYBER 170 Series, Model 171 is
	also included.
Publication No. 60450100	

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning this manual to:

Control Data Corporation
Publications and Graphics Division
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV	PAGE	REV	PAGE	REV	PAGE	REV	PAGE	REV
Front Cover	-	B-2	C						
Inside Front Cover	-	B-3	C						
Title Page	-	B-4	C						
ii	C	B-5	C						
iii/iv	C	C-1	A						
v/vi	C	C-2	C						
vii	B	C-3	A						
viii	C	C-4	A						
1-1	B	Index-1	B						
1-2	B	Index-2	C						
1-3	C	Index-3	B						
1-4	B	Comment Sheet	C						
2-1	C	Inside Back Cover	-						
3-1	B	Back Cover	-						
3-2	C								
3-3	B								
3-4	C								
3-5	C								
3-6	C								
4-1	A								
4-2	B								
4-3	A								
4-4	B								
4-5	A								
4-6	A								
5-1	B								
5-2	B								
5-3	A								
5-4	A								
6-1	C								
6-2	B								
6-3	B								
6-4	B								
6-5	B								
6-6	B								
7-1	B								
7-2	B								
7-3	A								
8-1	C								
8-2	B								
8-3	A								
9-1	B								
9-2	A								
9-3	B								
9-4	B								
10-1	A								
10-2	A								
10-3	B								
10-4	A								
10-5	A								
10-6	A								
10-7	A								
10-8	B								
10-9	A								
10-10	A								
10-11	A								
A-1	A								
B-1	C								

PREFACE

INTRODUCTION

This manual describes the program library maintenance utility Modify. Modify is part of the Network Operating System (NOS) for CONTROL DATA® CYBER 170 Series, Models 171, 172, 173, 174, and 175 Computer Systems; CDC® CYBER 70 Series, Models 71, 72, 73, and 74 Computer Systems; and CDC® CYBER 6000 Series Computer Systems. Modify is used to maintain and update source files that are on libraries in a compressed and symbolic format.

The introduction describes features of Modify and presents an overview of its operation. The remaining sections describe the directives that the user supplies to control library creation and editing. Because the advantages of Modify are best utilized by a programmer with a large volume of source program text or symbolic data, the manual is written for the experienced NOS applications or systems programmer. Wherever possible, Modify usage is illustrated through examples.

Appendix C describes the NOS utility OPLEDIT, which provides the capability to delete and reconstruct previous modification sets.

RELATED PUBLICATIONS

For further information concerning Modify and NOS, consult the following manuals.

Control Data Publication	Publication Number
NOS Modify Instant	60450200
NOS Reference Manual, Volume 1	60435400
NOS Applications Programmer's Instant	60436000
NOS Time-Sharing User's Reference Manual	60435500
NOS Terminal User's Instant	60435800

DISCLAIMER

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTENTS

APPENDIXES

INDEX

FIGURES

INTRODUCTION 1

Modify is used by the programmer to maintain text (large programs or data files) in a compressed form allowing him to easily change individual lines within the text. Modify transforms text into a specially formatted file whose structure enables Modify to make requested changes (or rescind previously made changes) efficiently. Such a file, a program library file, is in program library or Modify format. Once this file has been established, the user need only specify to Modify the changes he is making to the text. Modify then performs the requested changes and produces several files of different types which reflect the changes. One of these files is the compile file, a text file acceptable to language processors (for example, FORTRAN, BASIC, or COMPASS). This file can also be directed to an output device for listing or punching.

MODIFY ORGANIZATION

Modify can be organized into three main functional elements:

- Files used to initialize the program library – these contain the program text from which Modify establishes the program library, the body of text upon which modification directives act to effect user-requested changes to the text.
- Directives – these are user-specified instructions to Modify which establish the program library, produce changes in the text, perform various utility functions upon files used by Modify, and/or alter certain operational characteristics of Modify.
- Output files – these are produced by Modify after it performs the instructions specified by directives. Three of these files are updated versions (in different formats) of the original text; the fourth is a report of actions taken during Modify's execution.

Refer to figure 1-1 during the following discussion of the elements of Modify organization.

FILES USED TO INITIALIZE PROGRAM LIBRARY

These files contain program text in one of two forms: source format or program library format. Files used to initialize the program library may contain several program and/or subroutine decks, kept as separate logical records on the file. The user can designate a deck containing frequently used lines (such as a group of FORTRAN COMMON statements) as a common deck. The user can then direct Modify to insert the text of a common deck within the program text wherever a CALL directive appears within the program text (refer to section 6 for further information on the CALL directive).

Source-format files are coded text files, typically prepared either as a card deck or through the text-file creation facilities of the NOS time-sharing subsystem (refer to the NOS Time-Sharing User's Reference Manual). All program library files begin as source-format files, which Modify processes to create program library files.

A file in program library format is defined as follows.

- It is compressed (Modify has replaced three or more consecutive blanks within a line with special codes).
- Each line of text has been assigned, by Modify, a sequence number and name, thereby allowing the user to refer to individual lines when he wishes to change the text on subsequent Modify runs.
- It contains a directory, built by Modify, which serves as an index of the decks on the program library file.

DIRECTIVES

The user can control Modify execution by specifying directives to Modify. These directives (compile file directives excepted) form a logical record on a file which the user specifies on the Modify control statement. If Modify is being executed from a time-sharing terminal, Modify prompts the user for directives, unless he has specified otherwise on the Modify control statement.

The user may direct Modify to begin reading directives from an alternate file and position this file (or other files local to his job) with file manipulation directives. Certain files (refer to section 5) cannot be operated on by these directives.

Initialization directives declare which files Modify is to use to initialize the program library. They indicate whether the file is in source format (thereby causing Modify to make a copy of it in program library format) or is in program library format.

Directives which cause text to be changed fall into two groups: modification directives and compile file directives.

Modification directives specify line-by-line alterations (insertion; deletion or deactivation; and reactivation) for Modify to make. They also specify which decks Modify should copy to its output files with the specified modifications included.

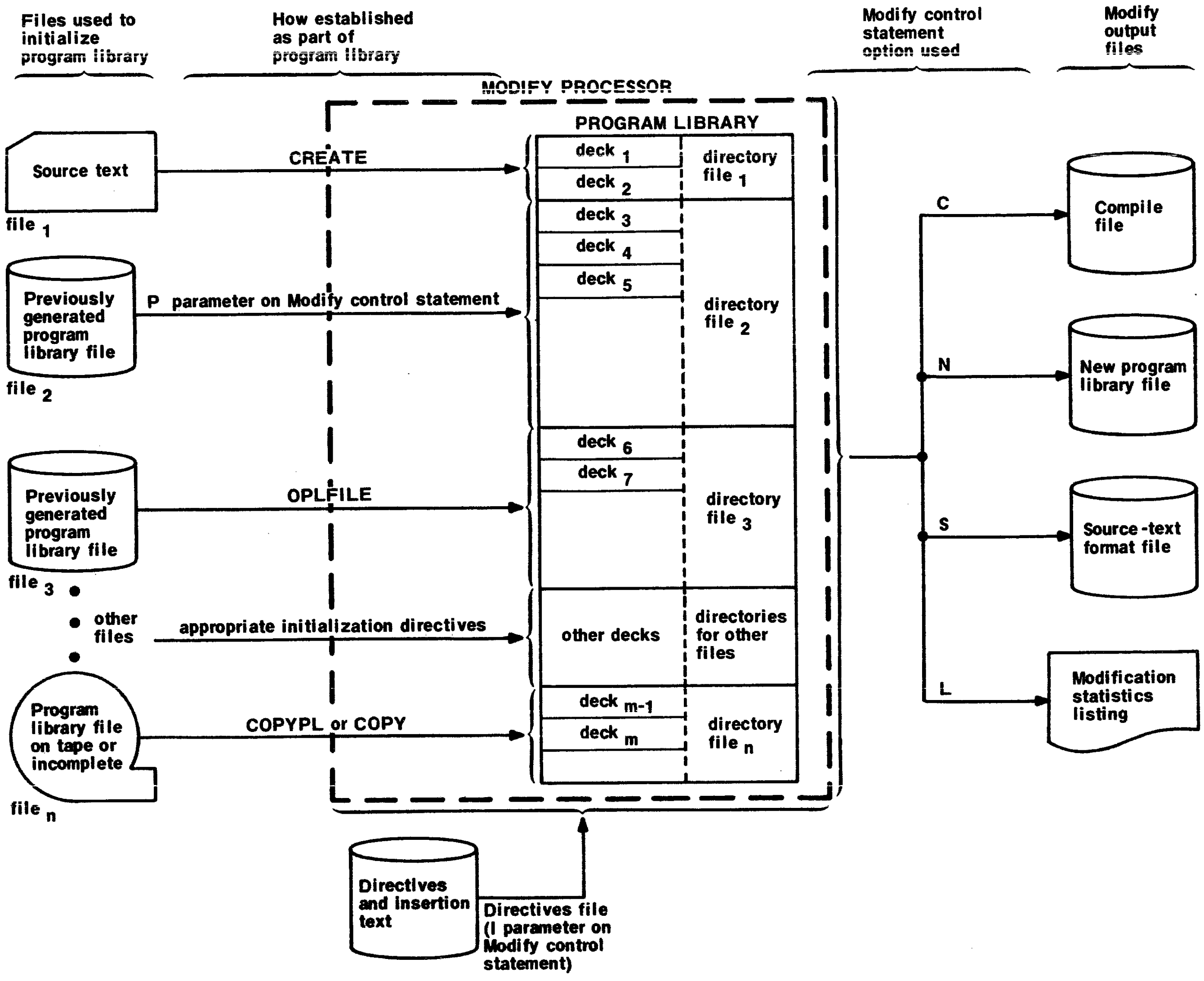

Files used to initialize program library
How established as part of program library
Modify control statement option used
Modify output files
MODIFY PROCESSOR
PROGRAM LIBRARY
Source text
file 1
CREATE
Previously generated program library file
file 2
P parameter on Modify control statement
Previously generated program library file
file 3
OPLFILE
other files
appropriate initialization directives
Program library file on tape or incomplete
file n
COPYPL or COPY
deck 1
deck 2
directory file 1
deck 3
deck 4
deck 5
directory file 2
deck 6
deck 7
directory file 3
other decks
directories for other files
deck m-1
deck m
directory file n
C
Compile file
N
New program library file
S
Source-text format file
L
Modification statistics listing
Directives and insertion text
Directives file (I parameter on Modify control statement)

Figure 1-1. Simplified Modify Organization

Compile file directives are part of the text on the program library; thus, compile file directives were either on a file used to initialize the program library, or were inserted by modification directives. An example of a compile file directive is the CALL directive.

Modify includes many other directives providing extended features. These primarily affect the operating characteristics of Modify which are described in section 7.

OUTPUT FILES

Modify produces several files as output, all of which are optional. The user specifies these files through options on the Modify control statement.

The compile file is a text file with user-specified modifications incorporated into it. It may be used as input to a language processor, directed to an output device such as a printer or card punch, or used as data for an applications program.

The new program library file contains the same updated text as the compile file, only in program library format. Thus, Modify can process this file directly on subsequent Modify runs.

Modify produces a list of text incorporated into the program library, details the status of the program library and the other files output by Modify, and notes errors and other significant events occurring during Modify execution.

The source-text output file contains updated text similar to that of the compile file. However, compile file directives on the program library have not been removed or acted upon by Modify.

MODIFY EXECUTION

Modify begins execution as a result of the operating system interpreting a Modify control statement. Modify execution then progresses in three phases:

- Initialize program library
- Read modification directives
- Incorporate changes/write output files

INITIALIZE PROGRAM LIBRARY

During this phase, Modify reads initialization directives (which must precede modification directives) from the directives file to prepare the program library. The first file to be included in the program library is the file declared on the Modify control statement (P parameter); refer to section 8. Other files declared by initialization directives are logically merged with this file to form the program library. If the initialization directive specifies that a file is in source-text format, Modify converts it to a file in program library format before merging it with the program library.

The initialization phase ends when Modify encounters the first modification directive. File manipulation directives do not terminate the initialization phase.

READ MODIFICATION DIRECTIVES

During the second phase, Modify reads the remaining directives on the directives file and stores any new text for insertion during the final phase. The time-sharing user is prompted for directives by Modify at his terminal. In batch usage, the file containing the directives is specified on the Modify control statement. This defaults to the job input file. An alternate directives file may be specified by the appropriate file manipulation directive (refer to section 5).

INCORPORATE CHANGES/WRITE OUTPUT FILES

During the final phase, Modify performs the requested changes on a deck-by-deck basis, incorporating them into the output files requested by the Modify control statement. Each inserted line is assigned a modification name, specified by a modification directive (refer to section 4), and a sequence number generated by Modify. These are used in later Modify runs to make further changes to the text. All lines having the same modification name comprise a modification set.

This phase can be initiated either by Modify interpreting an EDIT directive (refer to section 4) on the directive file, or by the presence of a Modify control statement option specifying that this phase should be initiated by Modify after it exhausts the directive file (refer to section 8).

FEATURES

Features of Modify include:

- Formatting of text files to facilitate line-by-line modification.
- Insertion, deletion, and restoration of previously deleted lines according to line sequence numbers.
- Facilities for rescinding one or more groups of changes (modification sets) previously applied to text, thereby preserving original appearance of text.
- Replacement of often-used groups of lines by one-line calls for their insertion.
- Facilities for limiting range of modifications to specified decks.
- Generation of a file in text format suitable for input to processors such as compilers and assemblers.
- Execution from either batch-origin or time-sharing jobs.
- Processing of directives from an alternate file.
- Comprehensive statistical output noting any changes effected during the run and presenting the status of the program library.
- Support of both 63- and 64-character sets.

MODIFY EXAMPLES

Examples in this manual are for illustrative purposes only. These examples are neither the most efficient nor necessarily recommended methods of using the Modify directives.

Figure 1-2 details a job submitted to local or remote batch and figure 1-3 illustrates the same job entered from a time-sharing terminal. The user need not be concerned with the meaning of directives or of parameters on the Modify control statement at this point. Instead, he should compare the structure of the two jobs.

Subsequent examples in this manual (with the exception of section 3 and section 10, Batch Job Examples) depict only jobs entered from a time-sharing terminal.

The examples pertaining to a group of directives immediately follow the discussion of those directives. Some of the files created and modified in an example have been retained and used in the succeeding example.

```
JOBMOD.
USER(USERNUM,PASSWRD,FAMILY)
CHARGE(CHARNUM,PROJNUM)
GET(MAINP)
COPYSBF(MAINP)
MODIFY(P=0,F,N)
SAVE(NPL=MAINPL)
--EOR--
*REWIND MAINP
*CREATE MAINP
--EOI--
```

Input directives for Modify statement.

End-of-information is 6/7/8/9 multiple punch in column 1.

Figure 1-2. Modify Execution from Batch

```
batch
$RFL,0.
/old,mainp
/lnh,r
DECK1
***    MAIN PROGRAM
       PROGRAM MAIN(OUTPUT)
       PRINT*,"BEGIN MAIN PROGRAM."
       CALL SUB1
       PRINT*,"END MAIN PROGRAM."
       STOP
       END
--EOR--
DECK3
***    EMPTY DECK
--EOR--
/modify,p=0,f,n,l=0
? *rewind mainp
? *create mainp
?
 MODIFICATION COMPLETE.
/save,npl=mainpl
```

After logging in, user requests batch subsystem.

User specifies (l=0) indicating that he does not wish to receive Modify output.

Input directives are requested and entered immediately following Modify statement. Null input line (carriage return only) terminates input.

Program notifies user that it has completed modification.

Figure 1-3. Modify Execution from Time-Sharing Terminal

ASCII MODE CONSIDERATIONS

Several problems may arise when using Modify from a time-sharing job while the terminal is in full ASCII character set mode. Refer to appendix A of the NOS Reference Manual, volume 1, for a description of ASCII character sets.

Directives entered interactively from the terminal, or those in an alternate directive input file, must not contain ASCII characters with escape codes; that is, directives must be entered in all uppercase characters. Modify does not recognize lowercase directives that contain escape codes.

When creating a program library, several precautions should be taken. While a source file can contain full ASCII characters, all deck names and compile file directives must be in full uppercase (no escape codes). Care should also be taken when entering source lines in full ASCII mode. Since each character may actually occupy 12 bits (escape code and character), what appears to be a line width of 75 characters, for example, may actually be 150 characters. Modify does not allow line widths greater than 100 6-bit characters.

DIRECTIVE FORMAT 2

Directives allow the user to create libraries and extensively control and direct the correction and modification process. File initialization directives identify old program libraries and source decks to be placed on the new program library. Modification directives identify the text to be inserted, set parameters of the modification process, and inform Modify of insertions, deletions, and other corrections. File manipulation directives allow user control of the input files. Compile file directives can be in source decks originally or can be inserted during a Modify run. These directives are manipulated much like source lines during the creation, updating, and correction phases but are recognized when the compile file is written.

A directive has the following format.

*dirname $p_1, p_2, \ldots, p_n$

*	The prefix character is in column 1. It is initially defined by Modify as an asterisk, but may be changed with PREFIX and PREFIXC directives. In this manual, the asterisk is used as the prefix character.
dirname	The directive name starts in column 2. It is terminated by one or more blanks or a separator (for example, a comma).
p_i	Optional directive parameters. Numeric parameters are decimal.

The directive name and parameters are separated by any character that has a display code value of 55_8 or greater; that is (assuming 64-character set), a character other than:

: A-Z 0-9 + - * / ( ) $ =

Some directives require specific separators. No embedded blanks are permitted within a parameter. However, any number of blanks can be between the directive name and the first parameter or between two parameters, provided the entire directive does not exceed 72 columns.

LINE IDENTIFICATION

The modification directives DELETE, INSERT, and RESTORE, and the file manipulation READPL directive require line identifiers. These identifiers can be in either the complete or abbreviated form.

The complete format of a line identifier is:

modname. number

modname.	1- to 7-character name of a modification set or deck. A period terminates the modification name.
number	Decimal ordinal (1 to 262143) of the line within the correction set or deck. Any character other than 0 through 9 terminates the sequence number.

The abbreviated form of a line identifier is:

number

When only the number is used for line identification (modification name is omitted), Modify uses the name from the MODNAME directive or the most recent DECK directive.

INITIALIZATION DIRECTIVES

Modify initialization directives are placed on the directive file and precede all directives other than file manipulation directives. They are:

CREATE	Converts source decks to program library format for modification.
OPLFILE	Declares additional program library files as input.
COPY	Copies one or more records from named file to old program library.
COPYPL	Copies one or more records from named file to an internal scratch file which is logically merged with program library.
WIDTH	Defines the number of columns preceding the sequencing information on the compile and source files; can occur anywhere in directives file.
NOSEQ	Specifies no sequence information on compile file.

CREATE, OPLFILE, COPY, and COPYPL are illegal after the first use of modification directives. WIDTH and NOSEQ can be processed as compile file directives.

When a second deck of the same name is introduced during initialization, the second deck takes precedence. In directory list output, the name of a replaced deck is enclosed in parentheses.

PREPARING THE SOURCE FILE

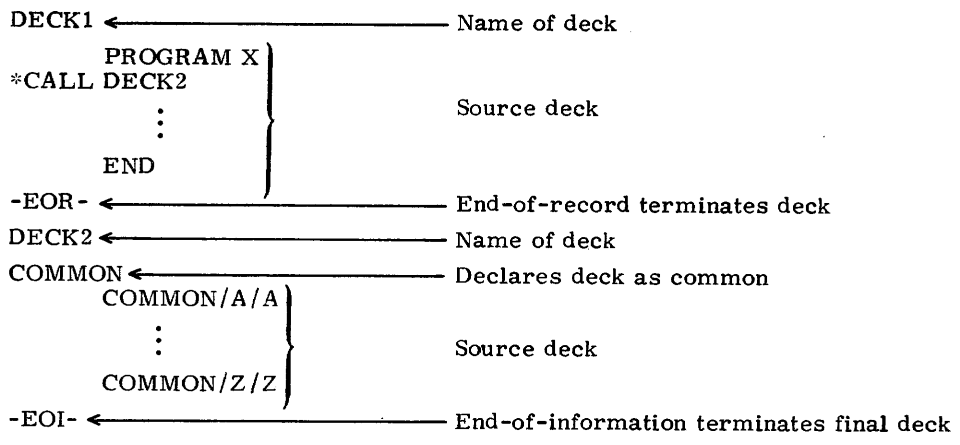
Before Modify can create a program library, the user must prepare the source file by assigning a deck name to each record of the source file and by identifying those decks that are to be common decks. The deck name must be the first line of the source deck. A 1- to 7-character deck name begins in column 1. Legal characters are:

A through Z 0 through 9 + - * / ( ) $ =

The second line of the source deck can identify the deck as common. To do so, it must contain the word COMMON in columns 1 through 6. An end-of-record terminates the deck. A set of decks is terminated by an end-of-file (6/7/9 multiple punch in column 1 for batch origin jobs) or end-of-information.

Figure 3-1 illustrates a typical Modify source deck.

Usually a deckname (optionally followed by a COMMON) precedes each program or subprogram. However, more than one subprogram may be included in a deck as is indicated in figure 3-2. A user might group two programs if modification of one requires reassembly or recompilation of both programs.



```
DECK1
       PROGRAM X
*CALL DECK2
           :
       END
-EOR-
DECK2
COMMON
       COMMON/A/A
           :
       COMMON/Z/Z
-EOI-
```

Name of deck
Source deck
End-of-record terminates deck
Name of deck
Declares deck as common
Source deck
End-of-information terminates final deck

Figure 3-1. Modify Source Deck

```
FIRST
         IDENT      FIRST      } Program one   }
           :                                   }
         END                                   }  First deck
         IDENT      SECOND     } Program two   }
           :                                   }
         END                                   }
-EOR-

FDATA
COMMON
         DATA       0                          }
         DATA       0                          }  Second deck
         DATA       0                          }
-EOI-
```

Figure 3-2. Deck with Several Programs

CREATE – CREATE PROGRAM LIBRARY

When Modify encounters this directive, it writes the contents of the named file from its current position until it encounters an end-of-file onto a scratch file in program library format with a directory. CREATE provides a means of initially creating a program library for subsequent modification, for adding decks to the program library, or for replacing decks on the program library. †

Format:

```
*CREATE file
```

file	Name of file containing one or more source decks. A format error occurs if the name of the file is omitted from the directive. This file must be local to the user's job.

OPLFILE – DECLARE ADDITIONAL OPL FILES

The OPLFILE directive specifies additional files, already in program library format, that Modify logically merges with any existing program library. The existing library is made up of the old program library declared on the Modify control statement (P parameter) and/or other program library files established internally by CREATE or COPYPL. †

The total number of files declared by OPLFILE directives cannot exceed 20 files. Additional files are ignored with the message:

```
TOO MANY OPL FILES.
```

Format:

*OPLFILE $file_1, file_2, \ldots, file_n$

$file_i$	Names of one or more files in program library format to be merged logically with the existing program library.

COPYPL – COPY PROGRAM LIBRARY TO SCRATCH

The COPYPL directive copies records (decks) already in program library format to an internal scratch file which Modify logically merges with any existing program library. † Modify builds a directory for this file as it is copied, ignoring any existing directory on the file from which the copy is made. All or part of the file can be copied. The file may reside on either mass storage or magnetic tape. Modify ignores all records on the file which are not in program library format.

Format:

```
*COPYPL file,deckname
```

file	Name of file containing decks in program library format, with or without directory, and with or without other records in nonprogram library format.
deckname	Optional; name of last deck (record) to be copied. If deckname is omitted from directive, or is not found on file, Modify copies all decks from the file starting at the current file position.

† If the resulting program library contains two or more decks having the same name, the last one introduced to Modify takes precedence; that is, the previous deck is logically replaced.

COPY – COPY PROGRAM LIBRARY TO OPL

The COPY directive performs the same functions as the COPYPL directive, with the following differences:

- The records (decks) are copied to the old program library file declared on Modify control statement (P parameter). If P=0 is specified on the Modify control statement, the use of the COPY directive is not allowed.
- Modify performs an EVICT on the old program library file before the copy takes place. Hence, this file (if it already exists) should not contain any useful information. See the NOS Reference Manual, volume 1, for a description of EVICT.
- COPY can be preceded only by file manipulation directives.
- Only one COPY directive is allowed for each Modify execution.

COPY is useful when copying all or part of a program library residing on magnetic tape to a mass storage device, since the resulting program library file may be saved as a permanent file without having Modify create a new program library. See the NOS Reference Manual, volume 1, for a description of permanent file control statements.

Format:

```
*COPY file,deckname
```

file	Name of file containing decks in program library format, with or without directory, and with or without other records in nonprogram library format.
deckname	Optional; name of last deck (record) to be copied. If deckname is omitted from directive, or is not found on file, Modify copies all decks from the file, starting at the current file position.

WIDTH – SET LINE WIDTH ON COMPILE FILE

The WIDTH directive allows the user to set the width of lines prior to the modify program library and write compile phase. The last (or only) WIDTH directive encountered on the directives file is used during the compile phase until a compile file WIDTH is encountered. If text is being inserted, the WIDTH directive is left in the text stream and is later processed as a compile file directive. WIDTH can occur anywhere in the directive file.

Format:

```
*WIDTH n
```

n	Number of columns preceding sequence information on compile file and source file. Modify allows a maximum of 100 columns. During initialization of Modify, width is preset to 72.

NOSEQ – NO SEQUENCE INFORMATION

The NOSEQ directive allows the user to set the no sequence flag prior to the write compile phase. When no sequencing is requested, Modify does not include sequence information on the compile file. A SEQ directive encountered during the write compile phase clears the no sequence flag. If text is being inserted, the NOSEQ directive is inserted into the text stream and processed as a compile file directive.

FORMAT:

```
*NOSEQ
```

INITIALIZATION DIRECTIVES EXAMPLES

Figures 3-3 and 3-4 illustrate the creation of program libraries and the use of several initialization directives. Figure 3-3 is a detailed terminal session; figure 3-4 represents the same job formatted for batch input. The user can submit the batch origin job to obtain and examine output produced by Modify and FORTRAN

```
batch,45000
$RFL,45000.
```

User selects batch subsystem, requesting 45000 words of CM.

```
/old,mainp
/lnh,r
DECK1
***    MAIN PROGRAM
       PROGRAM MAIN(OUTPUT)
       PRINT*,"BEGIN MAIN PROGRAM."
       CALL SUB1
       PRINT*,"END MAIN PROGRAM."
       STOP
       END
--EOR--
DECK3
***    EMPTY DECK
--EOR--
```

Listing of source file, showing end-of-record marks, to be used to create program library. Notice required deck names.

```
/modify,p=0,l=0,f,n=mainpl,c=0
? *create mainp
?
 MODIFICATION COMPLETE.
```

Modify statement to create program library with name MAINPL. MAINPL is the result of converting the source text file MAINP to program library format.

```
/catalog,mainpl,r
           CATALOG OF MAINPL         FILE      1
    REC    NAME      TYPE          LENGTH    CKSUM      DATE

      1    DECK1     OPL   (64)      30       4476    76/01/22.
      2    DECK3     OPL   (64)       4       1725    76/01/22.
      3    OPL       OPLD             5       1310    76/01/22.

      4    * EOF *        SUM =      41
1
 CATALOG COMPLETE.
```

The catalog utility is a convenient means of determining the decks and their types that were written on the program library. Refer to the NOS Reference Manual, volume 1, for information on the CATALOG control statement.

```
/save,mainpl
/get,sub1
/copycf,sub1
DECK2
***    SUBROUTINE 1
       SUBROUTINE SUB1
       PRINT*,"ENTER SUBROUTINE 1."
       CALL SUB2
       PRINT*,"EXIT SUBROUTINE 1."
       RETURN
       END
 END OF INFORMATION ENCOUNTERED.
```

Another source deck that the user wishes to maintain on a separate program library.

```
/rewind,sub1
$REWIND,SUB1.
/modify,p=0,l=0,f,n=altpl1,c=0
? *create sub1
?
 MODIFICATION COMPLETE.
```

Modify statement to create program library ALTPL1.

```
/catalog,altpl1,r
           CATALOG OF ALTPL1         FILE      1
    REC    NAME      TYPE          LENGTH    CKSUM      DATE

      1    DECK2     OPL   (64)      30       5013    76/01/22.
      2    OPL       OPLD             3       2117    76/01/22.

      3    * EOF *        SUM =      33
1
 CATALOG COMPLETE.
/get,altpl2
```

User obtains alternate program library that he had created at an earlier session.

```
/catalog,altpl2,r
           CATALOG OF ALTPL2         FILE      1
    REC    NAME      TYPE          LENGTH    CKSUM      DATE

      1    DECK3     OPL   (64)      25       0100    76/01/21.
      2    OPL       OPLD             3       2517    76/01/21.

      3    * EOF *        SUM =      30
```

Figure 3-3. Initialization Directive Examples (Sheet 1 of 2)

```
1
 CATALOG COMPLETE.
/rename,opl=mainpl
$RENAME,OPL=MAINPL.
```

Program library MAINPL is renamed OPL. In this manner, the P parameter is not needed on the Modify statement.

```
/modify,f,l=0,n=mainpl
? *oplfile altpl1
? *copypl altpl2,deck3
?
 MODIFICATION COMPLETE.
```

Modify run to merge OPL with program library ALTPL1 and then use ALTPL2 to replace deck DECK3 on OPL. The compile output of MAINPL is written on the default file COMPILE.

```
/catalog,mainpl,r
          CATALOG OF MAINPL           FILE      1
    REC   NAME      TYPE         LENGTH      CKSUM      DATE

     1    DECK1     OPL  (64)        30       4476   76/01/22.
     2    DECK3     OPL  (64)        25       0100   76/01/21.
     3    DECK2     OPL  (64)        30       5013   76/01/22.
     4    OPL       OPLD              7       5011   76/01/22.

     5    * EOF *        SUM =      114
1
 CATALOG COMPLETE.
/replace,mainpl
/copycf,compile
***   MAIN PROGRAM                                                DECK1      1
      PROGRAM MAIN(OUTPUT)                                        DECK1      2
      PRINT*,"BEGIN MAIN PROGRAM."                                DECK1      3
      CALL SUB1                                                   DECK1      4
      PRINT*,"END MAIN PROGRAM."                                  DECK1      5
      STOP                                                        DECK1      6
      END                                                         DECK1      7
***   SUBROUTINE 2                                                DECK3      1
      SUBROUTINE SUB2                                             DECK3      2
      PRINT*,"ENTER SUBROUTINE 2."                                DECK3      3
      PRINT*,"EXIT SUBROUTINE 2."                                 DECK3      4
      RETURN                                                      DECK3      5
      END                                                         DECK3      6
***   SUBROUTINE 1                                                DECK2      1
      SUBROUTINE SUB1                                             DECK2      2
      PRINT*,"ENTER SUBROUTINE 1."                                DECK2      3
      CALL SUB2                                                   DECK2      4
      PRINT*,"EXIT SUBROUTINE 1."                                 DECK2      5
      RETURN                                                      DECK2      6
      END                                                         DECK2      7
 END OF INFORMATION ENCOUNTERED.
```

Listing of compile file created by Modify. Notice sequencing information.

```
/rewind,compile
$REWIND,COMPILE.
/ftn,i=compile,l=0
      .145 CP SECONDS COMPILATION TIME
```

Compile file is used as input to FORTRAN Extended compiler.

```
/lgo
 BEGIN MAIN PROGRAM.
 ENTER SUBROUTINE 1.
 ENTER SUBROUTINE 2.
 EXIT SUBROUTINE 2.
 EXIT SUBROUTINE 1.
 END MAIN PROGRAM.
      .006 CP SECONDS EXECUTION TIME
```

Execution of FORTRAN program.

Figure 3-3. Initialization Directive Examples (Sheet 2 of 2)

```
JOB1.
USER(USERNUM, PASSWRD, FAMILY)
CHARGE(CHARNUM, PROJNUM)
OLD(MAINP)
COPYSBF(MAINP)
MODIFY(P=0, F, N=MAINPL, C=0)
CATALOG(MAINPL, R)
SAVE(MAINPL)
GET(SUB1)
COPYSBF(SUB1)
REWIND(SUB1)
MODIFY(P=0, F, N=ALTPL1, C=0)
CATALOG(ALTPL1, R)
GET(ALTPL2)
CATALOG(ALTPL2,R)
RENAME(OPL=MAINPL)
MODIFY(F, N=MAINPL)
CATALOG(MAINPL, R)
REPLACE(MAINPL)
COPYSBF(COMPILE)
REWIND(COMPILE)
FTN(I=COMPILE)
LGO.
-EOR-
*CREATE MAINP
-EOR-
*CREATE SUB1
-EOR-
*OPLFILE ALTPL1
*COPYPL ALTPL2, DECK3
-EOI-
```

Figure 3-4. Batch Job Creating Program Libraries

MODIFICATION DIRECTIVES 4

Modification directives and their accompanying insertion lines are placed on the directives file after the last initialization directive. The first occurrence of a modification directive terminates the initialization phase.

The following modification directives assign a modification name to the corrections being made, identify the deck being modified, and give the modification set name to be used when the short form of the line identifiers is used.

IDENT	Specifies modification name to be assigned to new modification set.
DECK	Identifies deck to be altered.
MODNAME	Identifies modification set within deck to be modified when short form of line identifier is used and the modification name is different from that used in the last IDENT or MODNAME directive.

The following modification directives are used for inserting and deleting lines.

DELETE or D	Deactivates lines and optionally inserts lines in their place.
RESTORE	Reactivates lines and optionally inserts text after them.
INSERT or I	Inserts lines after specified line.

These directives indicate to Modify that:

- New lines are to be inserted into the deck and sequenced according to the correct modification set identifier.
- Old lines are to be deleted.

While inserting, Modify interprets file manipulation directives (for example, READPL changes the source of insertion lines but does not terminate insertion). Insertion terminates when Modify next encounters another modification directive or end-of-record.

Insertion lines can include compile file directives. These directives are not interpreted but are inserted as if they were text; the prefix character written on the program library is that specified on the directive.

Other directives described in this section include:

YANK	Deactivate modification set.
UNYANK	Reactivate modification set.
PURDECK	Remove all lines in a deck.
IGNORE	Ignore subsequent modifications to a named deck.
EDIT	Modify and write named deck to files specified on Modify control statement.

IDENT – IDENTIFY NEW MODIFICATION SET

The IDENT directive assigns a name to a modification set. Modify does not require any IDENT directive; however, this practice is discouraged. If the directives file does not contain an IDENT directive, the system uses ******* as the modname. This default name should not be used when a new program library is made. The user can use one IDENT for several decks or can use several IDENT directives for one deck. There is no restriction on the placement of IDENT within the modification directives input file.

Format:

```
*IDENT modname
```

modname — 1- to 7-character modification name to be assigned to this modification set. This name causes a new entry in the modification table for each deck for which the modification set contains a DECK directive until the next IDENT. Each line inserted by this set, and each line for which the status is changed, receive a modification history byte that indexes this modname.

Normally, sequencing of new lines begins with one for each deck using the modification name. However, when the UPDATE directive is used, sequence numbers continue from deck to deck.

Omitting modname causes a format error. If modname duplicates a name previously used for modifying a deck, Modify generates the message

DUPLICATE MODIFIER NAME.

A duplicate modname or encountering modifications that refer to this modification name prior to this *IDENT modname cause a fatal error accompanied by the message MOD(S) TO MOD BEFORE THIS IDENT CARD.

DECK — IDENTIFY DECK TO BE MODIFIED

The DECK directive identifies the name of the deck to which subsequent modifications apply.

Format:

```
*DECK deckname
```

deckname	Name of deck for which modifications following this line apply. The modifications for this deck terminate with the next DECK directive. A DECK directive is required for each deck being modified. If the deckname is not found, Modify flags the error with the message UNKNOWN DECK. Omitting the deckname causes a format error.

MODNAME — IDENTIFY MODIFICATION SET TO BE MODIFIED

By using the MODNAME directive, the user indicates that subsequent line identifiers for which a modification name is omitted apply to modification set modname previously applied to the deck. Subsequent directives need only the sequence number for the modification set. The system assumes that the line is in set modname of the deck being modified.

A MODNAME directive is effective only to the next deck or MODNAME directive. The hierarchy for line identifiers is such that if the MODNAME directive is used and the user wishes to return to use of the deckname as the assumed line identifier, he must restore the deckname by use of another MODNAME directive or use the long form of the line identifier, specifying the deck name. A MODNAME directive does not terminate an insertion if it is encountered in text being inserted.

Format:

```
*MODNAME modname
```

modname	Name of modification set previously applied to the deck. A line identifier that does not specify a modname is assumed to apply to this modification set. The modname remains in effect until another MODNAME or DECK directive is encountered.

DELETE — DELETE LINES

With the DELETE or D directive, the user deactivates a line or block of lines and optionally replaces it with insertion lines following the DELETE directive. The next modification directive (or EOR) terminates insertion. File manipulation directives are interpreted and may change the source of insertion lines but do not terminate insertion and are not inserted into the deck. Insertion lines can include compile file directives.

A deactivated line remains on the library and retains its sequencing, but is not included in compile decks or source decks.

Formats:

*DELETE c	or	*D c
*DELETE c_1, c_2	or	*D c_1, c_2

c	Line identifier for single line to be deleted.
c_1, c_2	Line identifiers of first and last lines in sequence of lines to be deleted. c_1 must occur before c_2 on the library. Any lines in the sequence that are already inactive are not affected by the DELETE.

RESTORE — REACTIVATE LINES

With the RESTORE directive, a user reactivates a line or block of lines previously deactivated through a delete or yank and optionally inserts additional lines after the restored line or block of lines. The lines to be inserted immediately follow the RESTORE directive. The next modification directive (or EOR) terminates insertion. File manipulation directives are interpreted (and may change the source of insertion lines) but do not terminate insertion. They are not inserted into the deck. Insertion lines can include compile file directives.

Formats:

*RESTORE c

*RESTORE c_1, c_2

c	Line identifier of single line to be restored.
c_1, c_2	Line identifiers of first and last lines in sequence of lines to be restored. Any lines in the sequence that are already active are not affected by the RESTORE. c_1 must occur before c_2 on the library.

INSERT — INSERT LINES

To insert new lines in the program library, use the INSERT directive. The line to be inserted immediately follows the INSERT or I directive on the directives file. The next modification directive (or EOR) terminates insertion. File manipulation directives are interpreted (and may change the source for insertion lines) but do not terminate insertion. They are not inserted into the deck. Insertion lines can include compile file directives.

Formats:

*INSERT c or *I c

c	Identifies line after which new lines will be inserted.

YANK – REMOVE EFFECTS OF MODIFICATION SET

The YANK directive is used to deactivate a modification set. Modify searches the edited decks for all lines affected by the named modification set. If a line was activated by the modification set, Modify deactivates it. If a line was deactivated by the modification set, Modify reactivates it. Thus, Modify generates a new modification history byte for every line that changed status as a result of the YANK and effectively restores the edited decks to the status they had prior to modification modname or all modifications subsequent to modname.

For the first format, only the one modification set is yanked. For the second format, Modify yanks all modification sets applied after modname, provided modname appears on the edited decks. YANK or UNYANK directives contained in the yanked modification set are not rescinded.

YANK affects only those decks that are edited through the EDIT directive or the F or U options on the Modify control statement. In this way, the YANK directive can be selective.

Formats:

*YANK modname

YANK modname,

modname	Name of modification set previously applied to decks in the library. Omitting modname produces a format error. If Modify fails to find the modname in the modification table for the library, it issues an error.

UNYANK – RESCIND ONE OR MORE YANK DIRECTIVES

With the UNYANK directive, the user can rescind previous YANK directives. For the first format, only the one modification set is rescinded. For the second format, Modify rescinds all of the yanked modification sets, starting with modname, provided modname appears on the edited decks.

Formats:

*UNYANK modname

UNYANK modname,

modname	Name of only modification set to be rescinded or name of first of two or more modification sets to be rescinded for the library. Omitting modname results in a format error.

PURDECK – PURGE DECK

A PURDECK directive causes the permanent removal of a deck or group of decks from the program library. Every line in a deck is purged, regardless of the modification set it belongs to. A deck name purged as a result of PURDECK can be reused as either a deck name or a modification name.

A PURDECK directive can be any place in the directives input. It terminates any previous correction set. Therefore, INSERT, DELETE, and RESTORE cannot follow a PURDECK directive but must come after an IDENT directive. Purging cannot be rescinded.

Format one:

*PURDECK $\text{dname}_1, \text{dname}_2, \ldots, \text{dname}_n$

dname_i	Deck names for decks to be purged.

Format two:

*PURDECK $\text{dname}_a.\text{dname}_b$

The deck named dname_a and all decks up to and including dname_b listed in the deck list are purged.

IGNORE – IGNORE DECK MODIFICATIONS

An IGNORE directive causes any further modification directives for the designated deck to be ignored. Modify skips modification directives other than IDENT, EDIT, and DECK. When one of these directives is encountered, Modify processes it and resumes processing the input stream. Any modification directives for the decks that precede the IGNORE directive are processed normally. The EDIT deck name(s) encountered after an IGNORE directive are checked against the current ignore list. Any EDIT deck names are deleted. If an ignored deck is encountered in the EDIT directive form $\text{deckname}_a.\text{deckname}_b$, the directive is flagged and is considered as having a modification error. The following message is issued.

FORMAT ERROR IN DIRECTIVE

Format:

*IGNORE dname

EDIT – EDIT DECKS

Editing is a process of modifying a deck, if modifications are encountered during the modification phase, and writing the deck on the compile file, new program library, and source file.

The three possible modes of editing are selective, full, and update. The modes are selected through Modify control statement options.

Format:

*EDIT $p_1, p_2, \ldots, p_n$

p_i A deckname or range of decknames in one of the following forms:

deckname

$\text{deckname}_a.\text{deckname}_b$

The first form requests that Modify edit a deck on the program library; the second form requests a range of decks starting with deckname_a and ending with deckname_b. If decknames are in the wrong sequence, Modify issues the error message:

NAMES SEPARATED BY *.* IN WRONG ORDER.

If Modify fails to find one of the decks, it issues the message:

UNKNOWN DECK - deckname.

SELECTIVE EDIT MODE

When selective editing is desired (neither F nor U selected on the Modify control statement), Modify edits only the decks specified on EDIT directives. EDIT directives cause a deck to be written regardless of whether it was corrected or not. Decks are edited in the sequence encountered on EDIT directives unless an UPDATE directive specifies otherwise. Modifications encountered during the modification phase are not incorporated in a deck if the deck is not specified on an EDIT directive. In particular, calling a common deck from within a deck being edited does not automatically result in the common deck being edited.

If decks are being replaced or new decks are added, the new decks are placed at the end of the library. Thus, a deck formerly included in an EDIT sequence will no longer lie within the sequence.

FULL EDIT MODE

When a full edit is requested (F selected on Modify control statement), Modify ignores EDIT directives. It writes all decks in the sequence encountered on the program library. This option provides for creating a complete new program library. Because the same decks that are written on the new program library are also written on the compile file, a user wishing to obtain only a partial set of decks on the compile file must request separate runs of Modify — one run for creating the new program library and one run for creating the compile file.

UPDATE EDIT MODE

If the U option is selected on the Modify control statement, Modify edits only those decks mentioned on DECK directives and ignores the EDIT directives. Thus, only decks being updated by the Modify run are written on the compile file. This mode is not normally requested when a new program library or source file is desired.

MODIFICATION DIRECTIVE EXAMPLES

Figure 4-1 is a detailed example of some of the modification directives presented in this section.

```
batch,45000
$RFL,45000.
/get,opl=mainpl
/modify,f,l=0,n=mainpl
? *ident mod1<-------------------  This modification set is given name MOD1.
? *deck deck3
? *delete deck3.1
? ***    subroutine 2, deck deck3.
? *deck deck2
? *d 1<--------------------------{ Refer to listing of compile file in figure 3-3
? ***    subroutine 1, deck deck2. { to reference line sequence numbers.
? *insert 3
? *      call subroutine sub2
? *      in deck deck2.
? *delete 7
? ***    end deck2.
? *deck deck1
? *d 1
? ***    main program, deck deck1.
?
 MODIFICATION COMPLETE.
```

Figure 4-1. Modification Directive Examples (Sheet 1 of 3)

```
/copycf,compile
***   MAIN PROGRAM, DECK DECK1.                                      MOD1       1
      PROGRAM MAIN(OUTPUT)                                           DECK1      2
      PRINT*,"BEGIN MAIN PROGRAM."                                   DECK1      3
      CALL SUB1                                                      DECK1      4
      PRINT*,"END MAIN PROGRAM."                                     DECK1      5
      STOP                               Listing of compile          DECK1      6
      END                                file created by             DECK1      7
***   SUBROUTINE 2, DECK DECK3.          Modify.                     MOD1       1
      SUBROUTINE SUB2                                                DECK3      2
      PRINT*,"ENTER SUBROUTINE 2."                                   DECK3      3
      PRINT*,"EXIT SUBROUTINE 2."                                    DECK3      4
      RETURN                                                         DECK3      5
      END                                                            DECK3      6
***   SUBROUTINE 1, DECK DECK2.                                      MOD1       1
      SUBROUTINE SUB1                                                DECK2      2
      PRINT*,"ENTER SUBROUTINE 1."                                   DECK2      3
*     CALL SUBROUTINE SUB2                                           MOD1       2
*     IN DECK DECK2.                                                 MOD1       3
      CALL SUB2                                                      DECK2      4
      PRINT*,"EXIT SUBROUTINE 1."                                    DECK2      5
      RETURN                   {Note that user inadvertently deleted END  DECK2 6
***   END DECK2.<--------------{statement.                           MOD1       4
 END OF INFORMATION ENCOUNTERED.
/modify,l=0,p=mainpl,n=mpl1,c=com1
? *ident mod2            <--
? *deck deck2                 \
? *restore 7                   {Modification run to restore deleted line, and
? *d mod1.3                    {delete line MOD1.3.
? *edit deck2
?
 MODIFICATION COMPLETE.
/copycf,com1                           Note that compile
***   SUBROUTINE 1, DECK DECK2.        file contains only            MOD1       1
      SUBROUTINE SUB1                  edited deck(s).               DECK2      2
      PRINT*,"ENTER SUBROUTINE 1."                                   DECK2      3
*     CALL SUBROUTINE SUB2                                           MOD1       2
      CALL SUB2<------------------Note deleted line.                 DECK2      4
      PRINT*,"EXIT SUBROUTINE 1."                                    DECK2      5
      RETURN                                                         DECK2      6
      END<------------------------END statement restored.            DECK2      7
***   END DECK2.                                                     MOD1       4
 END OF INFORMATION ENCOUNTERED.
/modify,l=0,p=mpl1,n=mpl2,c=com2
? *ident mod3
? *deck deck2
? *modname mod1
? *restore 3<-------------------Line deleted in previous Modify run is restored.
? *edit deck2
?
 MODIFICATION COMPLETE.
/copycf,com2
***   SUBROUTINE 1, DECK DECK2.                                      MOD1       1
      SUBROUTINE SUB1                                                DECK2      2
      PRINT*,"ENTER SUBROUTINE 1."                                   DECK2      3
*     CALL SUBROUTINE SUB2                                           MOD1       2
*     IN DECK DECK2.<-------------Restored line.                     MOD1       3
      CALL SUB2                                                      DECK2      4
      PRINT*,"EXIT SUBROUTINE 1."                                    DECK2      5
      RETURN                                                         DECK2      6
      END                                                            DECK2      7
***   END DECK2.                                                     MOD1       4
 END OF INFORMATION ENCOUNTERED.
/rewind,mainpl,mpl2                 {The LIBEDIT utility provides a convenient
$REWIND,MAINPL,MPL2.                {means of replacing or adding records on a file.
/libedit,i=0,p=mainpl,l=0,b=mpl2,c<-{Refer to the NOS Reference Manual, volume 1,
 EDITING COMPLETE.                  {for a description of the LIBEDIT utility.
```

Figure 4-1. Modification Directive Examples (Sheet 2 of 3)

```
/catalog,mainpl,r
          CATALOG OF MAINPL          FILE      1
    REC   NAME      TYPE          LENGTH     CKSUM      DATE

      1   DECK1     OPL   (64)         37     7732    76/01/22.
           MOD1

      2   DECK3     OPL   (64)         34     3117    76/01/21.
           MOD1

      3   DECK2     OPL   (64)         55     5026    76/01/22.
           MOD1      MOD2      MOD3

      4   OPL       OPLD               11     7477   76/01/22.

      5   * EOF *       SUM =         161
1
 CATALOG COMPLETE.
/replace,mainpl
/modify,l=0,p=mainpl,c=com3,n=nplx
? *ident modx
? *deck deck2
? *yank mod3
? *edit deck2
?
 MODIFICATION COMPLETE.
```

Temporary modification run to deactivate modification set MOD3 and selectively edit deck DECK2.

```
/catalog,nplx,r
          CATALOG OF NPLX            FILE      1
    REC   NAME      TYPE          LENGTH     CKSUM      DATE

      1   DECK2     OPL   (64)         55     6626    76/01/22.
           MOD1      MOD2     (MOD3     )

      2   OPL       OPLD                3     2117    76/01/22.

      3   * EOF *       SUM =          66
1
 CATALOG COMPLETE.
```

Note that yanked modification set is enclosed in parentheses.

```
/copycf,com3
***    SUBROUTINE 1, DECK DECK2.                                  MOD1        1
       SUBROUTINE SUB1                                            DECK2       2
       PRINT*,"ENTER SUBROUTINE 1."                               DECK2       3
*      CALL SUBROUTINE SUB2                                       MOD1        2
       CALL SUB2                                                  DECK2       4
       PRINT*,"EXIT SUBROUTINE 1."                                DECK2       5
       RETURN                                                     DECK2       6
       END                                                        DECK2       7
***    END DECK2.                                                 MOD1        4
 END OF INFORMATION ENCOUNTERED.
```

Compare with previous compile file of DECK2.

Figure 4-1. Modification Directive Examples (Sheet 3 of 3)

FILE MANIPULATION DIRECTIVES

File manipulation directives allow user control over files during the initialization and modification phases. Two of these directives, READ and READPL, may be used to change the source of directives and insertion text from the directives file to an alternate file. While an insertion is in progress, a file change does not terminate insertion. Insertion continues until Modify reads the next modification directive. File manipulation directives are illegal when Modify is reading from an alternate file and result in the following message:

OPERATION ILLEGAL FROM ALTERNATE FILE INPUT.

The file manipulation directives include:

READ	Read record or group of records from specified file.
READPL	Read deck or portion of deck from program library.
BKSP	Backspace specified number of records on file.
SKIP	Skip forward specified number of records on file.
SKIPR	Skip forward past the specified record on file.
REWIND	Rewind named files.
RETURN	Return named files to system.

These operations cannot be performed on the following reserved files (or their equivalents).

INPUT	Source of directives
OUTPUT	Statistics output
COMPILE	Compile
SOURCE	Source output
OPL	Old program library
NPL	New program library
SCR1	Scratch file 1
SCR2	Scratch file 2
SCR3	Scratch file 3

These file names are reserved only through their respective Modify control statement options. For example, if the S option is not specified, the file SOURCE is not reserved and the user can use file manipulation directives specifying a file of that name. However, file names SCR1, SCR2, and SCR3 should not be used.

READ — READ ALTERNATE DIRECTIVES FILE

The READ directive causes Modify to temporarily stop reading the directives file and begin reading directives and insertion text from the specified record on the named file or current position if deckname is omitted (or *). Unless * is the deckname field, Modify reads from the alternate directives file until it encounters an end-of-record and then resumes with the next directive on the primary directives file.

If Modify is unable to find the named record, it issues the message

RECORD NOT FOUND.

Formats:

```
*READ file
*READ file,dname
*READ file,*
```

file	Name of file containing insertion text and/or directives.
dname	Optional; if dname is specified, text must be in source file format; that is, the first word of record is the name of the record. Modify discards the name before processing any text.
*	Optional; if specified, Modify processes all records on the file up to an end-of-file or a zero-length record. These records must be in source file format.

READPL — READ PROGRAM LIBRARY

The READPL directive causes Modify to temporarily stop reading the directives file and begin reading directives and insertion text from the specified Modify deck. It allows a user to insert text from one deck on the program library into another program, or to move text within a program.

Formats:

*READPL dname

*READPL dname, c_1, c_2

dname	Name of deck on old program library.
c_1, c_2	Portion of deck to be read; must be more than one line.

Modify inserts all the active lines in the deck or portion of the deck specified by the READPL. If c_1, c_2 are omitted, it reads the entire deck before returning to the directive file.

NOTE

During processing of the READPL directive, Modify does not perform any modifications to the text in the deck it is reading. If the user wishes the new text to be modified, he must make the corrections to the deck into which the text is being inserted; that is, the text is taken from the deck exactly as it is on the program library.

BKSP — BACKSPACE FILE

The BKSP directive repositions the named file one or more logical records in the reverse direction. It does not backspace beyond the beginning-of-information.

Formats:

```
*BKSP file
*BKSP file,n
```

file	Name of file to be positioned.
n	Number of records to be skipped in the reverse direction. If n is omitted, Modify backspaces one record.

SKIP — SKIP FORWARD ON FILE

The SKIP directive repositions the named file forward one or more logical records. If an end-of-information is encountered before the requested number of records has been skipped, the file is positioned at the end-of-information.

Formats:

```
*SKIP file
*SKIP file,n
```

file	Name of file to be positioned.
n	Number of records to be skipped in the forward direction. If n is omitted, Modify skips one record.

SKIPR — SKIP FORWARD PAST RECORD

The SKIPR directive repositions the named file forward past the specified logical record. It does not position the file past the end-of-information. If Modify is unable to locate the record in the forward search, it positions the file at the end-of-information and issues the message

```
RECORD NOT FOUND.
```

Format:

```
*SKIPR file,rname
```

file	Name of file to be positioned.
rname	Name of record on file that file is positioned after.

REWIND — REWIND FILES

The REWIND directive repositions one or more files to their first records.

Format:

*REWIND $file_1, file_2, \ldots, file_n$

$file_i$	Names of files to be rewound.

RETURN — RETURN FILES TO SYSTEM

The RETURN directive immediately returns files to the operating system.

Format:

*RETURN $file_1, file_2, \ldots, file_n$

$file_i$	Names of file to be returned.

FILE MANIPULATION DIRECTIVE EXAMPLES

Figure 5-1 illustrates several of the file manipulation directives discussed in this section.

```
batch,45000
$RFL,45000.
/old,dirfil
/lnh,r
       PRINT*,"LINE 1 ADDED BY MODIFICATION SET MODX."
--EOR--
       PRINT*,"LINE 2 ADDED BY MODIFICATION SET MODX."
--EOR--
DECKX                                                      Alternate directives file.
       PRINT*,"LINE 3 ADDED BY MODIFICATION SET MODX."
--EOR--
*EDIT DECK1
*EDIT DECK2
*EDIT DECK3
--EOR--
/old,opl=mainpl
/get,dirfil
/modify,l=0,n=newpl,c=comx
? *skip dirfil,2
? *ident modx
? *deck deck2
? *i 2
? *read dirfil,deckx
? *bksp dirfil,2
? *deck deck3
? *i 3
? *read dirfil                               File manipulation directives.
? *rewind dirfil
? *deck deck1
? *i 4
? *read dirfil
? *skipr dirfil,deckx
? *read dirfil
? *return dirfil
?
 MODIFICATION COMPLETE.
/copycf,comx
***     MAIN PROGRAM, DECK DECK1.                                       MOD1          1
        PROGRAM MAIN(OUTPUT)                                            DECK1         2
        PRINT*,"BEGIN MAIN PROGRAM."                                    DECK1         3
        CALL SUB1                                                       DECK1         4
        PRINT*,"LINE 1 ADDED BY MODIFICATION SET MODX."                 MODX          1
        PRINT*,"END MAIN PROGRAM."                                      DECK1         5
        STOP                                                            DECK1         6
        END                                                             DECK1         7
***     SUBROUTINE 1, DECK DECK2.                                       MOD1          1
        SUBROUTINE SUB1                                                 DECK2         2
        PRINT*,"LINE 3 ADDED BY MODIFICATION SET MODX."                 MODX          1
        PRINT*,"ENTER SUBROUTINE 1."                                    DECK2         3
*       CALL SUBROUTINE SUB2                                            MOD1          2
*       IN DECK DECK2.                      Compile file containing     MOD1          3
        CALL SUB2                           modifications from          DECK2         4
        PRINT*,"EXIT SUBROUTINE 1."         alternate directives        DECK2         5
        RETURN                              file.                       DECK2         6
        END                                                             DECK2         7
***     END DECK2.                                                      MOD1          4
***     SUBROUTINE 2, DECK DECK3.                                       MOD1          1
        SUBROUTINE SUB2                                                 DECK3         2
        PRINT*,"ENTER SUBROUTINE 2."                                    DECK3         3
        PRINT*,"LINE 2 ADDED BY MODIFICATION SET MODX."                 MODX          1
        PRINT*,"EXIT SUBROUTINE 2."                                     DECK3         4
        RETURN                                                          DECK3         5
        END                                                             DECK3         6
 END OF INFORMATION ENCOUNTERED.
```

Figure 5-1. File Manipulation Directive Examples (Sheet 1 of 2)

```
/catalog,newpl,r
         CATALOG OF NEWPL          FILE     1
    REC  NAME      TYPE        LENGTH     CKSUM     DATE

      1  DECK1     OPL  (64)       47      7152   76/01/22.
          MOD1      MODX

      2  DECK2     OPL  (64)       65      6115   76/01/22.
          MOD1      MOD2      MOD3      MODX

      3  DECK3     OPL  (64)       44      7430   76/01/21.
          MOD1      MODX

      4  OPL       OPLD             7      7403   76/01/23.

      5  * EOF *       SUM =      207
1
 CATALOG COMPLETE.
/rewind,comx
SREWIND,COMX.
/ftn,i=comx,l=0
      .215 CP SECONDS COMPILATION TIME
/lgo
 BEGIN MAIN PROGRAM.
 LINE 3 ADDED BY MODIFICATION SET MODX.
 ENTER SUBROUTINE 1.
 ENTER SUBROUTINE 2.
 LINE 2 ADDED BY MODIFICATION SET MODX.
 EXIT SUBROUTINE 2.
 EXIT SUBROUTINE 1.
 LINE 1 ADDED BY MODIFICATION SET MODX.
 END MAIN PROGRAM.
      .009 CP SECONDS EXECUTION TIME
```

Execution of modified program.

Figure 5-1. File Manipulation Directive Examples (Sheet 2 of 2)

COMPILE FILE DIRECTIVES

The directives described in this section provide user control during the write compile file phase. These directives are interpreted at the time the program library decks are written onto the compile file. A call for a common deck results in the deck being written on the compile file. Other directives allow control of file format.

The user can prepare his original source deck with compile file directives embedded in it, or he can insert compile file directives into program library decks as a part of a modification set. Compile file directives are not recognized when they are on the directives file; they do not terminate insertion, but are simply considered as text lines to be inserted.

Compile file directives include:

CALL	Write called deck onto compile file.
IFCALL	Write called deck onto compile file if name is defined.
NIFCALL	Write called deck onto compile file if name is not defined.
CALLALL	Write all decks onto compile file that have deckname beginning with specified character string.
IF	Include lines in compile file if specified attribute is true and until a reversal directive is encountered (ELSE or ENDIF).
ELSE	Reverse an IF directive conditional range.
ENDIF	Terminate an IF directive conditional range.
COMMENT	Generate COMMENT pseudo instruction for COMPASS.
WIDTH	Define number of columns preceding sequence information on compile file.
NOSEQ	Specify no sequence information on compile file.
SEQ	Specify sequence information on compile file.
WEOR	Write end-of-record on compile file.
CWEOR	Write end-of-record on compile file if the buffer is not empty.
WEOF	Write end-of-file on compile file.

NOTE

A common deck cannot call another common deck. That is, if the directives CALL, IFCALL, NIFCALL, or CALLALL are in a common deck, they are ignored.

CALL – CALL COMMON DECK

Modify places a copy of the requested deck on the compile file. It does not copy the request to the compile file. However, the new program library and the source file contain the CALL directive.

Format:

*CALL deckname

deckname	Name of common deck to be written on compile file.

IFCALL – CONDITIONALLY CALL COMMON DECKS

Modify places a copy of the requested deck on the compile file if the conditional name has been defined on a DEFINE directive during the modification phase. If the name has not been defined, the common deck is not written on the compile file. Modify does not copy the IFCALL directive to the compile file.

Format:

*IFCALL name, deckname

name	1- to 7-character conditional name.
deckname	Name of common deck to be written on compile file if name is defined.

NIFCALL – CONDITIONALLY CALL COMMON DECKS

Modify places a copy of the requested deck on the compile file if the conditional name has not been defined (refer to DEFINE directive, section 7) during the modification phase. If the name has been defined, the common deck is not written on the compile file.

Format:

```
*NIFCALL name, deckname
```

name	1- to 7-character conditional name.
deckname	Name of common deck to be written on compile file if name is not defined.

CALLALL – CALL RELATED COMMON DECKS

Modify places a copy on the compile file of every deck name beginning with the specified character string.

Format:

```
*CALLALL string
```

IF – TEST FOR CONDITIONAL RANGE

Modify tests the specified condition and, if true, writes all following lines onto the compile file until encountering a reversal (ELSE) or termination (ENDIF) directive. If the condition is false, the lines are skipped until a reversal or termination directive is encountered. Lines skipped in such a range are treated as inactive.

Format:

```
*IF atr, name, value
```

atr — Attribute; must be one of the following:

DEF	name defined
UNDEF	name undefined
EQ	name equal to value
NE	name not equal to value

ELSE – REVERSE CONDITIONAL RANGE

ELSE is a conditional range reversal directive. When encountered, the effects of a previous IF directive are reversed. An ELSE directive encountered without an IF range in progress is diagnosed as an error.

Format:

```
*ELSE
```

ENDIF – TERMINATE CONDITIONAL RANGE

ENDIF is a conditional range termination directive. When encountered, the effects of a previous IF directive are terminated. An ENDIF directive encountered without an IF range in progress is diagnosed as an error.

Format:

```
*ENDIF
```

COMMENT – CREATE COMMENT LINE

This directive causes Modify to create a COMPASS language COMMENT pseudo instruction (beginning in column 3) in the following format. Modify obtains the dates from the operating system.

LOCATION	OPERATION	VARIABLE SUBFIELDS
COMMENT	crdate	moddate comments

crdate	Creation date in the format Δyy/mm/dd.
moddate	Modification date in the format Δyy/mm/dd.

Format:

```
*COMMENT comments
```

comments — Character string.

WIDTH – SET LINE WIDTH ON COMPILE FILE

The WIDTH directive allows the user to change the width of lines during the compile phase. Modify uses the new width until it encounters another WIDTH directive.

Format:

```
*WIDTH n
```

n — Number of columns preceding sequence information on compile file and source file. Modify allows a maximum of 100 columns.

NOTE

During initialization of Modify, width is set to 72; additional columns of data are truncated.

NOSEQ – NO SEQUENCE INFORMATION

The NOSEQ directive allows the user to set the no sequence flag during the write compile file phase. When no sequence information is requested, Modify does not include sequence information on the compile file. A SEQ directive encountered subsequent to NOSEQ resumes sequencing.

Format:

```
*NOSEQ
```

SEQ – INCLUDE SEQUENCE INFORMATION

The SEQ directive allows the user to clear the no sequence flag during the write compile file phase and to begin placing sequence information on the compile file. A NOSEQ directive encountered subsequent to a SEQ sets the no sequence flag.

Format:

```
*SEQ
```

WEOR – WRITE END OF RECORD

Modify unconditionally writes an end-of-record on the compile file when encountering the WEOR directive.

Format:

```
*WEOR
```

CWEOR – CONDITIONALLY WRITE END OF RECORD

Modify writes an end-of-record on the compile file if information has been placed in the buffer since the last end-of-record was written.

Format:

```
*CWEOR
```

WEOF – WRITE END OF FILE

Modify writes an end-of-file on the compile file.

Format:

```
*WEOF
```

COMPILE FILE DIRECTIVE EXAMPLES

Figure 6-1 illustrates several of the compile file directives presented in this section.

```
batch,45000
$RFL,45000.
/old,opl=mainpl
/get,csub
/copycr,csub<-------------------------  {Copy of source file to be incorporated into
DECK4                                   {program library.
          IDENT   SUB3
          ENTRY   SUB3
*COMMENT  CALL DECK DECK5
***       CALL COMMON DECK.
*CALL     DECK5 <--------------- Notice call to common deck DECK5.
 SUB3     DATA    0             ENTRY/EXIT
          ORIGIN JOT
          EQ      SUB3          RETURN
          USE     //
 JOT      BSS     1
          END
 COPY COMPLETE.
/copycr,csub
DECK5
COMMON
 ORIGIN   MACRO   A
          SA1     66B           GET JOB ORIGIN
          MX0     24
          BX6     -X0*X1
          AX6     24
          SA6     A             STORE JOB ORIGIN
          ENDM
 COPY COMPLETE.
/modify,f,p=0,l=0,n=mainpl,c=coml,s=mainp
? *oplfile opl               \
? *rewind csub                \
? *create csub                 \
? *ident mod4                   \{Modify run to create new program library
? *deck deck1                    {consisting of source file and OPL.
? *i 2
?       common jot
? *i 3
?       call sub3
?       if(jot.eq.3)print*,"time-sharing job."
?       if(jot.ne.3)print*,"batch job."
? *deck deck4   )
? *i 0          |
? *weor         |
? *deck deck3   |
? *i 0          }               Addition of compile file directives.
? *weor         |
? *deck deck2   |
? *i 0          |
? *weor         )
?
 MODIFICATION COMPLETE.
/catalog,mainpl,r
          CATALOG OF MAINPL            FILE      1
    REC   NAME       TYPE            LENGTH    CKSUM      DATE

      1   DECK1      OPL   (64)          61     3171   76/01/22.
           MOD1       MOD4

      2   DECK3      OPL   (64)          37     2333   76/01/21.
           MOD1       MOD4

      3   DECK2      OPL   (64)          60     5455   76/01/22.
           MOD1       MOD2       MOD3       MOD4

      4   DECK4      OPL   (64)          47     5063   76/01/23.
           MOD4

      5   DECK5      OPLC  (64)          27     6354   76/01/23.
      6   OPL        OPLD                13     3706   76/01/23.

      7   * EOF *         SUM =         311
1
 CATALOG COMPLETE.
```

Figure 6-1. Compile File Directive Examples (Sheet 1 of 3)

```
/copycr,coml
***    MAIN PROGRAM, DECK DECK1.                                MOD1      1
       PROGRAM MAIN(OUTPUT)                                     DECK1     2
       COMMON JOT                                               MOD4      1
       PRINT*,"BEGIN MAIN PROGRAM."                             DECK1     3
       CALL SUB3                                                MOD4      2
       IF(JOT.EQ.3)PRINT*,"TIME-SHARING JOB."                   MOD4      3
       IF(JOT.NE.3)PRINT*,"BATCH JOB."                          MOD4      4
       CALL SUB1                                                DECK1     4
       PRINT*,"END MAIN PROGRAM."                               DECK1     5
       STOP                                                     DECK1     6
       END                                                      DECK1     7
 COPY COMPLETE.
/copycr,coml
***    SUBROUTINE 2, DECK DECK3.                                MOD1      1
       SUBROUTINE SUB2                                          DECK3     2
       PRINT*,"ENTER SUBROUTINE 2."                             DECK3     3
       PRINT*,"EXIT SUBROUTINE 2."                              DECK3     4
       RETURN                                                   DECK3     5
       END                                                      DECK3     6
 COPY COMPLETE.
```

Listing of compile file. Notice separation into records.

```
/copycr,coml
***    SUBROUTINE 1, DECK DECK2.                                MOD1      1
       SUBROUTINE SUB1                                          DECK2     2
       PRINT*,"ENTER SUBROUTINE 1."                             DECK2     3
*      CALL SUBROUTINE SUB2                                     MOD1      2
*      IN DECK DECK2.                                           MOD1      3
       CALL SUB2                                                DECK2     4
       PRINT*,"EXIT SUBROUTINE 1."                              DECK2     5
       RETURN                                                   DECK2     6
       END                                                      DECK2     7
***    END DECK2.                                               MOD1      4
 COPY COMPLETE.
```

Notice that Modify has replaced *COMMENT directive with COMPASS COMMENT statement on compile file.

```
/copycr,coml
         IDENT   SUB3                                           DECK4     1
         ENTRY   SUB3                                           DECK4     2
  COMMENT 76/01/23. 76/01/23. CALL DECK DECK5                   DECK4     3
***      CALL COMMON DECK.                                      DECK4     4
 ORIGIN  MACRO   A                                              DECK5     1
         SA1     66B            GET JOB ORIGIN                  DECK5     2
         MX0     24                                             DECK5     3
         BX6     -X0*X1                                         DECK5     4
         AX6     24                                             DECK5     5
         SA6     A              STORE JOB ORIGIN                DECK5     6
         ENDM                                                   DECK5     7
 SUB3    DATA    0              ENTRY/EXIT                      DECK4     6
         ORIGIN  JOT                                            DECK4     7
         EQ      SUB3           RETURN                          DECK4     8
         USE     //                                             DECK4     9
 JOT     BSS     1                                              DECK4    10
         END                                                    DECK4    11
 COPY COMPLETE.
```

*CALL DECK5 is replaced by contents of common deck.

```
/copycr,coml
 END OF INFORMATION ENCOUNTERED.
/replace,mainpl
/pack,coml
 PACK COMPLETE.
/ftn,i=coml,l=0
      .503 CP SECONDS COMPILATION TIME
/lgo
 BEGIN MAIN PROGRAM.
 TIME-SHARING JOB.
 ENTER SUBROUTINE 1.
 ENTER SUBROUTINE 2.
 EXIT SUBROUTINE 2.
 EXIT SUBROUTINE 1.
 END MAIN PROGRAM.
      .009 CP SECONDS EXECUTION TIME
/primary,mainp
$PRIMARY,MAINP.
```

Figure 6-1. Compile File Directive Examples (Sheet 2 of 3)

```
/lnh,r
DECK1
***   MAIN PROGRAM, DECK DECK1.
      PROGRAM MAIN(OUTPUT)
      COMMON JOT
      PRINT*,"BEGIN MAIN PROGRAM."
      CALL SUB3
      IF(JOT.EQ.3)PRINT*,"TIME-SHARING JOB."
      IF(JOT.NE.3)PRINT*,"BATCH JOB."
      CALL SUB1
      PRINT*,"END MAIN PROGRAM."
      STOP
      END
--EOR--
DECK3
*WEOR
***   SUBROUTINE 2, DECK DECK3.
      SUBROUTINE SUB2
      PRINT*,"ENTER SUBROUTINE 2."
      PRINT*,"EXIT SUBROUTINE 2."
      RETURN
      END
--EOR--
DECK2
*WEOR
***   SUBROUTINE 1, DECK DECK2.
      SUBROUTINE SUB1
      PRINT*,"ENTER SUBROUTINE 1."
*     CALL SUBROUTINE SUB2
*     IN DECK DECK2.
      CALL SUB2
      PRINT*,"EXIT SUBROUTINE 1."
      RETURN
      END
***   END DECK2.
--EOR--
DECK4
*WEOR
          IDENT  SUB3
          ENTRY  SUB3
*COMMENT  CALL DECK DECK5
***       CALL COMMON DECK.
*CALL     DECK5
 SUB3     DATA   0            ENTRY/EXIT
          ORIGIN JOT
          EQ     SUB3         RETURN
          USE    //
 JOT      BSS    1
          END
--EOR--
DECK5
COMMON
 ORIGIN   MACRO  A
          SA1    66B          GET JOB ORIGIN
          MX0    24
          BX6    -X0*X1
          AX6    24
          SA6    A            STORE JOB ORIGIN
          ENDM
--EOR--
```

Contents of source file created by Modify.

Note that source file contains call to common deck.

Figure 6-1. Compile File Directive Examples (Sheet 3 of 3)

SPECIAL DIRECTIVES

The directives described in this section provide extended features. They can be any place in the directive file for either creation or correction and primarily affect the operating features of Modify.

/	List comment.
PREFIX	Changes prefix character for directives other than compile file directives.
PREFIXC	Changes prefix character for compile file directives.
INWIDTH	Sets width of input line to be compressed.
DEFINE	Defines name under which subsequent IFCALL directive may cause a common deck to be written, or NIFCALL may prevent a common deck from being written.
MOVE	Moves decks on new program library.
UPDATE	Specifies editing sequence and modification set numbering.

/ — LIST COMMENT

Other than being copied onto the Modify statistics (list) output, a comment line is ignored. It can occur any place in the directives file.

Format:

```
*/ comment
```

Example:

```
*/ ******MODIFICATIONS******
```

PREFIX — CHANGE MODIFY DIRECTIVES PREFIX

The PREFIX directive resets the prefix character for subsequent Modify directives. It does not affect the prefix of compile file directives. When Modify is initialized, the character is preset to *. Modify uses * if a PREFIX directive is not used.

Format:

```
*PREFIX x
```

x	Character used in first column of directive (except compile file directive). A blank character is illegal.

PREFIXC — CHANGE COMPILE FILE DIRECTIVES PREFIX

The PREFIXC directive resets the compile directive character so that only compile file directives with the x prefix are recognized. If a PREFIXc directive is not encountered, the default (*) is used.

Format:

```
*PREFIXC x
```

x	Character used in first column of compile file directive. A blank character is illegal.

INWIDTH — SET WIDTH OF INPUT TEXT

The INWIDTH directive allows the user to set the width of input text from primary and alternate sources before it is compressed and written in the Modify library deck. An INWIDTH directive takes precedence over any previously defined width. INWIDTH can be placed anywhere in the directives file.

Format:

```
*INWIDTH n
```

n	Number of columns on input line to be compressed. Modify allows a maximum of 100 columns. During initialization of Modify, width is preset to 72.

DEFINE — DEFINE NAME FOR USE BY IFCALL, NIFCALL, IF

By defining a name and its associated value, a user establishes the conditions that must be met for a conditional call of a common deck. This allows external control of the calls embedded in source decks. If the name is not defined, an IFCALL for a common deck is ignored. If the name is defined, a NIFCALL for a common deck is ignored. A DEFINE directive must be processed in order for an IF conditional test to be true.

Format:

```
*DEFINE name, value
```

name	Name used in compile file IFCALL, NIFCALL, or IF directive.
value	Value assigned to symbol name (maximum value may be 3777777B). If omitted, name is defined with value zero.

MOVE – MOVE DECKS

The MOVE directive enables the user to reorder decks while producing a new program library. The decks, dname, are moved from their positions on the old library and placed after dname_r on the new library.

Format:

*MOVE dname_r, dname_1, dname_2, dname_n

UPDATE – UPDATE LIBRARY

Use of this directive causes Modify to continue sequencing rather than restart sequencing with each deck using the same IDENT. UPDATE also causes the order in which decks are edited to be according to their sequence on the old program library.

Format:

*UPDATE

SPECIAL DIRECTIVE EXAMPLES

Figure 7-1 illustrates several special directives. Note that compile file directives can be ignored (depending on language processor) by changing the compile file prefix character.

```
batch,45000
$RFL,45000.
/old,opl=mainpl
/modify,f,c=coml,n=mainpl,l=0
? */  change prefix character to #
? *prefix # <------------------------------------Change Modify directive prefix character.
? #ident mod6
? #deck deck4
? #i 4
?           space  4
? #prefixc #<--------------------------------- { Change compile file prefix character so
? #move deck1,deck2,deck3                       { directives on program library will be inter-
?                                               { preted as comments.
 MODIFICATION COMPLETE.
/catalog,mainpl,r
         CATALOG OF MAINPL         FILE      1
    REC  NAME       TYPE          LENGTH     CKSUM     DATE

      1  DECK1      OPL   (64)       61       3171    76/01/22.
          MOD1       MOD4

      2  DECK2      OPL   (64)       60       5455    76/01/22.
          MOD1       MOD2       MOD3       MOD4

      3  DECK3      OPL   (64)       37       2333    76/01/21.
          MOD1       MOD4

      4  DECK4      OPL   (64)       53       3057    76/01/23.
          MOD4       MOD6

      5  DECK5      OPLC  (64)       27       6354    76/01/23.
      6  OPL        OPLD            13       3675    76/01/23.

      7   * EOF *       SUM =       315
1
 CATALOG COMPLETE.                  \---- Notice reordered decks.
```

Figure 7-1. Special Directive Examples (Sheet 1 of 2)

```
/copycr,com1
***   MAIN PROGRAM, DECK DECK1.                                           MOD1        1
      PROGRAM MAIN(OUTPUT)                                                DECK1       2
      COMMON JOT                                                          MOD4        1
      PRINT*,"BEGIN MAIN PROGRAM."                                        DECK1       3
      CALL SUB3                                                           MOD4        2
      IF(JOT.EQ.3)PRINT*,"TIME-SHARING JOB."                              MOD4        3
      IF(JOT.NE.3)PRINT*,"BATCH JOB."                                     MOD4        4
      CALL SUB1                                                           DECK1       4
      PRINT*,"END MAIN PROGRAM."                                          DECK1       5
      STOP                                                                DECK1       6
      END                                                                 DECK1       7
*WEOR                                                                     MOD4        1
***   SUBROUTINE 1, DECK DECK2.                                           MOD1        1
      SUBROUTINE SUB1                                                     DECK2       2
      PRINT*,"ENTER SUBROUTINE 1."                                        DECK2       3
*     CALL SUBROUTINE SUB2                                                MOD1        2
*     IN DECK DECK2.                                                      MOD1        3
      CALL SUB2                                                           DECK2       4
      PRINT*,"EXIT SUBROUTINE 1."                                         DECK2       5
      RETURN                                                              DECK2       6
      END                                                                 DECK2       7
***   END DECK2.                                                          MOD1        4
*WEOR                                         Listing of compile file.    MOD4        1
***   SUBROUTINE 2, DECK DECK3.               Compile file directives     MOD1        1
      SUBROUTINE SUB2                         have been ignored.          DECK3       2
      PRINT*,"ENTER SUBROUTINE 2."                                        DECK3       3
      PRINT*,"EXIT SUBROUTINE 2."                                         DECK3       4
      RETURN                                                              DECK3       5
      END                                                                 DECK3       6
*WEOR                                                                     MOD4        1
          IDENT  SUB3                                                     DECK4       1
          ENTRY  SUB3                                                     DECK4       2
*COMMENT  CALL DECK DECK5                                                 DECK4       3
***       CALL COMMON DECK.                                               DECK4       4
          SPACE  4                                                        MOD6        1
*CALL     DECK5                                                           DECK4       5
 SUB3     DATA   0          ENTRY/EXIT                                    DECK4       6
          ORIGIN JOT                                                      DECK4       7
          EQ     SUB3       RETURN                                        DECK4       8
          USE    //                                                       DECK4       9
 JOT      BSS    1                                                        DECK4       10
          END                                                             DECK4       11
 COPY COMPLETE.
/copycr,com1
 END OF INFORMATION ENCOUNTERED.
```

Figure 7-1. Special Directive Examples (Sheet 2 of 2)

MODIFY CONTROL STATEMENT

The following control statement causes the Modify program to be loaded from the operating system library into central memory and to be executed. Parameters specify options and files.

```
MODIFY (p1,p2,...,pn)
```

The optional parameters, p_i, may be in any order within the parentheses. Generally, a parameter can be omitted or can be in one of the following forms.

option

option= value

option=0

where option is one or two characters as defined in the following text. Unless Q or X is selected, parameters CB, CG, CL, or CS are meaningless. Value is a 1- to 7-character name of a file or is a character string.

Option	Significance
A - Compressed compile file	
omitted	Compile file is not in compressed format.
A	Compile file is in compressed format.
C - Compile file output	
omitted or C	Compile output to be written on file COMPILE.
C=filename	Write compile output on named file.
C=0	No compile output.
CB - COMPASS binary; Q or X option only.	
omitted or CB	COMPASS binary output written on the load-and-go file (B=LGO).
CB=filename	COMPASS binary output written on the named file (B=filename).
CB=0	No binary output (B=0).
CG - COMPASS get text option; Q or X option only. Takes precedence over CS.	
CG	Load systems text from SYSTEXT (G=SYSTEXT).
CG=filename	Load systems text from named file (G=filename).
CG=0	SYSTEXT not defined (G=0).
omitted	Load systems text from overlay named in CS option.
CL - COMPASS list output including *comment lines. Q or X option only.	
CL	List output on OUTPUT file (L=OUTPUT).
CL=filename	List output on named file (L=filename).
omitted or CL=0	Short list instead of full list is generated on OUTPUT file (L=0).
CS - COMPASS systems text; Q or X option only.	
omitted or CS	Systems text on SYSTEXT overlay (S=SYSTEXT)
CS=filename	Systems text on named file (S=filename)
CS=0	No systems text (S=0)
CV - Character set conversion	
omitted or CV=0	No conversion takes place.
CV=63	Convert library created using 64-character set to 63-character set.
CV=64	Convert library created using 63-character set to 64-character set.

NOTE

When the CV=63 or CV=64 conversion option is selected, Modify forces C=0 (no compile file generation).

Conversion is recommended if the character set of the old program library is not the same as the character set used when the program library is modified. Use CATALOG to determine the character set of the program library (refer to volume 1 of the NOS Reference Manual). Check with a systems analyst to determine the character set in use at the site.

Option	Significance
D - Debug	
omitted	A directive or fatal error aborts the job.
D	A directive error does not abort the job; the D option does not affect fatal error processing.
F - Full edit	
omitted	Decks to be edited are determined by the U parameter or by EDIT directives.
F	All decks on the library are to be edited and written on new program library, compile file, and source file if the respective options are selected.

Option	Significance
I - Directive input	
omitted or I	Directives on job INPUT file.
I=filename	Directives comprise next record on named file.
I=0	No directive input.
L - List output	
omitted or L	List output is written on job OUTPUT file. This file is automatically printed.
L=filename	List output is written on the named file. It is the user's responsibility to assure that the file is saved at job end or is printed.
L=0	Modify does not generate a list output file.
LO - List options	
omitted or LO	List options E, C, T, M, W, D, and S are selected.
LO=$c_1, c_2 \ldots c_n$	Each character (c_i) selects an option to a maximum of seven options.

Option	Significance
A	List active lines in deck
C	List directives other than INSERT, DELETE, RESTORE, MODNAME, I, or D
D	List deck status
E	List errors
I	List inactive lines in deck
M	List modifications performed
S	Include statistics on listing
T	List text input
W	List compile file directives

Option	Significance
N - New program library output	
N	New program library to be written on file NPL.
N=filename	New program library to be written on named file. It is the user's responsibility to assure that the file is saved at job end.
omitted or N=0	Modify does not generate a new program library.

NOTE

If a new program library is being generated, an EVICT is performed upon it (NPL or filename) before it is written on (refer to the NOS Reference Manual, volume 1, for a description of EVICT).

Option	Significance
NR - No rewind of compile file	
omitted	Compile file is rewound at beginning and end of Modify run.
NR	Compile file is not rewound at beginning and end of Modify run.
P - Program library input	
omitted or P	Program library on file OPL.
P=filename	Program library on named file.
P=0	No program library input file.
Q - Execute named program; no rewind of directives file or list output file.	
omitted or Q=0	Assembler or compiler is NOT automatically called at end of the Modify run.
Q=program	At the beginning of the Modify run, Modify sets LO=E and sets the A parameter. At the end of the run, Modify calls the assembler or compiler specified by program.
Q	At the beginning of the Modify run, Modify sets LO=E and sets the A parameter. At the end of the run, Modify calls the COMPASS assembler. When this option is selected, the CB, CL, CS, and CG parameters are meaningful. Compiler input is assumed to be COMPILE. All other parameters are set by default. If CL is not specified with Q, lines beginning with an asterisk in column 1 are not written to the compile file (compile file directives are processed, however).
S - Source output; illegal when A, Q, or X are selected.	
S	Source output written on file SOURCE.
S=filename	Source output written on named file. It is the user's responsibility to assure that the file is saved at job end.
omitted or S=0	Modify does not generate a source output file.
U - Update edit	
omitted	Decks to be edited are determined by EDIT directives or by the F parameter.
U	Only decks for which directives file contains DECK directives are edited and written on the compile file, new program library, and source file if the respective options are on. F, if specified, takes precedence.
X - Execute named program; directives file and list output file rewound.	
	Same as Q option, except Modify directives input (I parameter) and list output (L parameter) files are rewound before processing.

Option	Significance
Z - Control statement input	
omitted	The control statement does not contain the input directives.
Z	The Modify control statement contains the input directives following the terminator; the input file is not read. This eliminates the need to use a separate input file for the directives when only a few directives are needed. The first character following the control statement terminator is the separator character. Example: MODIFY(Z)/*EDIT, DECK1/*EDIT, DECK2

MODIFY FILE FORMATS

Types of Modify files significant to Modify execution include:

- Source files
- Program library files
- Directives file
- Compile file

SOURCE DECKS AND FILES

A source file is a collection of information either prepared by the user or generated by Modify.

SOURCE DECKS PREPARED BY USER AS INPUT TO MODIFY

A user prepares a source deck for input to Modify by placing a deck name and optionally a COMMON statement in front of the source language deck (figure 3-1). At the same time, the user also inserts compile file directives, as required, into the source language deck to control compile file output from Modify. Each source deck is terminated by an end-of-record. A group of decks is terminated by an end-of-file or end-of-information. The deckname and COMMON statements are not placed on the program library.

Modify source decks should not be confused with a compiler or assembler program. A Modify source deck can contain any number of FORTRAN programs, subroutines or functions; COMPASS assembler IDENT statements; or set of data. Typically, each Modify deck contains one program for the assembler or compiler or one set of data.

SOURCE FILES GENERATED BY MODIFY

The source file generated as output by Modify contains a copy of all active lines within decks written on the compile file and new program library. The source file is optional output from Modify and is controlled through use of the S option on the Modify control statement. Once generated, the source file can be used as source input on a subsequent Modify run. The file is a coded file that contains 80-column images. Any sequencing information beyond the 80th column is truncated. When F is selected on the Modify control statement, the source file contains all lines needed to recreate the latest copy of the program library.

When U is selected, the source file contains only those decks named on DECK directives; that is, only the decks updated during the current Modify run.

When neither F nor U is selected, the source file contains only those decks explicitly requested on EDIT directives.

PROGRAM LIBRARY FILES

Program library files (figure 9-1) provide the primary form of input to Modify. When a program library file is input, it is an old program library and has a default name of OPL. When it is output, it is a new program library and has a default name of NPL.

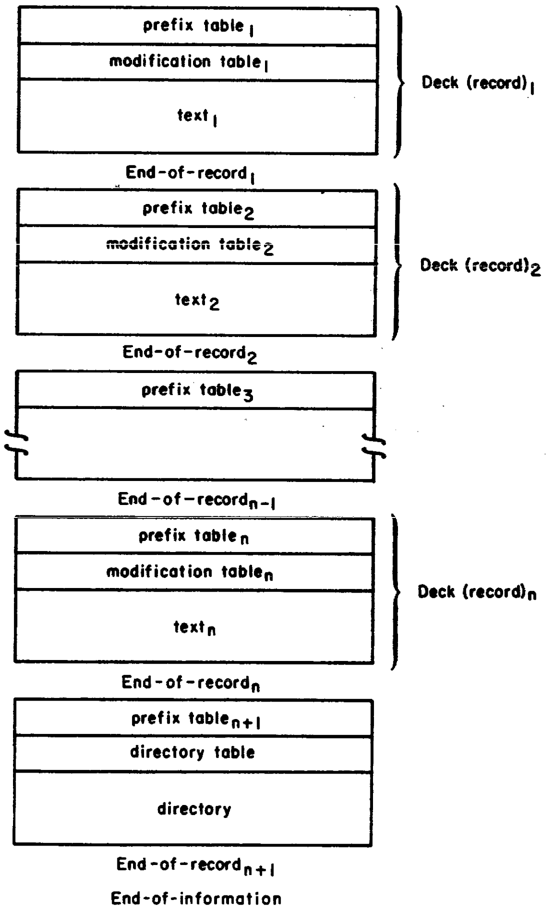


Figure 9-1. Library File Format

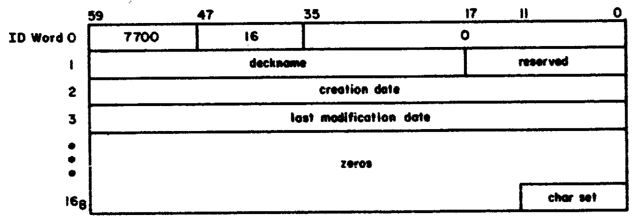
Before writing the new program library, an EVICT is performed on the file. Refer to the NOS Reference Manual, volume 1, for a description of the EVICT operation.

A program library consists of a record for each deck on the library. The last deck record is followed by a record containing the library directory. The contents of the new program library is determined by EDIT directives and the control statement options. Only edited decks are written on the new program library.

DECK RECORDS

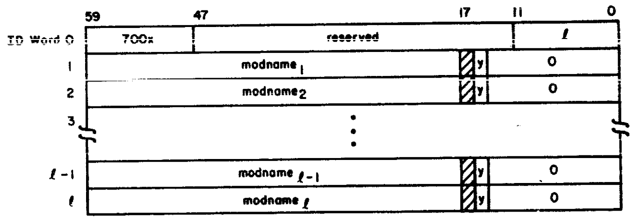
Each deck record consists of a prefix table, a modification table, and text.

Prefix Table Format:



Word	Bits	Field	Description
ID	59-48	Table type	Identifies table as prefix table.
	47-36	wc	Word count; length of table is 16_8 words.
	35-00	none	Reserved for future system use.
1	59-18	deckname	Name of deck obtained for source deck identification line; 1 to 7 characters.
	17-00	none	Reserved for future system use.
2	59-00	creation date	Date that deck was created. Format of date is: yy/mm/dd.
3	59-00	latest modification date	Date of most recent entry in modification table. Format of the date is the same as for creation date.
16_8	11-00	char set	Identifies character set used to create this deck. 0000_8 63-character set; 0064_8 64-character set

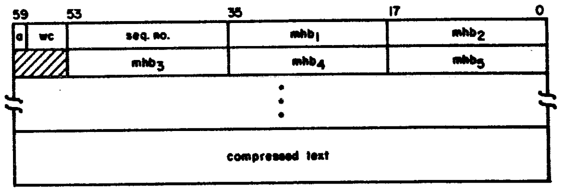
Modification Table Format:



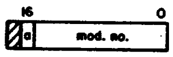
Word	Bits	Field	Description
ID	59-48	Table type	Identifies table as modification table. The least significant digit indicates whether the deck is common or not as follows: 1 Deck is not common; 2 Deck is common
	47-12	none	Reserved for future system use.
	11-00	ℓ	Number of modification names in table.
word$_i$	59-18	modname$_i$	1- to 7-character modification set name. Each modification to a deck causes a new entry in this table.
	16	y_i	YANK flag: 0 Modifier not yanked; 1 Modifier yanked

Text Format:

Text is an indefinite number of words that contain a modification history and the compressed image of each line in the deck. Text for each line is in the following format.



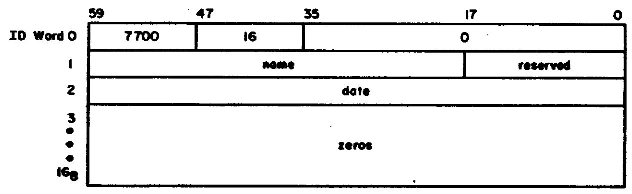
Bits	Field	Description
59	a	Activity bit: 0 Line is inactive; 1 Line is active
58-54	wc	Number of words of compressed text.
53-36	seq. no.	Sequence number of line (octal) according to position in deck or modification set.

Bits	Field	Description
35-18 and subsequent 18-bit bytes	mhb_i	Modification history byte. Modify creates a byte for each modification set that changes the status of the line. Modification history bytes continue to a zero byte. Since this zero byte could be the first byte of a word and the compressed line image begins a new word, the modification history portion of the text could terminate with a zero word. The format of mhb_i is:  a — Activate bit 0 Modification set deactivated the line 1 Modification set activated the line mod. no. — Index to the entry in the modification table that contains the name of the modification set that changes the line status. A modification number of zero indicates the deck name.
	compressed text	The compressed image of the line is display code. One or two spaces are each represented by 55_8; they are not compressed. Three or more embedded spaces are replaced in the image as follows: 3 spaces replaced by 0002 4 spaces replaced by 0003 ⋮ ⋮ ⋮ 64 spaces replaced by 0077_8 65 spaces replaced by 007755_8 66 spaces replaced by 00770001_8 67 spaces replaced by 00770002_8, etc. Trailing spaces are not considered as embedded and are not included in the line image. On a 64-character set program library or compressed compile file, a 00 character (colon) is represented as a 0001 byte. A 12-bit zero byte marks the end of the line.

DIRECTORY RECORD

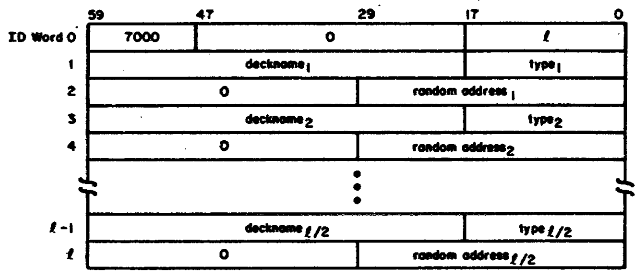
The library file directory contains a prefix table followed by a table containing a two-word entry for each deck in the library. Directory entries are in the same sequence as the decks on the library.

Prefix Table Format:



name	A Modify-generated directory has the name OPL. However, if the name of the directory is changed (by LIBEDIT, for example), that name is retained on new program libraries then generated.

Directory Table Format:



Word	Bits	Field	Description
ID	59-48	Table type	Identifies table as program library directory.
	17-00	ℓ	Directory length excluding ID word.
1, 3, ..., ℓ -1	59-18	$deckname_i$	Name of program library deck; 1 to 7 characters left-justified.
	17-00	$type_i$	Type of record. 6 Old program library deck (OPL) 7 Old program library common deck (OPLC) 10 Old program library directory (OPLD)
2, 4, ..., ℓ	29-00	random $address_i$	Address of deck relative to beginning of file.

NOTE

Other record types are defined but are ignored by Modify (refer to the NOS Reference Manual, volume 1, for a complete description of record types).

DIRECTIVES FILE

The directives file contains the Modify directives record. This record consists of initialization, file manipulation, and modification directives, and any source lines (including compile directives) to be inserted into the program library decks. An option on the Modify control statement designates the file from which Modify reads directives. Normally, the directives file is the job INPUT file. READ and READPL directives cause Modify to stop reading directives from the directives file named on the Modify statement and to begin reading from some other file containing directives or insertion lines.

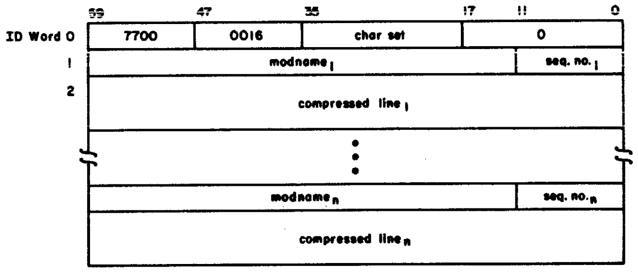
COMPILE FILE

The compile file is the primary form of output for Modify. It can be suppressed by the user as a Modify control statement option, when no compilation or assembly follows the modification.

If a compile file is specified on the Modify control statement, Modify writes the edited programs on it in a format acceptable as source input to an assembler, compiler, or other data processor. Through control statement parameters and directives, a user can specify whether the text on the file is to be compressed or expanded, sequenced or unsequenced. If the text is expanded, the user can also specify the width of each line of text preceding the sequence information.

Expanded compile file format for each line consists of x columns of the expanded line (where x is the width requested), followed by 14 columns of sequence information, if sequencing information is requested, and terminated by a zero byte. An end-of-record terminates the decks written on the compile file.

Compressed Compile File (A-Mode) Format:



char set	Character set of record. 0000_8 signifies 63-character set. 0064_8 signifies 64-character set.
seq. no.$_i$	Sequence number of the line relative to the modification set identified by modname.
compressed line	A line in compressed form. Refer to the compressed text description for text formats of deck records.

SCRATCH FILES

Modify uses scratch files in three situations.

Scratch File 1 (SCR1)	Used when common decks are modified and no new program library is requested.
Scratch File 2 (SCR2)	Used when insertions overflow memory.
Scratch File 3 (SCR3)	Used when a CREATE or COPYPL directive is processed. This file is in program library format.

These files are returned by Modify at the end of the Modify run.

CREATE PROGRAM LIBRARY

EXAMPLE 1

This example illustrates how Modify can be used to construct a file in program library format from source decks. This example contains only one source deck (PROG) consisting of a FORTRAN program. The deck is terminated by an end-of-file card. The next record on INPUT contains the directives. It is the user's responsibility to save the newly created program library (TAPE) for use in future Modify runs.

Unless C=0 is specified, a compile file is generated. This example shows the compile file (COMPILE) being used as input to the compiler. The compiler places the compiled program on LGO; the LGO card calls for loading and execution of the compiled program.

```
COPYBF(INPUT,SOURCE)
MODIFY(P=0,N=TAPE,F)
FTN(I=COMPILE)
 :<-------------------------
                             \
LGO.                          > File related cards
7/8/9                        /
PROG                        /
 :<-------------------------

(SOURCE DECK)
 :

6/7/9
*REWIND  SOURCE <--------------- Directives Input
*CREATE  SOURCE
6/7/8/9
```

EXAMPLE 2

This example illustrates creation of a library from source decks on a source file other than INPUT. After the library has been created, it can be modified, edited, and written on a compile file for use by an assembler or compiler.

```
Contents of File SALLY:          Job Deck:
RON                              (JOB CARD)
 :                                :<-------------- File related
                                                   cards
(SOURCE DECK FOR RON)            MODIFY(N,F,P=0)
 :                                :

*CALL    TOM                     7/8/9              )
*IFCALL  REQ,JACK                *REWIND   SALLY    |
7/8/9                            *CREATE   SALLY    |
TOM                               :                  > Directives Input
COMMON                                              |
 :                               *DEFINE   REQ      |
                                  :                 )
(SOURCE DECK FOR TOM)
 :                               7/8/9

7/8/9                             :
JACK
COMMON
 :

(SOURCE DECK FOR JACK)
 :

6/7/8/9
```

MODIFY PROGRAM LIBRARY

EXAMPLE 1

In this example, Modify uses all default parameters. The sequencing information shown for inserted cards is assigned during modification.

```
  :
MODIFY.                          File related cards
  :
7/8/9
*IDENT   MOD10
*DECK  BOTTLE
*/  *****MODIFICATIONS
*D       10
*D         4
(CARD TO BE INSERTED IS ASSIGNED MOD10.1)
*D       20,22                                                           Modification
(CARDS TO BE INSERTED ARE ASSIGNED MOD10.2 THROUGH MOD10.4)              set MOD10
I        MOD9.30
(CARD TO BE INSERTED IS ASSIGNED MOD10.5)
*EDIT    BOTTLE
6/7/8/9
```

EXAMPLE 2

This job modifies deck EDNA for replacement on the program library. No compile file is produced.

```
  :
MODIFY(N, C=0)                   File related cards
  :
7/8/9
*IDENT      A2                                    Modification set A2
*DECK       EDNA
*MODNAME    A1
*/  *****MODIFICATIONS
*D     30                                         Delete card A1.30
 TAG        RJ       CHECK                        Insert card A2.1
*MODNAME EDNA
*I      7011
 ERR        SA1      LIST1
            ZR       X1,ABORT                     Insert cards A2.2 through A2.5
            PRINT    (0***   ERROR 131    ***)    after EDNA.7011
            EQ       ABORT
*D   7644,7650
*EDIT   EDNA                                      Delete cards EDNA.7644 through
                                                  EDNA.7650
6/7/8/9
```

MOVE TEXT

EXAMPLE 1

The job illustrated below calls Modify twice. On the first call, Modify deactivates all but cards 32 through 54 and writes the source for these cards on source file FRANK. On the second call, Modify deletes the remainder of the cards and reinserts the saved cards at the beginning of KEN.

```
 :
MODIFY(S=FRANK,C=0)                     File related cards
MODIFY(N,C=CAL)
 :
7/8/9
*IDENT   MOV1                           Modification set MOV1
*DECK  KEN
*D      1,31                            Delete cards before card KEN.32
*D     55,63                            Delete cards KEN.55 through KEN.63
*EDIT   KEN                             Transfer remaining cards (KEN.32 through
7/8/9                                     KEN.54) to source file FRANK
*IDENT  MOV2                            Modification set MOV2
*REWIND  FRANK
*DECK   KEN
*D        32,54                         Delete remainder of cards in KEN
*I        0                             Insert cards at beginning of KEN
*READ    FRANK,KEN                      Read insertion text from deck KEN on file
*EDIT      KEN                            FRANK
6/7/8/9
```

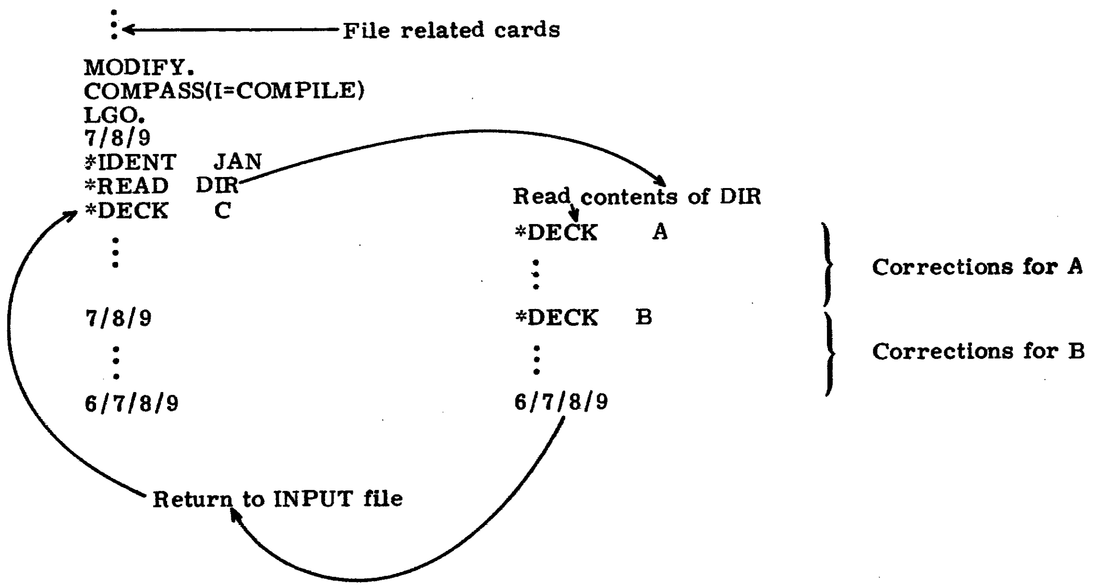
EXAMPLE 2

This job moves text cards from one deck to another. On the first call to Modify, cards 32 through 54 of deck KEN on file OPL are saved on source file FRANK. On the second call, the saved cards are inserted into deck WILL.

```
 :
MODIFY(S=FRANK,C=0)                     File related cards
MODIFY(N,C=MEL)
 :
7/8/9
*IDENT    F1                            Modification set F1
*DECK     KEN
*D        1,31                          Delete cards KEN.1 through KEN.31
*D        55,63                         Save cards KEN.32 through KEN.54 on source
*EDIT     KEN                             file FRANK
7/8/9
*REWIND FRANK
*IDENT    F2
*DECK     WILL
*I        25                            Insert text after card WILL.25
*READ     FRANK,KEN                     Insertion text taken from deck KEN on file FRANK
*EDIT     WILL                          Deck WILL is written on NPL and compile file MEL
6/7/8/9
```

READ DIRECTIVES FROM AN ALTERNATE FILE

This job illustrates how the READ directive can be used to change the source of directives and correction text from the primary input file (in this case INPUT) to some other file.



```
  :
  :
MODIFY.
COMPASS(I=COMPILE)
LGO.
7/8/9
*IDENT    JAN
*READ   DIR
*DECK     C
  :
  :
7/8/9
  :
  :
6/7/8/9
```

```
*DECK     A
  :
  :
*DECK    B
  :
  :
6/7/8/9
```

YANK AND UNYANK MODIFICATION SETS

This example illustrates a job that logically removes all of the modification sets applied to program library LIB from the modification set named JULY and on. The change is not incorporated into the library; it is for the benefit of this run only.

```
  :<------------------------------ File related cards
  :
MODIFY(P=LIB, F)
COMPASS(I=COMPILE)
LGO.
7/8/9
*IDENT    NEGATE
*DECK     MASTER
*YANK     JULY,*
6/7/8/9
```

To incorporate the preceding change on a new program library, add the N parameter to the Modify statement.

The effects of a YANK can be nullified in future runs and, consequently, the effects of the yanked modification sets can be restored through the UNYANK directive. Such a modification might appear as follows:

```
*IDENT    RESTORE
*DECK     MASTER
*UNYANK   JULY,*
```

PURGE DECKS

Decks BAD, WORSE, and WORST are no longer needed. The following job removes them from the library. They could also be removed through a selective edit using EDIT directives. In either case, the removal is permanent.

```
  :  <-----------------------
MODIFY(N, C=0, F)             \
  :  <------------------------- File related cards
7/8/9
*PURDECK  BAD,WORSE,WORST
6/7/8/9
```

CHANGE THE DIRECTIVES PREFIX CHARACTER

EXAMPLE 1

This example illustrates how to maintain directives input on a library. Because * is the prefix used on the library, a different prefix is required when modifying the library. In this case, / becomes the prefix character.

```
ATTACH(OPL)
GET(FIX)
MODIFY(P=FIX, C=Z, N=FIX2)
REWIND(Z)
COPYSBF(Z, OUTPUT)
REWIND(Z)
MODIFY(I=Z)
COMPASS(I, S, B=LT01)
  :
7/8/9
*PREFIX   /
/WIDTH    58
/IDENT    F1
/DECK     CORR
/I        873
*I        1007

          LDC       7777B
          STM       STMA+1
/D        880
/EDIT     CORR
6/7/8/9
```

The contents of deck CORR on compile file Z are as follows:

```
*IDENT    NIX                         CORR    1
*DECK     GRM1TD                      CORR    2
*I        MHD2.19                     CORR    3
  :
*D        997,1000                    CORR  873
*I        1007                        F1      1  \
          LDC       7777B             F1      2   } Inserted cards
          STM       STMA+1            F1      3  /
            :
          LJM       STM               CORR  879 <------- Instruction CORR.880
*D        980,984                     CORR  881           has been deleted
```

After file Z is produced, the deck GRM1TD is modified by the contents of Z. The resulting compile file (COMPILE) contains COMPASS language PPU code and is assembled using COMPASS.

The job produces a new program library (FIX2) which replaces FIX so that the changes to deck CORR are saved.

The resulting COMPASS listing would appear as follows:

```
                                    Corrections        Contents of
   .                                 on File Z          COMPILE
   .                               (Correction IDs)     (Deck IDs)
   .
STD    SM                                             GRM1TD      1007
LOC    7777B                         F1        2         NIX        11
STM    STMA+1                        F1        3         NIX        12
   .
   .
   .
```

Since the comments go through the correction identification, the INWIDTH directive must be deleted if a new program library is generated. However, for maintenance, there is an advantage of seeing the correction identifiers with the deck identifiers.

EXAMPLE 2

This example illustrates changing the compile file prefix character so that when Modify produces the compile file, it recognizes only directives using the specified prefix. The directives prefix, in this case, is unaltered.

```
  .
  .
  .
ATTACH(OPL)
MODIFY.
COMPASS(I,S,B)
7/8/9
*IDENT    TEST1
*DECK     TEST
*PREFIXC   /
*EDIT     TEST
6/7/8/9
```

Deck TEST contains the following:

```
            .
            .
            .
           LDM     TCLT
           STD     CM
            .
            .
            .
*CALL   PPC
/CALL   PPCA
```

Modify ignores the common deck call to PPC. COMPASS interprets it as a comment card. Modify acts on the common deck call to PPCA and replaces the /CALL directive with a copy of common deck PPCA.

USE OF THE Z PARAMETER

EXAMPLE 1

Suppose you want to create a compile file using an alternate OPL. The following deck illustrates this technique.

```
  .
  .
  .
MODIFY(Z)/*OPLFILE,OPLZ/*EDIT,DECK1
  .
  .
  .
6/7/8/9
```

EXAMPLE 2

Another use of Z might be to request editing of specific decks:

```
  .
  .
  .
MODIFY(Z)/*EDIT,DECK1,DECK2
  .
  .
  .
6/7/8/9
```

SAMPLE FORTRAN PROGRAM

This set of Modify examples illustrates how Modify can be used for maintaining a FORTRAN Extended program in program library format. The FORTRAN program calculates the area of a triangle from the base and height read from the words in the data record.

EXAMPLE 1

The following job places the FORTRAN program and subroutine as a single deck (ONE) on the new program library (NPL) and on the compile file (COMPILE). Following Modify execution, FORTRAN is called to compile the program. The LGO card calls for execution of the compiled program. This program does not execute because of an error in the SUBROUTINE statement. The name of the subroutine should be MSG, not MSA.

```
  :
COPYBF(INPUT,S)
MODIFY(P=0,N,F)                File related cards
FTN(I=COMPILE)
LGO.
  :
7/8/9     END OF RECORD
ONE <-------------------------------------------- Deck name
       PROGRAM ONE (INPUT,OUTPUT,TAPE1)
       PRINT 5
5      FORMAT  (1H1)
10     READ 100,BASE,HEIGHT,I
100    FORMAT(2F10.2,  I1)
       IF   (I.GT.0)   GO TO 120
       IF   (BASE.LE.0)  GO TO 105
       IF   (HEIGHT.LE.0)   GO TO 105
       GO TO 106
105    CALL MSG
106    AREA =  .5*BASE*HEIGHT
       PRINT  110,BASE,HEIGHT,AREA
110    FORMAT(///,*   BASE=*F20.5,*   HEIGHT=*
      1F18.5,/,*   AREA=*F20.5)
       WRITE(1)   AREA                        Should be
       GO TO 10                               SUBROUTINE MSG
120    STOP
       END
       SUBROUTINE MSA <-
       PRINT 400
400    FORMAT  (///,* FOLLOWING INPUT DATA NEGATIVE OR ZERO *)
       RETURN
       END
6/7/9   END OF FILE <------------------------ End of source deck
*REWIND  S
*CREATE  S  <-------------------------------- Directives input
7/8/9     END OF RECORD
     200.24        500.76
     300.24        600.76
     400.00        700.00
     326.32        425.36
     500.00        600.00              Data record
     000.00        150.00
     700.43        800.00
     100.00        300.00
     050.00        100.00
     150.00        200.00
                        1
6/7/8/9 END OF INFORMATION
```

EXAMPLE 2

Examination of Modify output from the creation job reveals that the erroneous SUBROUTINE statement has card identifier ONE. 20. The following job corrects the error and generates a new program library.

```
 .
 .
 .
MODIFY(N, F)
FTN(I=COMPILE)
LGO.
7/8/9      END OF RECORD
*IDENT     MOD1
*DECK      ONE
*DELETE  20
        SUBROUTINE MSG <------------------ Identified as MOD1. 1 on NPL
7/8/9
    200.24          500.76      )
    300.24          600.76      |
    400.00          700.00      |
    326.32          425.36      |
    500.00          600.00      }   Data record
    000.00          150.00      |
    700.43          800.00      |
    100.00          300.00      |
    050.00          100.00      |
    150.00          200.00      )
                          1
6/7/8/9   END OF INFORMATION
```

EXAMPLE 3

This job uses the same input as the first job but divides the program into two decks: ONE and MSG. Deck MSG is a common deck. A CALL MSG directive is inserted into deck ONE to ensure that MSG is written on the compile file whenever deck ONE is edited.

```
COPYBF(INPUT,S)
MODIFY(P=0,N,F)
FTN(I=COMPILE)
LGO.
  :  <------------------------ File related cards

7/8/9     END OF RECORD
ONE
          PROGRAM ONE (INPUT,OUTPUT,TAPE1)
          PRINT 5
5         FORMAT (1H1)
10        READ 100,BASE,HEIGHT,I
100       FORMAT(2F10.2, I1)
          IF  (I.GT.0)   GO TO 120
          IF  (BASE.LE.0)  GO TO 105
          IF  (HEIGHT.LE.0)  GO TO 105
          GO TO 106
105       CALL MSG
106       AREA =  .5*BASE*HEIGHT
          PRINT  110,BASE,HEIGHT,AREA
110       FORMAT (///,*  BASE=*F20.5,* HEIGHT=*
         IF18.5,/,*   AREA=*F20.5)
          WRITE (1)  AREA
          GO TO 10
120       STOP
          END
*CALL    MSG <--------------------------      Replaced by common deck MSG
7/8/9     END OF RECORD                       on compile file
MSG
COMMON
          SUBROUTINE MSG
          PRINT 400
400       FORMAT (///,* FOLLOWING INPUT DATA NEGATIVE OR ZERO *)
          RETURN
          END
6/7/9 END OF FILE
*REWIND   S
*CREATE  S
7/8/9     END OF RECORD
      200.24        500.76  )
      300.24        600.76  |
      400.00        700.00  |
      326.32        425.36  |
      500.00        600.00  |
      000.00        150.00  }        Data record
      700.43        800.00  |
      100.00        300.00  |
      050.00        100.00  |
      150.00        200.00  )
                         1
6/7/8/9 END OF INFORMATION
```

EXAMPLE 4

This example adds a deck to the library created in the previous example. With no new program library generated (N is omitted from Modify card), the addition is temporary.

```
 :
COPYBF(INPUT, S)
MODIFY.                                     File related cards
FTN(I=COMPILE)
LGO.
 :

7/8/9     END OF RECORD
TWO
       PROGRAM  TWO(INPUT,OUTPUT)
          :
       END
*CALL MSG                                   Replaced by common deck MSG on
6/7/9                                       compile file
*REWIND S
*CREATE S
*IDENT   MOD2
*DECK MSG
*DELETE MSG.3
400      FORMAT (///,* FOLLOWING INPUT DATA POSITIVE *)
*EDIT   TWO
7/8/9
(DATA RECORD)
6/7/8/9
```

STANDARD CHARACTER SET A

CDC GRAPHIC	ASCII GRAPHIC SUBSET	DISPLAY CODE	HOLLERITH PUNCH (026)	EXTERNAL BCD CODE	ASCII PUNCH (029)	ASCII CODE
:†	:	00†	8-2	00	8-2	3A
A	A	01	12-1	61	12-1	41
B	B	02	12-2	62	12-2	42
C	C	03	12-3	63	12-3	43
D	D	04	12-4	64	12-4	44
E	E	05	12-5	65	12-5	45
F	F	06	12-6	66	12-6	46
G	G	07	12-7	67	12-7	47
H	H	10	12-8	70	12-8	48
I	I	11	12-9	71	12-9	49
J	J	12	11-1	41	11-1	4A
K	K	13	11-2	42	11-2	4B
L	L	14	11-3	43	11-3	4C
M	M	15	11-4	44	11-4	4D
N	N	16	11-5	45	11-5	4E
O	O	17	11-6	46	11-6	4F
P	P	20	11-7	47	11-7	50
Q	Q	21	11-8	50	11-8	51
R	R	22	11-9	51	11-9	52
S	S	23	0-2	22	0-2	53
T	T	24	0-3	23	0-3	54
U	U	25	0-4	24	0-4	55
V	V	26	0-5	25	0-5	56
W	W	27	0-6	26	0-6	57
X	X	30	0-7	27	0-7	58
Y	Y	31	0-8	30	0-8	59
Z	Z	32	0-9	31	0-9	5A
0	0	33	0	12	0	30
1	1	34	1	01	1	31
2	2	35	2	02	2	32
3	3	36	3	03	3	33
4	4	37	4	04	4	34
5	5	40	5	05	5	35

3AE13A

CDC GRAPHIC	ASCII GRAPHIC SUBSET	DISPLAY CODE	HOLLERITH PUNCH (026)	EXTERNAL BCD CODE	ASCII PUNCH (029)	ASCII CODE
6	6	41	6	06	6	36
7	7	42	7	07	7	37
8	8	43	8	10	8	38
9	9	44	9	11	9	39
+	+	45	12	60	12-8-6	2B
−	−	46	11	40	11	2D
*	*	47	11-8-4	54	11-8-4	2A
/	/	50	0-1	21	0-1	2F
(	(	51	0-8-4	34	12-8-5	28
)	)	52	12-8-4	74	11-8-5	29
$	$	53	11-8-3	53	11-8-3	24
=	=	54	8-3	13	8-6	3D
BLANK	BLANK	55	NO PUNCH	20	NO PUNCH	20
,(COMMA)	,(COMMA)	56	0-8-3	33	0-8-3	2C
.(PERIOD)	.(PERIOD)	57	12-8-3	73	12-8-3	2E
≡	#	60	0-8-6	36	8-3	23
[	[	61	8-7	17	12-8-2	5B
]	]	62	0-8-2	32	11-8-2	5D
%††	%	63	8-6	16	0-8-4	25
≠	" (QUOTE)	64	8-4	14	8-7	22
→	_(UNDERLINE)	65	0-8-5	35	0-8-5	5F
∨	!	66	11-0	52	12-8-7	21
∧	&	67	0-8-7	37	12	26
↑	'(APOSTROPHE)	70	11-8-5	55	8-5	27
↓	?	71	11-8-6	56	0-8-7	3F
<	<	72	12-0	72	12-8-4	3C
>	>	73	11-8-7	57	0-8-6	3E
≤	@	74	8-5	15	8-4	40
≥	\	75	12-8-5	75	0-8-2	5C
¬	‿(CIRCUMFLEX)	76	12-8-6	76	11-8-7	5E
;(SEMICOLON)	; (SEMICOLON)	77	12-8-7	77	11-8-6	3B

3AE6A

† TWELVE OR MORE ZERO BITS AT THE END OF A 60-BIT WORD ARE INTERPRETED AS END-OF-LINE MARK RATHER THAN TWO COLONS. END-OF-LINE MARK IS CONVERTED TO EXTERNAL BCD 1632.

†† IN INSTALLATIONS USING THE CDC 63-GRAPHIC SET, DISPLAY CODE 00 HAS NO ASSOCIATED GRAPHIC OR HOLLERITH CODE; DISPLAY CODE 63 IS THE COLON (8-2 PUNCH). THE SELECTION OF THE 63- OR 64-CHARACTER SET FOR TAPES IS AN INSTALLATION OPTION.

OUTPUT LISTING AND MESSAGES B

Depending on list options selected on the Modify control statement, list output for Modify contains the following.

- Input directives
- Status of each deck

  Modifiers are listed first, followed by a list of activated lines, deactivated lines, active lines, and inactive lines as they are encountered. To the left of each line are two flags, a status flag and an activity flag. The status flag can be I (inactive) or A (active). The activity flag can be D (deleted) or A (activated). Following these lines are the unprocessed modifications and errors, if any. The last line contains a count of active lines, inactive lines, and inserted lines.

- Statistics

  This includes lists of the following.

  Decks on program library

  Common decks on program library

  Decks added by initialization directives

  Decks on new program library

  Decks written on compile file

  A replaced deck is enclosed by parentheses. Completing the statistics is a line containing counts of the number of lines on the compile file and the amount of storage used during the Modify run.

- Errors

  Modify prints the line in error, if any, above the diagnostic message. Error messages other than those identified as fatal can be overridden through selection of the Modify statement D (debug) option.

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
CARD NOT REACHED.	Sequence number exceeds deck range.	Use correct sequence number.	MODIFY
COLUMN OUT OF RANGE.	Requested width exceeds maximum allowed (100).	Change width to 100 or less.	MODIFY
COPY FILE EMPTY.	No information on program library being copied.	Verify that COPY file exists and is properly positioned at BOI.	MODIFY
CREATION FILE EMPTY.	No source decks on file being used for creation.	Verify that creation file contains proper source decks.	MODIFY
CV OPTION INVALID.	CV option other than 63 or 64.	Specify 63 or 64 for conversion option.	MODIFY, OPLEDIT
DIRECTIVE ERRORS.	A format error has been detected during processing of directives. Fatal error.	Consult listing for description of error.	MODIFY, OPLEDIT
DUPLICATE MODIFIER NAME.	Modifier or IDENT has been used previously for the deck.	Choose unique name for deck.	MODIFY
ERROR IN ARGUMENTS.	An invalid parameter has been encountered on the OPLEDIT control statement.	Correct control statement and retry.	OPLEDIT
ERROR IN DIRECTORY.	The program library contains an error. Fatal error.	Use COPY or COPYPL to create new program library.	MODIFY, OPLEDIT
ERROR IN MODIFY ARGUMENTS.	Illegal parameter on Modify control statement. Fatal error.	Consult manual for correct control statement syntax.	MODIFY
FILE NAME CONFLICT.	The same file cannot be used for both applications without conflict. Fatal error.	Use different file name for one of the applications.	MODIFY, OPLEDIT
FIRST CARD IS AFTER SECOND CARD.	Parameters are erroneous or lines are out of order.	Verify that correct line sequence is used.	MODIFY
FORMAT ERROR IN DIRECTIVE.	A format error has been detected in a directive.	Consult manual for correct format.	MODIFY, OPLEDIT

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
ILLEGAL DIRECTIVE.	Directive is out of sequence. For example, the CREATE directive is after a modification directive for Modify.	Use correct sequence.	MODIFY, OPLEDIT
ILLEGAL NUMERIC FIELD.	Invalid parameter on Modify or OPLEDIT control statement.	Verify control statement parameters and retry.	MODIFY, OPLEDIT
INVALID ATTRIBUTE.	Attribute specified on IF directive is other than EQ, NE, DEF, or UNDEF.	Use correct attribute.	MODIFY
-LC-ERROR, MUST BE ECTMWDSIA-	Illegal list option requested. Fatal error.	Specify either E, C, T, M, W, D, S, I, or A for list option.	MODIFY
MEMORY OVERFLOW.	Insufficient field length has been specified for OPLEDIT to execute.	Increase field length with RFL control statement and retry.	OPLEDIT
MIXED CHARACTER SET OPL.	OPLEDIT detected decks on the program library that are in different character sets (63 and 64, for example).	Use Modify to recreate erroneous decks under one character set and retry.	OPLEDIT
MOD(S) TO MOD BEFORE THIS IDENT CARD.	A modification directive or a different IDENT directive refer to the current modname.	Choose a different modification name for the IDENT directive.	MODIFY
MODIFICATION ERRORS.	Modify has detected errors during the modification phase; fatal if D option is not selected.	Consult listing and correct specified errors.	MODIFY
MODIFICATION/DIRECTIVE ERRORS.	Modification and/or directive errors are encountered when debug mode is selected.	Consult listing and correct specified errors.	MODIFY
NAMES SEPARATED BY *.* IN WRONG ORDER.	Requested decks not in correct sequence.	Determine correct sequence and retry.	MODIFY, OPLEDIT
NO *IF IN PROGRESS.	An ELSE or ENDIF directive was encountered without a previous IF directive.	Check for omitted IF directive or unnecessary ELSE OR ENDIF directive.	MODIFY
NO DIRECTIVES.	Directives file empty. Fatal error.	Verify that directives file exists and is correctly positioned at BOI.	MODIFY, OPLEDIT
OPERATION ILLEGAL FROM ALTERNATE INPUT.	File manipulation attempted from other than original directives file.	Move file manipulation directives to original directives file.	MODIFY

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
OPLEDIT COMPLETE.	Informative message indicating that OPLEDIT has completed processing.	None.	OPLEDIT
OPLEDIT ERRORS.	Errors were encountered during OPLEDIT execution.	Consult output listing for description of errors.	OPLEDIT
OVERLAPPING MODIFICATION.	Line modified more than once.	Remove redundant line modifications.	MODIFY
PL ERROR IN DECK deckname.	An error was detected in the program library format during processing of deck named. Fatal error.	Replace or recreate erroneous deck.	MODIFY, OPLEDIT
PROGRAM LIBRARY EMPTY.	No information on file specified as program library. Fatal error.	Verify that program library file is available for Modify to manipulate.	MODIFY, OPLEDIT
RECORD NOT FOUND.	Modify was unable to locate requested record on file specified.	Verify that record exists on specified file.	MODIFY
RECURSIVE *IF,S ILLEGAL.	An IF directive was encountered while a previous IF range was still active (no ELSE or ENDIF encountered). Fatal error.	Check for missing ENDIF or ELSE directive or unnecessary IF directive.	MODIFY
REDUNDANT CONVERSION IGNORED.	An attempt was made to convert the program library file to a like character set (63 to 63 or 64 to 64). Conversion option set to zero.	Verify conversion mode desired.	MODIFY
RESERVED FILE NAME.	Operation attempted on a file name reserved by this utility.	Choose a nonreserved file name.	MODIFY, EDIT, OPLEDIT
S OPTION ILLEGAL WITH A, X, OR Q.	Source option not legal when A, X, or Q option is selected. Fatal error.	Remove S option from control statement and specify on separate modification.	MODIFY
TOO MANY OPL FILES.	More than 20 program library files declared.	Specify excess program libraries on subsequent Modify runs.	MODIFY

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
UNKNOWN DECK.	Unable to locate requested deck on program library.	Verify that deck name is correct.	MODIFY
UNKNOWN MODIFIER.	Modifier not in modification table for deck.	Determine correct modifier.	MODIFY
VALUE ERROR.	Value specified on IF or DEFINE directive is greater than 3777777B. Fatal error.	Select value less than or equal to 3777777B.	MODIFY
X OR Q ILLEGAL WITHOUT COMPILE.	Selection of X or Q option requires that a compile file name be selected.	Specify C option on Modify control statement (not C=0).	MODIFY
deckname - INVALID CS, 63 ASSUMED.	The lower byte of word 16B of the prefix table for the named deck on the program library does not contain 0000 or 0064.	If 64-character set is desired, the deck must be recreated.	MODIFY, OPLEDIT
deckname - MIXED CHARACTER SET DETECTED.	Upon editing the named deck on the program library, the character set was different from the character set of previously edited decks.	Recreate the deck under the desired character set.	MODIFY

OPLEDIT UTILITY

OPLEDIT is an NOS utility used in conjunction with Modify-formatted old program libraries (OPLs). The OPLEDIT routine is used to completely remove specified modification decks and modification identifiers from an OPL. It can also be used to extract the contents of specified modification sets on an OPL file.

The following are the OPLEDIT directives.

*EDIT	Edit deck
*PULLALL	Generate modification set
*PULLMOD	Reconstruct modification set
*PURGE	Remove modification set

The format of OPLEDIT directives is essentially the same for Modify directives (refer to section 2). The main difference is that OPLEDIT does not allow the user to change the prefix character. Therefore, the asterisk (*) must be used.

EDIT — EDIT SPECIFIED DECKS

The EDIT directive requests OPLEDIT to edit a program library deck and transfer it to the new program library. The deck names specified normally are the decks that contain the modification identifiers.

Format:

*EDIT $p_1, p_2, \ldots, p_n$

p_i A deck name or range of decknames in one of the following forms:

deckname

$\text{deckname}_a.\text{deckname}_b$

The first form edits a deck on the library; the second form requests a range of decks starting with deckname_a and ending with deckname_b.

If the deck names are in the wrong sequence, OPLEDIT issues the error message:

NAMES SEPARATED BY *.* IN WRONG ORDER.

If OPLEDIT fails to find one of the decks, it issues the message:

UNKNOWN DECK - deckname.

PULLALL — GENERATE MODIFICATION SET

The PULLALL directive allows the user to generate a modification set that contains the net effect of all current modification sets or all modification sets added after and including a specific modification set.

Formats:

*PULLALL

*PULLALL modname

modname First modset to be included; all modsets following modname are also included, provided modname appears in the edited deck.

For the first format, OPLEDIT builds a directive file suitable for submission to Modify using the *READ Modify directive. The file (specified by the M parameter on the OPLEDIT control statement) contains the net effect of all modifications currently applied to the program library. As such, all Modify IDENT directives are deleted and replaced by an IDENT ******* at the beginning of the file.

PULLMOD — RECONSTRUCT MODIFICATION SET

With the PULLMOD directive, the user can reconstruct one or more modification sets applied to edited decks. The structure of the original modset is maintained; that is, Modify IDENT directives are not changed or deleted as in the PULLALL directive.

Format:

*PULLMOD $\text{modname}_1, \text{modname}_2, \ldots, \text{modname}_n$

modname_i Modification name to be generated onto file specified by M parameter on OPLEDIT control statement.

PURGE — REMOVE MODIFICATION SET

The PURGE directive enables the user to completely remove the effects of a previous modification set or group of modsets from decks written on the new program library. The modification identifiers are no longer maintained in the history bytes (refer to Text Format, section 9) of the new program library.

Formats:

*PURGE modname

*PURGE modname, *

modname	Modification set to be removed.
*	Indicates that the modset and all subsequent modsets are to be removed, provided modname appears on the edited decks.

Note that it is not possible to remove modsets implicitly; that is, *PULLMOD A.B is illegal. Also, *PULLMOD A, * does not pull modset A and all modsets that follow (as on the *PURGE directive). Rather, it pulls modset A and modset *.

Modification names requested are removed only from decks edited. Modsets generated by OPLEDIT are in a form suitable for use by Modify as follows:

*READ, file, *

*READ, file, ident

That is, each modset is a separate record, with ident being the first line. The *PULLALL modset, if used, is the first record on the file. The file (specified by the M parameter) is returned before, and rewound after use.

OPLEDIT CONTROL STATEMENT

The control statement format is:

OPLEDIT(p_1, p_2, . . . , p_n)

p_i Any of the following in any order:

I	Use directive input from file INPUT. If the I option is omitted, file INPUT is assumed.
I=lfn_1	Use directive input from file lfn_1.
I=0	Use no directive input.
P	Use file OPL for the old program library. If the P option is omitted, file OPL is assumed.
P=lfn_2	Use file lfn_2 for the old program library.
P=0	Use no old program library.
N	Write new program library on file NPL.
N=lfn_3	Write new program library on file lfn_3.
N=0	Write no new program library. If this option is omitted, N=0 is assumed.
L	List output on file OUTPUT. If the L option is omitted, file OUTPUT is assumed.
L=lfn_4	List output on file lfn_4.
L=0	List no output.
M=lfn_5	Write output from *PULLMOD and *PULLALL directives on file lfn_5. If M is omitted, M=MODSETS is assumed.
LO=x	Set list options x; each bit in x, if set, turns on the corresponding option.

001	Errors
002	Directives
004	All other input statements
010	Modifications made
020	Directives processed from the program library
040	Deck status
100	Directory lists
200	Inactive statements
400	Active statements

If this option is omitted, x=177 is assumed (that is, the first seven options listed).

F	Modify all decks.
D	Debug; ignore errors.
U	Generate *EDIT directives for all decks.
U=0	Generate no *EDIT directives. If the U option is omitted, generate *EDIT directives for common decks.
Z	The OPLEDIT control statement contains the input directives following the terminator; the input file is not read. This eliminates the need to use a separate input file for the directives when only a few directives are needed. The first character following the control statement terminator is the separator character. If Z is omitted, the control statement does not contain the input directives.

NOTE

Do not place another terminator after the directives.

OPLEDIT EXAMPLES

Figure C-1 illustrates the four OPLEDIT directives.

```
batch,45000
$RFL,45000.
/get,mainpl
/catalog,mainpl,r
          CATALOG OF MAINPL         FILE     1
    REC   NAME     TYPE           LENGTH   CKSUM     DATE

      1   DECK1    OPL   (64)         61     3171   76/01/22.
           MOD1      MOD4

      2   DECK3    OPL   (64)         37     2333   76/01/21.
           MOD1      MOD4

      3   DECK2    OPL   (64)         60     5455   76/01/22.
           MOD1      MOD2      MOD3      MOD4

      4   DECK4    OPL   (64)         47     5063   76/01/23.
           MOD4

      5   DECK5    OPLC  (64)         27     6354   76/01/23.
      6   OPL      OPLD               13     3706   76/01/23.

      7   * EOF *       SUM =        311
1
 CATALOG COMPLETE.
/opledit,p=mainpl,m=mods,lo=1,n=newpl
? *purge mod4,*
? *pullmod mod2,mod3
? *pullall mod1
? *edit deck1.deck4
?
 OPLEDIT COMPLETE.
/catalog,newpl,r
          CATALOG OF NEWPL          FILE     1
    REC   NAME     TYPE           LENGTH   CKSUM     DATE

      1   DECK1    OPL   (64)         37     7732   76/01/22.
           MOD1

      2   DECK3    OPL   (64)         34     3117   76/01/21.
           MOD1

      3   DECK2    OPL   (64)         55     5026   76/01/22.
           MOD1      MOD2      MOD3

      4   DECK4    OPL   (64)         44     0216   76/01/23.
      5   OPL      OPLD               11     4076

      6   * EOF *       SUM =        225
1
 CATALOG COMPLETE.
/primary,mods
$PRIMARY,MODS.
```

Figure C-1. OPLEDIT Examples (Sheet 1 of 2)

```
/lnh,r
*******
*IDENT    *******
*DECK     DECK1
*D,1
***   MAIN PROGRAM, DECK DECK1.
*I,2
      COMMON JOT
*I,3
      CALL SUB3
      IF(JOT.EQ.3)PRINT*,"TIME-SHARING JOB."
      IF(JOT.NE.3)PRINT*,"BATCH JOB."
*DECK     DECK3
*I,Ø
*WEOR
*D,1
***   SUBROUTINE 2, DECK DECK3.
*DECK     DECK2
*I,Ø                                                  PULLALL directive
*WEOR
*D,1
***   SUBROUTINE 1, DECK DECK2.
*I,3
*     CALL SUBROUTINE SUB2
*     IN DECK DECK2.
*I,7
***   END DECK2.
--EOR--
MOD2
*IDENT    MOD2
*DECK     DECK2
*D,MOD1.3
*RESTORE,7                                            PULLMOD directive
--EOR--
MOD3
*IDENT    MOD3
*DECK     DECK2
*RESTORE,MOD1.3
--EOR--
```

Figure C-1. OPLEDIT Examples (Sheet 2 of 2)

INDEX

COMMENT SHEET

MANUAL TITLE CDC NOS Version 1 Modify Reference Manual

PUBLICATION NO. 60450100 REVISION C

FROM: NAME: ______

BUSINESS ADDRESS: ______

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

CUT ALONG LINE

AA3419 REV. 7/76 PRINTED IN U.S.A.

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY
CONTROL DATA CORPORATION
Publications and Graphics Division
ARH219
4201 North Lexington Avenue
Saint Paul, Minnesota 55112

CUT ALONG LINE

FOLD FOLD

MODIFY CONTROL STATEMENT PARAMETERS

MODIFY(p_1, p_2, . . . , p_n)

A	Presence of A causes compressed compile file.
C	Compile file output; COMPILE if C or omitted. No compile file if C=0. Otherwise, output on file named (C=lfn).
CB	COMPASS binary output file; used with Q and X options only. Output on LGO if CB. No binary if CB=0. Otherwise, output on file named (CB=lfn).
CG	COMPASS get text option; used with Q and X options only. Systems text on SYSTEXT if CG. No systems text if CG=0. Defined by CS option if CG is omitted. Otherwise, systems text on file named (CG=lfn).
CL	COMPASS list output; used with Q and X options only. Short list if CL=0 or omitted. Output on file OUTPUT if CL. Otherwise, list output on file named (CL=lfn).
CS	COMPASS systems text; used with Q and X options only. Systems text on SYSTEXT overlay if omitted or CS. No systems text if CS=0; otherwise, systems text on file named (CS-lfn).
CV	Program library character set conversion. None if CV is omitted; 63 to 64 if CV=64; 64 to 63 if CV=63.
D	Debug option. Directive error or fatal error causes job abort if D is omitted. No job abort for directive errors if D is used.
F	Full edit. If omitted, deck editing determined by U option or by EDIT directives. If F is specified, all decks are edited and written on compile file, new program library, and source file.
I	Directives input. If omitted, directives and corrections on INPUT. If I=0 there is no input file. Otherwise, on named file (I=lfn).
L	List output. Omitted or L, listings on OUTPUT. L=lfn, output to named file.
LO	List options. Omitted or LO, options E, C, T, M, W, D, and S are selected. Otherwise, LO=c_1, c_2. . . c_n to a maximum of seven options (AECDIMST or W).
N	New program library. Omitted or N=0. No new library. N, output on NPL. N=lfn, output to named file.
NR	No rewind on compile file. Omitted, compile file rewound before and after MODIFY run. RN, no rewinding.
P	Program library input. Omitted or P, library on OPL. P=lfn, library on named file. P=0, no program library input file.
Q	Execute assembler or compiler; no rewind of directives file or list output file. Omitted or Q=0, assembler or compiler not automatically called. Q, Modify sets A parameter and LO=E and calls COMPASS. This option enables CB, CG, CL, and CS options. If Q=lfn, Modify calls assembler on lfn.
S	Source output (illegal if A, Q, or X selected). Omitted or S=0, no source output. S, output on SOURCE. S=lfn, output on named file.
U	Update edit. Omitted, editing set by F or by EDIT directives. F takes precedence over U. If U, only decks changed (named on DECK directives) are edited and written on compile file, new program library, and source file.
X	Execute assembler or compiler; same as Q except directives file and list output file are rewound.
Z	Directives on Modify card. Omitted, directives are next record on INPUT or identified by one option. Z, directives follow parameters on Modify. A separator bar separates two directives.