СООБЩЕНИЯ
ОБЪЕДИНЕННОГО
ИНСТИТУТА
ЯДЕРНЫХ
ИССЛЕДОВАНИЙ
ДУБНА

E10 - 10542

D.C.Marinescu, V.P.Shirikov

A CDL WRITTEN INTERCOM EDITOR

1977

E10 - 10542

D.C.Marinescu,* V.P.Shirikov

A CDL WRITTEN INTERCOM EDITOR

* On leave of absence from the Institute for Atomic Physics, Bucharest, Romania.

Маринеску Д.К., Шириков В.П.

E10 - 10542

Описание программы-редактора системы ИНТЕРКОМ
на языке CDL

В работе содержится формализованное описание с помощью языка CDL (Compiler Description Language) программы EDITOR, являющейся главной частью телекоммуникационной программной системы ИНТЕРКОМ для обслуживания терминалов ЭВМ CDC серии 6000 и CYBER. EDITOR позволяет накапливать и редактировать текстовые файлы, выполнять целый ряд управляющих приказов системе математического обеспечения ЭВМ. Его формализованное описание дает возможность внедрения аналогичной программы на ЭВМ других типов (в том числе малых и средних ЭВМ).

Работа выполнена в Лаборатории вычислительной техники и автоматизации ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна 1977

Marinescu D.C., Shirikov V.F.

E10 - 10542

A CDL Written INTERCOM EDITOR

This report contains the formalized description of the program EDITOR, which is the most important part of the teleprocessing system INTERCOM driving terminals for CDC series 6000 and CYBER computers; CDL (Compiler Description Language) is used for this description. EDITOR is a tool for the text file acquisition and modifications; it also gives the possibility to execute some commands to the computer software. The EDITOR machine independent description may be used for the implementation of EDITOR-like programmes for different types of computers (in particular, "small" computers).

The investigation has been performed at the Laboratory of Computing Techniques and Automation, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna 1977

INTERCOM is a teleprocessing system which when used in conjuncture with most CDC operating systems (SCOPE, KRONOS, NOS/BE) provides a time sharing access to CYBER and 6000 series computers.

Since some scientifical communities are using besides a CDC machine, some other medium to large scale computers a problem is to transfer some of the facilities available on the first to some other computers; the INTERCOM system is a likely candidate since a unified teleprocessing system is highly convenient for users.

In such a case a small computer can act as a concentrator driving a number of terminals; an interactive system accepting as input INTERCOM commands and converting them into target machine control statements or executing itself some of the requested functions (for example the functions related to text editing) must be designed.

In this case all the system processors but for a small interface monitor (providing the necessary linkage to the host machine supervisor, mainly for I/O actions) can be written in a machine independent fashion using CDL as an implementation language.

The text editor is a good example to illustrate the advantage of using CDL; it is sophisticated enough to make a non-trivial example and in the same time it is rather easy to follow the ideas behind its design.

The present paper is a preliminary report on a project under completion at the JOINT INSTITUTE FOR NUCLEAR RESEARCH.

1. THE PHILOSOPHY OF THE TEXT EDITOR

The text EDITOR is a system processor which gives to a user sitting at a terminal the possibility to create, maintain and examine a file, the editor file to be either submitted for processing or simply saved as a permanent file for some later use.

Logically the edit file must be considered as a collection of text lines, each line being uniquely identified by a line number.

When structuring the edit file the simplest approach could be to keep in primary storage a directory of the edit file (an entry into the directory would contain the line number and a pointer to the text of line) and also the text of each line. Since we always have a limited amount of primary storage available such an ideal organization is ruled out.

The opposite approach is to keep in storage only the directory and to write on a backup device (disk for example) each line as it is entered. The edit file could be organized as a direct access file; deletion and insertion could be greatly simplified in this case. The major disadvantage of such an organization is the need to perform an I/O operation whenever the content of a line must be examined; imagine the amount of overhead when a rather big file (tens of thousands of lines) is to be examined for the occurrence of a given string of characters.

If the file is organized as an index sequential file but also using a line as a physical record no substantial improvement is foreseen.

The first step to increase efficiency is to block a number of lines and to exchange information with the backup device at a block level.

We must keep in core a master file directory (MFD) and we must have an entry in this directory for each block. Such an entry must contain:

- the smallest line number for a line in the block - FLN, first line in the block
- the highest line number for a line in the block - LLN, last line in the block.
- the amount of free space available in the block - BFS, block free space.

At any moment of time the number of entries in MFD is equal to HBN (highest block number).

Each time we need to access a certain line with the line number LN we search first the MFD and find the block index (BLOCK ID) such that:

FLN< LN< LLN

If the index of the block currently being in primary storage (BCC -- block currently in core) is not equal to BLOCK ID then a 'BLOCK FAULT CONDITION' occurs and an external action READ BLOCK+ID, reads the block with ID-BLOCK ID from the backup device.

As far as the master file directory is concerned, its structure is depicted in Figure 1.

In create mode or when editing a file, MHBN (MAX HBN), denotes the maximum number of entries which can be accepted, i.e., HBN< MHBN. When working in command mode, updating the existing edit file, we have the provision of another OWMFD entries in the overflow area of MFD, when addition of new blocks to the edit file is desired. Here OWMFD is given by: OWMFD = MFDSIZE - MHBN. The size of MFD is: MFDSIZE (ENTRIES) and MAXMFD-MINMFD (WORDS).

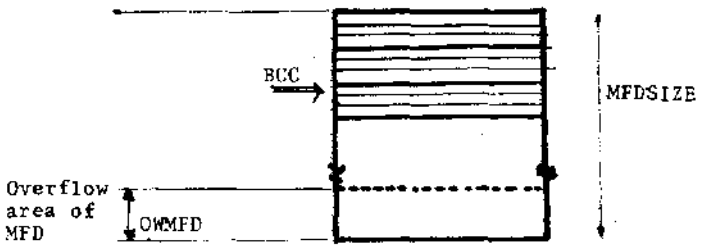

Figure 1. Master file directory (MFD) structure.

The main actions related to the MFD management are: compute MFD entry address, get (put) MFD entry, find block, insert (delete) entry in MFD.

MFD is always kept in order so that its K-th entry contains the descriptor of the K-th block. Whenever we delete a block, we move up the entries for all blocks succeeding the deleted one and decrease HBN. When we add a new block, the J-th block, then we move down entries for J<k HBN and increase HBN.

A block has the following structure:

A.) A block directory of fixed size, BDS (block directory size). The maximum number of entries in block directory is MEN (maximum entry number) and each entry has a fixed size, ES (entry size). The first entry has a special meaning, it is a block identificator. We also have a block directory overflow area so that:

BDS=MEN+OVE

In this expression OVE denotes the number of overflow entries in the directory.

For a machine with a word size of 10 characters, like CDC 6500 a typical choice could be:

MEN=16, OVE=4, ES=2.

Each entry in the block directory contains a line number in character format (up to six digits) and a pointer to the text of the line, inside the block. The block directory is kept in order so that a binary search can be performed.

B.) The text part of the block. Since the block is of a fixed size, BLKSIZE (block size), a variable number of lines of size LINESIZE (line size) can be accomodated. The line size has a default value but this value can be altered with the aid of the format command. A global variable BFS (block free space) shows the amount of space still available in the block.

A pointer to the next available location in the text area of the block is maintained, LINEPTR (line pointer). The initial value of this pointer is:

LINEPTR= BDS*ES+1

Each line of text is entered at a fullword boundary; the characters are packed in each word leaving the leftmost character of the word, free for marking purpose. The last word of a line is eventually padded with special fill characters. For CDC 6500 the number of characters per word could thus be, CHARPERWD=9; the

last word of a line will be marked with a minus sign in the leftmost character.

Three actions, ADDTO, STACK and STACK LAST, are designed for such an activity.

It should be pointed out again that all the global pointers we are refering to, are given in number of either words or entries but still the text of the line is stored in a packed format (a number of characters to each word).

An option must be made if the text of the line should be stored in a compressed format or not. In our option there is a tremendous saving which occurs especially for oversized files but still the use of a compressed format leads occasionally to difficult situations; for example if a text replacement is to take place (N1 characters of text are to be replaced by N2 other characters) and if N2>N1, a time consuming function to extend the space of each line, must be built in. The really bad situation occurs when there is no more available space in the block and we have to use the overflow area or even worse, we have to move lines from one block to another.

Pondering over all these inconvenients we finally decided to use a compressed format. Whenever we have to add a new line of size ALS (actual line size), the line pointer is incremented as:

LINEPTR = LINEPTR + ALS

and the amount of free space is decreased as:

BFS = BFS - ALS

The edit block has a more complicated structure (see Figure 2) and actions to access and modify its elements (the BLOCK ID, the block directory and the text section of the block) must be designed.

We have adopted the same strategy as in the case of MFD; block directory has an overflow area which we use when we have to add new lines to the block.

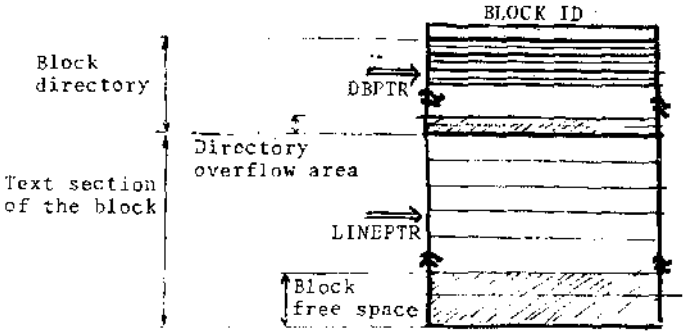

Figure 2. Edit file block structure.

When the edit file is initially created (either in increment mode or with the edit command) the DBPTR (directory block pointer) and the LINEPTR (pointer in the text section of the block) are updated observing the conditions:

DBPTR<MEN and LINEPTR< BLKSIZE

Whenever we wish to add a new line we first check if an entry in block directory is still available and if so we check if the space available in the text section of the block is enough for a full line (a line having the maximum size declared with the format command or its default value), BFS>LINE SIZE.

When a line is deleted from the block, it is logically deleted and this means that only its entry in block directory is marked. Only when we reach a limit situation (no more available space in the text section of the block) we perform an expensive operation of block compaction. In such a case we first compact the directory and then we compact the text, by reading only the text of undeleted lines into a buffer (NEWTEXT) and then restoring the block.

Sometimes we have to insert (either with the ADD command or in the ADD regime) a number of lines into the middle of a partially full block and even the block compaction does not save us, providing enough space in the text section of the block.

In such a case a block splitting operation occurs. We have designed a strategy to split the block (either in half or at the

place where insertion is to be made) and to create two new blocks. Obviously the amount of overhead related to the splitting of the K-th block is high; we have to update the MFD and the BLOCK ID'S for the blocks with the index J:

K+1 < J< HBN

The moment has come to stress the great significance of a proper choice for: MFD size, block size, block directory size. All of them are given as parameters so that can be easily adjusted; measuring system performances we can tune them in such a way that the unpleasant situation of block splitting will be very unfrequently encountered.

Let us briefly explain how different functions of the text editor will work in this scheme.

1. EDIT and SAVE commands.
Two external actions, ≠READ INPUT RECORD+RECID≠ and ≠WRITE OUTPUT RECORD+RECID≠, perform the actual I/O operations (both are low level language written routines).

When editing a file its records are sequentially read into a buffer with the aid of the first routine.

A switch indicating if the previous editor file has been saved is first checked, then a master file directory is created.

Using the information concerning the structure of the permanent file, blocks of the edit file are created; as soon as each block is filled up, it is written on the backup device using the ≠WRITE BLOCK≠ external action. An entry in the master file directory for the block is inserted. When the end of file is encountered on the input file, a partially filled block is written in the edit file and the editor returns to the command mode.

The SAVE command is used to convert from the editor file structure to the structure of either local or permanent file. It uses the master file directory to get the edit file blocks in order, deblocks them and then reblocks to local fine records writting them to the backup device.

2. ADD and DELETE commands.
The first thing to remember when thinking about addition is that in the text section of the block lines need not be in order. Due

to the fact that the block directory is kept in order and each line number has its associated pointer to the text of the line we normally add a new line at the first available location in the block, we only reorder the block directory to insert the entry for the new line at the proper place.

The critical problem occurs when in the block there is no more available space; then we apply the block splitting strategy. Line deletion is performed at the logical level; initially the line descriptor in the block directory is marked and then the block directory is undergoing a compactization process.

3. The resequence command determines some action to be taken upon the line numbers; they are recomputed and updated in the block directory. Also the master file directory is updated.

The other commands are implemented in a straightforward way and no special comments are required.

A few words must be said about keyword processing; keywords are either the names of EDITOR commands or of INTERCOM commands or other names with a precise meaning in the context of a command.

We took advantage of the fact that no keyword is longer than eight characters; for our implementation we have defined strings of fixed length (nine characters, to fit in one machine word), containing the keyword (leftjustified) and with the rightmost character denoting the length of the keyword. If length is not specified it means that every prefix of the full keyword can be accepted as a keyword, e.g., for LIST we can accept L, LI, LIS or LIST; otherwise we must have a matching for the number of characters specified by the length attribute, e.g., for PATCH we must have a full matching of 5 characters (this is the case of INTERCOM command keywords while for EDITOR command keywords we can accept any prefix).

In this preliminary version of the EDITOR no provision has been made for reentrancy.

ABREVIATIONS USED IN THE TEXT AND IN THE CDL PROGRAM

CDL = complier description language
FLB = first line in the block
LLB = last line in the block
FNL = first line number in the file
LLN = last line number in the file
CLINE = current line number
FLINE = first line upon which the action is to be performed
MFD = master file directory
BLOCK = the area in core where one block of edit file resides at a time
NEWTEXT = the area in core used for compaction of the BCC
NEWLINES = a core area used to hold line numbers when splitting a block
BCC = block currently in core
BLOCK ID = an identificator which uniquely identifies each file block
DBPTR = the pointer in the block directory
LINEPTR = the pointer in the text section of the block
MFDSIZE = the size of MFD in number of entries
HBN = highest block number in the edit file (actually its size)
MHBN = maximum number of entries in master file (equals: MHBN1 for CREATE mode and MFDSIZE for command mode)
BDS = block directory size
MEN = maximum number of entries in block directory (equals MEN1 for CREATE mode and BDS for command mode)
BFS = block free space
LINESIZE = the maximum number of characters in a line
SL = starting line upon which the action is to be performed
EL = ending line upon which the action is to be performed
SC = starting column when searching for the occurrence of a string
EC = ending column when the searching for the occurrence of a string
ERRSW = error switch which can inhibit the execution of a command
SAVED FLAG = switch indicating if the edit file has been saved
BUFFER = a core area used as a I/O buffer when reading or writing from or to a permanent or local file record.

INPTEXT = an area used to hold a string of characters used as a search text; it also contains the input string in a text replacement command.

OUTTEXT - an area to hold the new text in a text replacement command

CHARACTERS1 = since the characters of a line are packed in a number of words, when text searching occurs we use this list to decode the characters packed in a word.

CHARACTERS2 = the same use as the previous list but for the output string

KEYWORD = a global variable used to hold a keyword

WORD = global variable used when packing and unpacking characters.

MSG = message; this is an action to print error messages; its text is supplied as a parameter. Some messages are intended only as a debugging tool for the EDITOR.

DFLINE - default value of the first line number

INCR = the value of the increment for line number computation

DINCR = default value for the increment.

ACKNOWLEDGEMENTS

The authors wish to express their gratitude to Professor C.H.A.KOSTER and to Mr. J.JACKEL of the TECHNICAL UNIVERSITY OF WEST BERLIN; Professor KOSTER provided us with the information concerning CDL and Mr. JACKEL sent us the version of the CDL complier complier we have been using so far.

REFERENCES

1. C.H.A.Koster. A compiler compiler, MCA report, 121, 1971.
2. C.H.A.Koster. Using the compiler compiler, chapter 4 in Compiler construction , vol. 21 of lecture notes on computer science, Springer 1974.
3. J.C.Jackel. Bootstrap eine CDL compiler auf die CDC 6500, Technische Universität Berlin, April 1975.
4. J.C.Jackel. Intercom Reference manual, CDC Publication, 603007200.

2. SOURCE LISTING OF THE EDITOR

```
(   1. GENERAL ENVIRONEMENT )

(   1.1. INTERFACE WITH THE MACHINE   )


*LONG*
*EXTERNAL**ACTION* RESYM , PRSYM , READ BLOCK , WRITE BLOCK ,
                   READ INPUT RECORD , WRITE OUTPUT RECORD ,
                   INTERCOM ACTION , EXIT TO INTERCOM MODE ,
                   CONNECT FILE , READ THE SEQUENCE NUMBER OF LINE ,
                   INTERRUPT PROCESSING ,RUN PROGRAM,WAIT REPLY.,RUN P
                   PUSYM,CHECK FILE NAME .


*MACRO**POINTER*   NIX=111 , MINUS CHAR=77 , SPACE CHAR = 105 ,
                   EOL CHAR = 113 , ACCENT CHAR = 104 ,
                   QUOTE CHAR=87 , NULL CHAR = 94 , EOF CHAR = 115 .
*MACRO**POINTER*
L CHAR=39 , T CHAR=47 , D CHAR=31 , S CHAR=46 , DOLLAR = 72 ,
SEMICOLON = 83 , PERCENT=80 , TEXT DELIMITER=78 , FILL CHAR = 0 ,
COMMA=79 .


*MACRO**POINTER* CHARSIZE = 64 , CHARPERWD = 9 , DINCR = 10 ,
                 DFLINE = 100 , MEN1 = 16 , OVE = 4 , ES=2 ,
                 TEXTSIZE = 144 , INPUTRECORDSIZE = 200 .


(   1.2. DATA STRUCTURES AND THEIR ACCESS   )


*MACRO**POINTER* MINBLK = 100001 , BLKSIZE = 110000 ,
                 MINNEWTEXT = 201001 , MAXNEWTEXT = 210000 ,
                 MINNEWLINES= 300001 , MAXNLWLINES= 300100 ,
                 MINMFD = 400001 , MAXMFD = 400500 ,
                 MINBUFFER = 500001 , MAXBUFFER = 510600 ,
                 MININP = 600001 , MAXINP = 600004 ,
                 MINOUT = 700001 , MAXOUT = 700004 ,
                 MINCHARACTERS = 800001 ,MAXCHARACTERS = 800100 ,
                 MINCHARACTERS2 = 900001 , MAXCHARACTERS2 =900100 .


*POINTER* DBPTR , LINEPTR , BCC , INCNT , OUTCNT , INPSW , OUTSW ,
         WORD,STOCK,CHAR,CHAR1,CHAR2,KEYWORD,CHARCOUNT,MINUSONE .


*POINTER* SW1,SW2,SW3,SW4,SW5,SW6,SW7,SW8,SW9,
         PSUP,PVETO,PINCR,POVERWRITE,PUNIT,PSEQUENCE,PNOSEQUENCE,
         PMERGE,PINCREMENTSW,TABCHAR,PCH,PTAB1,PSHOW
         PTAB2,PTAB3,PTAB4,PTAB5,FLINE,INCR,FLN,LLN,FLB,LLH,
         CLINE,BCC,ERRSW,CREATEFLAG,POS,MEN,BDS,LAST COLUMN,
         EDIT FILE EMPTY,RLCNBR,REPLY,PFILE,PROCESSOR,FILE NAME.
```

```
*LIST*
        BLOCK [MINBLK : BLKSIZE ] ,
        NEWTEXT [ MINNEWTEXT : MAXNEWTEXT ] ,
        NEWLINES [ MINNEWLINES : MAXNEWLINES ] ,
        MFD [ MINMFD : MAXMFD ]
        BUFFER [ MINBUFFER : MAXBUFFER ] ,
        INPTEXT [ MININP : MAXINP ] ,
        OUTTEXT [ MINOUT : MAXOUT ] ,
        CHARACTERS1 [ MINCHARACTERS : MAXCHARACTERS ] ,
        CHARACTERS2 [ MINCHARACTERS2 : MAXCHARACTERS2 ] .


*MACRO**ACTION*
        PUT FIRST LINE  = MFD [ *1* ] := *2* ,
        GET FIRST LINE  = *2* := MFD [ *1* ] ,
        PUT LAST LINE   =  MFD [ *1**1 ] := *2* ,
        GET LAST LINE   =  *2* := MFD [ *1**1 ] ,
        PUT FREE SPACE  = MFD [ *1**2 ] := *2* ,
        GET FREE SPACE  = *2* := MFD [ *1**2 ] .


*MACRO**ACTION*
        GET LINE NR  = *2* := BLOCK [*1*] ,
        PUT LINE NR   = BLOCK [*1*] := *2* ,
        GET LINE POINTER  = *2* := BLOCK [*1* + 1] ,
        PUT LINE POINTER  = BLOCK [*1**1] := *2* .


(   1.3. GENERAL MACROS AND FLAGS   )


(   1.3.1. MACROS   )


*MACRO**ACTION*
GET     = *3* := *1*[*2*] ,
PUT     = *1*[*2*] := *3* ,
MAKE    = *1* := *2* ,
SET     = *1* := *2* ,
MARK    = *1* := -*1* ,
ADD     = *3* := *1* + *2* ,
SUBTR   = *3* := *1* - *2* ,
ADDMULT= *4* := *1**2* + *3* ,
DIVREM = *3*  := *1* *#/#* *2* ; *4* := *1* - *2***3* ,
INCR    = *1* := *1* + 1 ,
DECR    = *1* := *1* - 1 ,
UNPACK = *2* := *1* *#/#* CHARSIZE ,*3* := *1* - *2**CHARSIZE .


(   1.3.2. MACRO FLAGS   )


*MACRO**FLAG*
MARKED    = *1*  <  0 ,
LESS      = *1*  <  *2* ,
LSEQ      = *1* *#NOT GREATER#* *2* ,
EQUAL     = *1* = *2*  ,
WAS LETTER = *1* > 27 *#AND#* *1* < 54 ,
WAS DIGIT  = *1* > 53 *#AND#* *1* < 64 ,
WAS LETGIT = *1* > 27 *#AND#* *1* < 64 ,
```

1.4. MACRO POINTERS

1.4.1. KEYWORDS AS MACRO POINTERS

```
*MACRO**POINTER*
TRUE        = *#TRUE#* ,
FALSE       = *#FALSE#* ,
FORMAT      = *#FORMAT   #* ,
TAB         = *#TAB      #* ,
SHOW        = *#SHOW     #* ,
CH          = *#CH       #* ,
ALGOL       = *#ALGOL    #* ,
BASIC       = *#BASIC    #* ,
COBOL       = *#COBOL   3#* ,
COMPASS     = *#COMPASS 3#* ,
FORTRAN     = *#FORTRAN  #* ,
CREATE      = *#CREATE   #* ,
SUP         = *#SUP      #* ,
EDIT        = *#EDIT     #* ,
SAVE        = *#SAVE     #* ,
NOSEQ       = *#NOSEQ    #* ,
OVERWRITE   = *#OVERWRIT #* ,
MERGE       = *#MERGE    #* ,
ALL         = *#ALL      #* ,
LAST        = *#LAST     #* ,
UNIT        = *#UNIT     #* ,
VETO        = *#VETO     #* ,
LIST        = *#LIST     #* ,
ADD         = *#ADD      #* ,
DELETE      = *#DELETE   #* ,
RESEQ       = *#RESEQ    #* ,
RUN         = *#RUN      #* ,
FILE        = *#FILE     #* ,
NOEX        = *#NOEX     #* ,
FIN         = *#FIN      #* ,
YES         = *#YES      #* ,
BY          = *#BY       #* ,
ALTER       = *#ALTER   5#* ,
ATTACH      = *#ATTACH  6#* ,
ASSETS      = *#ASSETS  6#* ,
BATCH       = *#BATCH   5#* ,
BKSP        = *#BKSP    4#* ,
COMBINE     = *#COMBINE 7#* ,
COMPARE     = *#COMPARE 7#* ,
CONVERT     = *#CONVERT 7#* ,
COPYN       = *#COPYN   5#* ,
CATALOG     = *#CATALOG 5#* ,
CONNECT     = *#CONNECT 5#* ,
COPY        = *#COPY    4#* ,
COPYCF      = *#COPYCF  6#* ,
COPYBF      = *#COPYBF  6#* ,
COPYBR      = *#COPYBR  6#* ,
COPYCR      = *#COPYCR  6#* ,
COPYBCD     = *#COPYBCD 7#* ,
COPYSBF     = *#COPYSBF 7#* ,
DISCONT     = *#DISCONT 7#* ,
```

```
EFL         = *#EFL     3#* ,
ETL         = *#ETL     3#* ,
FETCH       = *#FETCH   5#* ,
FILES       = *#FILES   5#* ,
LOCK        = *#LOCK    4#* ,
MAP         = *#MAP     3#* ,
MESSAGE     = *#MESSAGE 7#* ,
PAGE        = *#PAGE    4#* ,
Q           = *#Q       1#* ,
RETURN      = *#RETURN  6#* ,
RENAME      = *#RENAME  6#* ,
REWIND      = *#REWIND  6#* ,
REQUEST     = *#REQUEST 7#* ,
SKIPB       = *#SKIPB   5#* ,
SAVEFL      = *#SAVEFL  6#* ,
SWITCH      = *#SWITCH  6#* ,
SCREEN      = *#SCREEN  6#* ,
PURGE       = *#PURGE   5#* ,
STORE       = *#STORE   5#* ,
SKIPF       = *#SKIPF   5#* ,
SITUATE     = *#SITUATE 7#* ,
SEND        = *#SEND    4#* .
```

(1.4.2. ERROR MESSAGES AS MACRO POINTERS)

```
*MACRO**POINTER*
ERR1  = *#ADD WONT REPLACE OR BYPASS LINES#* ,
ERR2  = *#ERR  CANT FIND FILE#* ,
ERR3  = *#ERR  CH MUST SPECIFY COUNT LESS THAN 511#* ,
ERR4  = *#ERR  COL OR UNIT SPECIFIED BUT NOT  TEXT1#* ,
ERR5  = *#ERR  COMPILER NAME REQUIRED#* ,
ERR7  = *#ERR  FILE NAME MUST BE ALPHANUMERIC#* ,
ERR8  = *#ERR  FILE NAME REQUIRED#* ,
ERR9  = *#ERR  ILLEGAL COLUMN RANGE#* ,
ERR10 = *#ERR  ILLEGAL LINE RANGE#* ,
ERR11 = *#ERR  INCREMENT MUST BE GREATER THAN ZERO#* ,
ERR12 = *#ERR  LINE NUMBER GREATER THAN 99999#* ,
ERR13 = *#ERR  LINE NUMBER REQUIRED#* ,
ERR14 = *#ERR  LINE NUMBER OUT OF SEQUENCE#* ,
ERR15 = *#ERR  NO INFORMATION IN EDIT FILE#* ,
ERR16 = *#ERR  OVERWRITE ILLEGAL ON PERMANENT FILE#* ,
ERR6  = *#ERR  TEXT2.REQUIRED#* ,
ERR18 = *#ERR  INVALID EDITOR COMMAND#* ,
ERR17 = *#ERR  COLUMN SPECIFICATION INCOMPLETE#* ,
ERR19 = *#ERR  ILLEGAL LINE NUMBER#* ,
ERR20 = *#ERR  TOO MANY DIGITS#* ,
ERR21 = *#ERR  UNRECOGNIZABLE PARAMETER#* ,
ERR22 = *#ERR  RESERVED FILE NAME#* ,
ERR23 = *#ERR  TABS TOO BIG OR OUT OF ORDER#* ,
ERR24 = *#ERR  TOO MANY PARAMETERS#* ,
ERR25 = *#ERR  TOO MANY TABS#* ,
ERR26 = *#ERR  TEXT1 MUST HAVE AT LEAST ONE CHARACTER#* ,
ERR27 = *#ERR  LOCAL FILE ALREADY EXISTS#* ,
ERR28 = *#ERR  DISK ERROR ON EDIT FILE#* ,
ERR29 = *#ERR  LINE TRUNCATED TO 511 CHARACTERS#* ,
ERR30 = *#MUJ SYSTEM ERROR#* ,
```

```
ERR31 = *≠NO SUCH LINES≠* ,
ERR32 = *≠TABS LESS THAN CH CLEARED≠* ,
ERR33 = *≠WARNING  EDIT FILE NOT SAVED≠* ,
ERR34 = *≠TEXT TRUNCATED TO 20 CHARACTERS  AUTOMATIC VETO SET≠* ,
ERR35 = *≠TRUNCATED FROM LONG LINE≠* ,
ERR36 = *≠NO SEQUENCE NUMBERS IN EDIT FILE≠* ,
EE1   = *≠EDITOR ERROR  INVALID CODE FOR M≠* ,
EE2   = *≠EDITOR ERROR  MFD ENTRY OUT OF RANGE≠* ,
EE3   = *≠EDITOR ERROR  MFD OVERFLOW≠* ,
EE4   = *≠EDITOR ERROR  INVALID ENTRY IN BLOCK DIRECTORY≠* ,
EE5   = *≠EDITOR ERROR  BLOCK DIRECTORY OVERFLOW≠* ,
EE6   = *≠EDITOR ERROR  WRONG BINARY SEARCH≠* ,
EE7   = *≠EDITOR ERROR  LIST FULL≠* ,
EE8   = *≠EDITOR ERROR  INVALID KEYWORD≠* .


(   1.5. ACTIONS TO DECODE CHARACTERS PACKED IN A WORD  )


*ACTION* DECODE WORD INTO CHARACTERS,
         DECODE KEYWORD INTO CHARACTERS.

DECODE WORD INTO CHARACTERS*LISTNAME*POINTER*CHARLIST-X-Y-K
   GET*LISTNAME*POINTER*WORD
    ,MAKE*K*9 ,
    ABC: UNPACK*WORD*X*Y , PUT*CHARLIST*K*Y , MAKE*WORD*X ,
         DECR*K,LSEQ*1*K,  *ABC : .


DECODE KEYWORD INTO CHARACTERS*KWD*LISTNAME*Q-X-Y-K:
  MAKE*K*9,MAKE*WORD*KWD,
   ABC: UNPACK*WORD*X*Y,PUT*LISTNAME*K*Y,
        MAKE*WORD*X,DECR*K,LSEQ*1*K,  *-BC :
                                GET*LISTNAME*9*Q,SUBTR*Q*54*Q .
```

```
(     1.6.  ACTIONS TO PROCESS INPUT AND OUTPUT STRINGS    )

*ACTION* GET STRING , GET INPUT STRING , GET OUTPUT STRING .


GET STRING+LISTNAME+POINTER+MAXLIST+SW :
   MAKE+SW+1,MAKE+POINTER+1,MAKE+CHARCOUNT+1,
  (ABC: NEXTCHAR+CHAR,LSEQ+POINTER+20,
 (EQUAL+CHAR+TEXT DELIMITER,STACK LIST+LISTNAME+MAXLIST+POINTER:
  ADD TO+CHAR+LISTNAME+MAXLIST+POINTER, :ABC) ;
  MSG + ERR34 .

GET INPUT STRING : GET STRING+INPTEXT+INCNT+INPMAX+INPSW ,
                   ( EQUAL+INCNT+1,MSG + ERR26 : ) .
GET OUTPUT STRING : GET STRING+OUTTEXT+OUTCNT+OUTMAX+OUTSW .


(   1.7. INTEGER CONVERSION  )


*ACTION* CONVERT INTEGER FROM CHARACTER TO VALUE ,
         CONVERT INTEGER TO CHARACTER FORMAT ,
         INCREMENT LINE NUMBER .


 CONVERT INTEGER FROM CHARACTER TO VALUE + CF + VALUE-X-Y-Z-K-K1:
   MAKE+WORD+CF,MAKE+K+9,MAKE+VALUE+0,
   ABC: UNPACK+WORD+X+Y,
    WAS DIGIT+Y,
     (SUBTR+Y+54+Z,
       (SUBTR+9+K+K1,
         QQQ: EQUAL+K1+0,ADDMULT+Z+1+VALUE+VALUE :
                         ADDMULT+Z+10+0+Z , :QQQ ) ,
      DECR+K,EQUAL+K+0 :
      MAKE+WORD+X, :ABC ):    .

CONVERT INTEGER TO CHARACTER FORMAT+X+Y-K-DIV-REM:
 MAKE+WORD+X,MAKE+K+9,
 ABC: DIVREM+WORD+10+DIV+REM,ADD+REM+54+REM,
  (EQUAL+K+1, MSG + ERR20 : ) ,
    PUT+CHARACTERS1+K+REM,
     (LSEQ+DIV+9,ADD+DIV+54+DIV,DECR+K,PUT+CHARACTERS1+K+DIV:
      DECR+K,MAKE+WORD+DIV, :ABC) ,
 MAKE+WORD+0,
 NXT: GET+CHARACTERS1+K+CHAR1,ADDMULT+WORD+CHARSIZE+CHAR1+WORD,
     EQUAL+K+9 , MAKE+Y+WORD :
     INCR+K, :NXT .

INCREMENT LINE NUMBER -X-Y-Z :
  CONVERT INTEGER FROM CHARACTER TO VALUE+CLINE+X,
   CONVERT INTEGER FROM CHARACTER TO VALUE+INCR+Y,
     ADD+X+Y+Z,
       CONVERT FROM VALUE TO CHARACTER+Z+CLINE .
```

```
(   2. INPUT,OUTPUT AND KEYWORD PROCESSING  )

(   2.1. INPUT PROCESSING  )

(   2.1.1. INTERRUPT RECOGNITION   )

*ACTION* NEXTCHAR.

NEXTCHAR :
 RESYM * CHAR ,
  HOLDS * STOCK,(EQUAL * CHAR * A CHAR , PRSYM * CHAR ,
                      INTERRUPT PROCESSING :
                 RESET * STOCK , PRSYM * CHAR ) ;
  EQUAL * CHAR * PERCENT CHAR , SET * STOCK , PRSYM * CHAR :
  PRSYM * CHAR .

(   2.1.2. READ KEYWORDS   )

READ KEYWORD - N - X :
  MAKE*N*0,
   (ABC: WAS LETTER*CHAR,INCR*N,
         (LSEQ*N*8 :
         MSG * ER8 , EXIT ) ,
         ADDMULT*WORD*CHARSIZE*CHAR,NEXTCHAR*CHAR, !ABC),
      EQUAL*N*0,FALSE :
      WAS A KEYWORD DELIMITER*CHAR, SUBTR*8*N*X,
       (QQQ: EQUAL*X*0,ADD*N*54*N,ADDMULT*WORD*CHARSIZE*N,
                                        MAK *KEYWORD*WORD :
             MAKE*CHAR1*SPACE CHAR,ADDMULT*WORD*CHARSIZE*CHAR1,
                   DECR*X, !QQQ) , TRUE :
      FALSE.

WAS A KEYWORD DELIMITER * X :
     EQUAL * X * COMMA , EQUAL * X * SPACE CHAR .

IS A VALID LINE NUMBER *X - Y :
   READ INTEGER * X ,
      CONVERT INTEGER FROM CHARACTER TO VALUE * X * Y ,
         ( LSEQ * Y * 999999 : MSG * ERR12 ) .

IS A VALID COLUMN NUMBER * X - Y :
   READ INTEGER * X ,
      CONVERT INTEGER FROM CHARACTER TO VALUE * X * Y ,
        ( LSEQ * Y * CH , LESS * 0 * Y : MSG * ERR9  ) .

IS A VALID INCREMENT * X :
   (LSEQ*X*8,MSG*ERR11: ) ,
       IS A VALID LINE NUMBER * X .

IS IT END OF FILE :
  EQUAL * CHAR * EOF CHAR .
```

```
(   2.2.  OUTPUT PROCESSING     )


(   2.2.1. BASIC ACTIONS TO PRINT   )


ACTION+  OUTINT , OUTINT1 , PRINT1 , POSITION , SPACES ,
         SPACE , PRCHAR , OUTPUT LINE NUMBER .


SPACE :
  PRCHAR + SPACE CHAR .

SPACES + X - N :
  MAKE + N + 0 ,
   NXT: LESS + N + X , SPACE , INCR + N , NXT .

POSITION + X - Q :
  SUBTR + X + POS + Q , LSEQ + 0 + Q , SPACES + Q :
  PRSYM + EOL CHAR , MAKE + POS + 0 , SPACES + X .

PRCHAR + X :
  EQUAL + X + EOL CHAR , PRSYM + EOL CHAR , MAKE + POS + 0 ,:
  EQUAL + X + TAB CHAR , SPACES + PTAB1 :
  EQUAL + X + NIX :
  PRSYM + CHAR , INCR + POS .

PRINT1 + X - Q - R :
   EQUAL + NCHARS + CHARPERWD:
   DIVREM+X+CHARSIZE+Q+R,INCR+NCHARS,PRINT1+Q,PRCHAR+R.

OUTINT + X - Q - R :
  SPACE ,
   LESS+X+0,MAKE+R+X,MARK+R,PRCHAR+MINUS CHAR,OUTINT+R:
   EQUAL+X+0,PRCHAR+NULL CHAR,SPACE:
   DIVREM+X+10+Q+R,OUTINT1+Q,ADD+R+54+P,PRCHAR+R,SPACE.

OUTINT1 + X - R - Q :
   EQUAL+X+0:
   DIVREM+X+10+Q+R,OUTINT1+Q,ADD+R+54+R,PRCHAR+R .

OUTPUT LINE NUMBER + X - Y :
  CONVERT INTEGER FROM CHARACTER TO VALUE + X + Y , OUTINT + Y .


(   2.3. KEYWORD IDENTIFICATION   )


IS KEYWORD + NAME -Q - P - N :
   ( LESS+KEYWORD+0,READ KEYWORD :  ) ,
   DECODE KEYWORD INTO CHARACTERS+KEYWORD+CHARACTERS2+P,
   DECODE KEYWORD INTO CHARACTERS+NAME+CHARACTERS1+Q ,
   MAKE+N+1,
   (LESS+Q+54,LESS+63+Q,MAKE+Q+1;
                SUBTR+Q+54+Q ) ,
  ABC: GET+CHARACTERS1+N+CHAR1,GET+CHARACTERS2+N+CHAR2,
     EQUAL+CHAR1+CHAR2
       LESS+N+P, !ABC :
       EQUAL+N+P,LSEQ+Q+N,MARKED+KEYWORD .
```

```
(   3.  READING OPTIONAL PARAMETERS OF EDITOR COMMANDS   )


(   3.1. COMMON PARAMETERS      )

READ LINE NUMBER + X :
  EQUAL + SW1 + 0 ,
     IS A VALID LINE NUMBER + X ,
         MAKE + SW1 + 1 .

READ INCREMENT VALUE + X :
  EQUAL + SW2 + 0 ,
     IS A VALID INCREMENT + X ,
         MAKE + SW2 + 0 .

READ SUPPRESS :
   EQUAL + SW3 + 0 ,
      IS KEYWORD + SUP , MAKE + PSUP + 1 ,
         MAKE + SW3 + 1 .

READ OVERWRITE :
   EQUAL + SW4 + 0 ,
      IS KEYWORD + OVERWRITE , MAKE + POVERWRITE + 1 .

READ VETO :
   EQUAL + SW5 + 0 ,
   IS KEYWORD + VETO , MAKE + PVETO + 1 , MAKE + SW5 + 1 .

READ UNIT :
   EQUAL + SW6 + 0 ,
      IS KEYWORD + UNIT , MAKE + PUNIT + 1 , MAKE + SW6 + 1 .

READ NOSEQ :
   EQUAL + SW7 + 0 ,
      IS KEYWORD + NOSEQ , MAKE + PNOSEQ + 1 , MAKE + SW7 + 1 .

READ MERGE :
   EQUAL + SW8 + 0 ,
      IS KEYWORD + MERGE , MAKE + PMERGE + 1 , MAKE + SW8 + 1 .

READ SEARCH TEXT :
   EQUAL + SW9 + 0 ,
      EQUAL + CHAR + TEXT DELIMITER , GET INPUT TEXT ,
         MAKE + SW9 + 1 .


(   3.2.  READING PARAMETERS OF FORMAT COMMAND   )

IS FORMAT NAME SPECIFIED :
   IS KEYWORD + ALGOL ,
     MAKE + TABCHAR + DOLLAR , MAKE + PTAB1 + 7 , MAKE + PTAB2 + 10 ,
       MAKE + PTAB3 + 13 , MAKE + PTAB4 + 16 , MAKE + PTAB5 + 19 ;
   IS KEYWORD + BASIC ,
       MAKE + INCREMENT SWITCH + 1 , MAKE + PTAB1 + 7 ;
   IS KEYWORD + COBOL ,
       MAKE + PTAB1 + 8 , MAKE + PTAB2 + 12 , MAKE + PTAB3 + 16 ,
         MAKE + PTAB4 + 20 , MAKE + PTAB5 + 24 ;
   IS KEYWORD + COMPASS ,
       MAKE + PTAB1 + 11 , MAKE + PTAB2 + 18 , MAKE + PTAB3 + 36 ,
    IS KEYWORD + FORTRAN .
```

```
IS SHOW PARAMETER :
    IS KEYWORD + SHOW , MAKE + PSHOW +1 .


GET TAB CHAR :
  EQUAL + SW1 + 0 ,
    IS KEYWORD + TAB , RESYM + CHAR , MAKE + TABCHAR + CHAR ,
       MAKE + SW1 + 1 .


GET LINE SIZE :
 EQUAL + SW2 + 0 ,
    IS KEYWORD + CH , READ INTEGER + PCH ,
   (LSEQ+PCH+510 : MSG+ERR3) , MAKE + SW2 + 1 .


GET TAB COLUMNS :
 EQUAL + SW3 + 0 ,
   READ INTEGER + PTAB1 , MAKE + SW3 + 1 , READ INTEGER + PTAB2 ,
   READ INTEGER + PTAB3 , READ INTEGER + PTAB4 , READ INTEGER + PTAB5 .


*ACTION* PARAMETER INITIALISATION FOR FORMAT COMMAND ,
         READ OPTIONAL PARAMETERS OF FORMAT COMMAND .


PARAMETER INITIALISATION FOR FORMAT COMMAND :
  MAKE + INCREMENT SWITCH + 0 , MAKE + PCH + 72 ,
    MAKE + TABCHAR + SEMICOLON , MAKE + PTAB1 + 7 , MAKE + SW1 + 0,
     MAKE + SW2 + 0 , MAKE + SW3 + 0 .


READ OPTIONAL PARAMETERS OF FORMAT COMMAND :
  PARAMETER INITIALISATION FOR FORMAT COMMAND ,
   IS FORMAT NAME SPECIFIED , READ END OF COMMAND :
   IS SHOW PARAMETER , SHOW IT , READ END OF COMMAND :
   (ABC: GET TAB CHAR , !ABC :
         GET LINE SIZE , !ABC :
         GET TAB COLUMNS , !ABC :
         READ END OF COMMAND ) :
  MSG+ERR21 .

READ END OF COMMAND :
   EQUAL + CHAR + EOL CHAR .


(   3.3.  READING PARAMETERS OF CREATE AND EDIT COMMANDS )


*ACTION* INITIALISATION OF CREATE COMMAND PARAMETERS ,
         READ OPTIONAL PARAMETERS OF CREATE COMMAND ,
         READ NAME OF SOURCE FILE , GET FILE NAME ,
         READ OPTIONAL PARAMETERS OF EDIT COMMAND ,
         GET FILE NAME .


INITIALISATION OF CREATE COMMAND PARAMETERS :
  MAKE + SW1 + 0 , MAKE + SW2 + 0 , MAKE + SW3 + 0 ,
   MAKE + FLINE + DFLINE , MAKE + INCR + DINCR , MAKE + PSUP + 0 .
```

```
READ OPTIONAL PARAMETERS OF CREATE COMMAND :
 INITIALISATION OF CREATE COMMAND PARAMETERS .
  ABC: READ LINE NUMBER , !ABC ;
       READ INCREMENT   , !ABC ;
       READ SUPPRESS    , !ABC ;
       READ END OF COMMAND :
      MSG+ERR21 .

GET FILE NAME - F :
  READ KEYWORD , CHECK FILE NAME + F ,
     EQUAL + F + 1 , MSG + ERR22 ;
     EQUAL + C + 2 , MSG + ERR7 .


READ NAME OF SOURCE FILE - Q :
  GET FILE NAME + Q , CONNECT FILE + Q .


READ OPTIONAL PARAMETERS OF EDIT COMMAND :
 (IS KEYWORD+SEQUENCE,MAKE+PSEQUENCE+1;MAKE+PSEQUENCE+0),
 (READ END OF COMMAND : MSG + ERR21 ) .


(   3.4. READING PARAMETERS FOR THE GROUP:LIST,DELETE,SAVE,TEXT )


*ACTION* PARAMETER INITIALISATION ,
         CHECK PARAMETER CONSISTENCY ,
         READ SELECTIVE PARAMETERS OF COMMAND.


READ LINE LIMITS+SL+EL-Q:
  EQUAL+SW1+0,
   (IS KEYWORD+ALL,MAKE+SL+FLN,MAKE+EL+LLN:
    IS KEYWORD+LAST,MAKE+SL+LLN,MAKE+EL+LLN:
    IS A VALID LINE NUMBER+Q,MAKE+SL+Q,
      (IS KEYWORD+LAST,MAKE+EL+LLN:
       IS A VALID LINE NUMBER+Q,MAKE+EL+Q:
       QQQ: MAKE+ERRSW+1,MSG+ERR19)),
 (LESS+EL+SL,!QQQ : MAKE+SW1+1) .


READ COLUMN LIMITS + SC + EC - Q :
  EQUAL+SW2+0,EQUAL+CHAR+LEFT PARANTHESIS,
   (IS A VALID COLUMN NUMBER+Q,MAKE+SC+Q ;
    MAKE+ERRSW+1,MSG+ERR9),
  EQUAL+CHAR+COMMA ,
   (IS A VALID COLUMN NUMBER+Q,MAKE+EC+Q,
    QQQ: EQUAL+CHAR+RIGHT PARANTHESIS, MAKE+SW2+1 :
         MSG+ERR17)) .
```

```
PARAMETER INITIALISATION+SL+EL+SC+EC:
 MAKE+SW1+0,MAKE+SW2+0,MAKE+SW3+0,MAKE+SW4+0,MAKE+SW5+0,MAKE+SW6+0,
   MAKE+SW7+0,MAKE+SW8+0,MAKE+SW9+0,MAKE+SL+CLINE,MAKE+EL+CLINE,
    MAKE+SC+1,MAKE+EC+LASTCOL,MAKE+ERRSW+0 .


SEARCH CONSISTENCY :
  EQUAL+SW9+0,(EQUAL+SW2+0,MAKE+ERRSW+0:
  MSG+ERR+,MAKE+ERRSW+1).


CHECK PARAMETER CONSISTENCY+SL+EL+SC+EC+W:
 LESS+EL+SL,MSG+ERR14, ABC: MAKE+ERRSW+1 :
 LESS+LLN+EL,MSG+ERR10 , !ABC :
 LSEQ+LC+SC,  STR! MSG+ERR9 , !ABC :
 EQUAL+W+L CHAR ,
   (EQUAL+SW5+0, AKK: EQUAL+SW7+0,EQUAL+SW4+0,EQUAL+SW8+0,
      SEARCH CONSISTENCY :
      QQQ: MSG+ERR21, !ABC) :
 EQUAL+W+D CHAR ,
   (KOK: EQUAL+SW3+0, !AKK :
         !QQQ) :
 EQUAL+W+T CHAR ,
   (EQUAL+SW3+0,EQUAL+SW9+0, !AKK :
         !QQQ) :
 EQUAL+W+S CHAR ,
   (EQUAL+SW3+0,MAKE+ERRSW+0:
         !QQQ) :
 MSG+EL1 .


READ SELECTIVE PARAMETERS OF COMMAND+SL+EL+SC+EC+W:
  PARAMETER INITIALISATION+SL+EL+SC+EC+W,
  (ABC: READ LINE LIMITS+SL+EL, !ABC :
        READ COLUMN LIMITS+SC+EC,!ABC:
        READ SUPPRESS,           !ABS:
        READ OVERWRITE,          !ABC:
        READ VETO,               !ABC:
        READ UNIT,               !ABC:
        READ NOSEQ,              !ABC:
        READ MERGE,              !ABC:
        READ SEARCH TEXT,        !ABC:
        READ END OF COMMAND        :
        MSG+ERR21),
 CHECK PARAMETER CONSISTENCY+SL+EL+SC+EC+W .
```

```
(   3.5.  READ ADD AND RESEQUENCE COMMAND PARAMETERS    )


*ACTION*  PARAMETER INITIALISATION FOR ADD COMMAND ,
          READ OPTIONAL PARAMETERS OF ADD COMMAND ,
           PARAMETER INITIALISATION FOR RESEQUENCE COMMAND ,
           READ OPTIONAL PARAMETERS OF RESEQUENCE COMMAND .


PARAMETER INITIALISATION FOR ADD COMMAND + X + Y :
   MAKE + X + LLN , ADD + X + INCR + X , MAKE + SUP + 0 ,
    MAKE + OVERWRITE + 0 , MAKE + Y + INCR , MAKE + SW1 + 0 ,
      MAKE + SW2 + 0 ,MAKE + SW3 + 0 , MAKE + SW4 + 0 .


READ OPTIONAL PARAMETERS OF ADD COMMAND + X + Y :
   PARAMETER INITIALISATION FOR ADD COMMAND + X + Y ,
    ABC:  READ LINE NUMBER + X ,  !ABC :
          READ INCREMENT VALUE + Y , !ABC :
          READ SUPPRESS , !ABC ;
          READ OVERWRITE , !ABC :
          READ END OF COMMAND   :
          MSG+ERR21.


PARAMETER INITIALISATION FOR RESEQUENCE COMMAND+X+Y:
   MAKE+X+FLINE,MAKE+Y+DINCR.


READ OPTIONAL PARAMETERS OF RESEQUENCE COMMAND+X+Y:
   PARAMETER INITIALISATION FOR RESEQUENCE COMMAND+X+Y,
    ABC:  READ LINE NUMBER+X,  !ABC :
          READ INCREMENT VALUE+Y, !ABC :
          READ END OF COMMAND :
          MSG+ERR21.


(   3.6.  READ RUN COMMAND PARAMETERS    )


READ NAME OF PROCESSOR :
   IS KEYWORD   + ALGOL ,MAKE+PROCESSOR+KEYWORD :
   IS KEYWORD   + BASIC ,MAKE+PROCESSOR+KEYWORD :
   IS KEYWORD   + COBOL ,MAKE+PROCESSOR+KEYWORD :
   IS KEYWORD   + COMPASS ,MAKE+PROCESSOR+KEYWORD :
   IS KEYWORD   + FTN ,MAKE+PROCESSOR+KEYWORD :
   IS KEYWORD +KEYWORDRUN ,MAKE+PROCESSOR+KEYWORD

*ACTION* READ OPTIONAL PARAMETERS OF RUN COMMAND.

READ OPTIONAL PARAMETERS OF RUN COMMAND :
  ABC:  IS KEYWORD+FILE,MAKE+PFILE+1, GET FILE NAME , !ABC :
        IS KEYWORD+NOEX,MAKE+PNOEX+1,    !ABC :
        READ END OF COMMAND :
        MSG+ERR21.
```

(3.7. IDENTIFICATION OF INTERCOM COMMANDS)

```
ANY OF THE INTERCOM COMMANDS :
  IS KEYWORD + ALTER ;
  IS KEYWORD + ATTACH ;
  IS KEYWORD + ASSETS ;
  IS KEYWORD + BATCH ;
  IS KEYWORD + BKSP ;
  IS KEYWORD + COMBINE ;
  IS KEYWORD + COMPARE ;
  IS KEYWORD + CONVERT ;
  IS KEYWORD + COPYN ;
  IS KEYWORD + CATALOG ;
  IS KEYWORD + CONNECT ;
  IS KEYWORD + COPY ;
  IS KEYWORD + COPYCF ;
  IS KEYWORD + COPYBF ;
  IS KEYWORD + COPYCR ;
  IS KEYWORD + COPYBR ;
  IS KEYWORD + COPYBCD ;
  IS KEYWORD + COPYSBF ;
  IS KEYWORD + DISCONT ;
  IS KEYWORD + EFL ;
  IS KEYWORD + ETL ;
  IS KEYWORD + FETCH ;
  IS KEYWORD + FILES ;
  IS KEYWORD + LOCK ;
  IS KEYWORD + MAP ;
  IS KEYWORD + MESSAGE ;
  IS KEYWORD + PAGE ;
  IS KEYWORD + Q ;
  IS KEYWORD + RETURN ;
  IS KEYWORD + RENAME ;
  IS KEYWORD + REWIND ;
  IS KEYWORD + SKIPB ;
  IS KEYWORD + SAVEFL ;
  IS KEYWORD + SWITCH ;
  IS KEYWORD + SCREEN ;
  IS KEYWORD + REQUEST ;
  IS KEYWORD + PURGE ;
  IS KEYWORD + STORE ;
  IS KEYWORD + SKIPF ;
  IS KEYWORD + SITUATE ;
  IS KEYWORD + SEND ;
  IS KEYWORD + XEQ .
```

```
{   3.8.  MISCELLANEOUS ACTIONS AND PREDICATES FOR INITIALISATION }


*ACTION*  SAVE CURRENT LINE NUMBER ,
          RESTORE CURRENT LINE NUMBER ,
          TURN ON SAVED FLAG ,
          TURN OFF SAVED FLAG ,
          COMPUTE BLOCK ELEMENTS ,
          BLOCK INITIALISATION ,
          CREATE MODE INITIALISATION ,
          COMMAND MODE INITIALISATION ,
          READY FOR COMMAND .


SAVE CURRENT LINE NUMBER * X :
  MAKE * SAVED LINE * CLINE , MAKE * CLINE * X .


RESTORE CURRENT LINE NUMBER :
  MAKE * CLINE * SAVED LINE .


TURN ON SAVED FLAG :
  MAKE * SAVED FLAG * 1 .

TURN OFF SAVED FLAG :
  MAKE * SAVED FLAG * 0 .


COMPUTE BLOCK ELEMENTS * Y :
  ADD * OVE * MEN * BOS , ADDMULT * BOS * ES * I * Y ,
    SUBTR * BLKSIZE * Y * TEXTSIZE .
  MAKE * SAVED FLAG * 0 .


BLOCK INITIALISATION :
  COMPUTE BLOCK ELEMENTS , INCR * BOC , PUT BLOCK ID * BOC ,
    MAKE * MEN * MEN1 .


CREATE MODE INITIALISATION :
  MAKE * CREATE FLAG * 1 , MAKE * BOC * 0 , BLOCK INITIALISATION ,
    MAKE * HBN * MHBN , MAKE * FLINE * DFLINE , MAKE * INCR * DINCR ,
      MAKE * LAST COLUMN * CH .


COMMAND MODE INITIALISATION :
    MAKE * CREATE FLAG * 0 , MAKE * MEN * BOC , MAKE * BOR * BOS ,
    MAKE*PVETO*0,MAKE*PSHOW*0,MAKE*POVERWRITE*0,
    MAKE*PUNIT*0,MAKE*PNOSEQ*0,MAKE*PMERGE*0.
```

```
IS TEXT REPLACEMENT :
  EQUAL + CHAR + TEXT DELIMITER ,
    GET INPUT STRING ,
    [ EQUAL + CHAR + EQUAL SIGN , NEXTCHAR + CHAR ,
       EQUAL + CHAR + TEXT DELIMITER , GET OUTPUT STRING ;
      MSG+ERR16.


WAS EDIT FILE SAVED :
  EQUAL + SAVED FLAG + 1 .


IS END OF CREATE MODE :
  EQUAL + CHAR + EQUAL SIGN ,
   ABC: SKIP BLANKS , EQUAL + CHAR + EOL CHAR :
   EQUAL + SAVED CHAR + EQUAL SIGN . :ABC .


IS A REQUEST TO ADD A LINE WITH A GIVEN LINE NUMBER  + X :
  EQUAL + CHAR + EQUAL SIGN , MAKE + SAVED CHAR + CHAR ,
    READ LINE NUMBER + X , EQUAL + CHAR + EQUAL SIGN .


READY FOR COMMAND :
    ACKNOWLEDGE EDITOR MODE , WAIT REPLY .
```

```
(  4.  MASTER FILE DIRECTORY MANAGEMENT  )


*ACTION* COMPUTE MFD ENTRY ADDRESS ,
         GET MFD ENTRY ,
         PUT MFD ENTRY ,
         GET MFD ENTRY FOR BCC ,
         PUT MFD ENTRY FOR BCC ,
         FIND BLOCK ,
         DELETE ENTRY FROM MFD ,
         INSERT ENTRY IN MFD .

COMPUTE MFD ENTRY ADDRESS * X * Y :
   SUBTR * X * 1 * Y ,  ADDMULT * Y * 3 * 1 * Y .

GET MFD ENTRY * X * Y * Z * W - P :
  LSEQ * X * HBN , COMPUT. MFD ENTRY ADDRESS * X * P ,
    GET FIRST LINE * P * Y , GET LAST LINE * P * Z ,
       GET FREE SPACE * P * W : MSG * EE2 .

PUT MFD ENTRY * X * Y * Z * W - P :
 LSEQ * X * MHBN , COMPUTE MFD ENTRY ADDRESS * X * P ,
   PUT FIRST LINE * P * Y , PUT LAST LINE * P * Z ,
      PUT FREE SPACE * P * W : MSG * EE3 .

GET MFD ENTRY FOR BCC :
   GET MFD ENTRY * BCC * FL3 * LLB * BFS .

PUT MFD ENTRY FOR BCC :
   PUT MFD ENTRY * BCC * FL3 * LLB * BCC .

FIND BLOCK * BLOCKNR * LN - X - Y - Z :
   LSEQ * FLN * LN , LSEQ * LN * LLN ,
      MAKE * BLOCKNR * BCC ,
 ABC: GET MFD ENTRY FOR BLOCK * BLOCKNR * X * Y * Z ,
      ( LESS * LN * X , DECR * BLOCK NR , !ABC :
        LESS * Y * LN , INCR * BLOCK NR , !ABC :
        MAKE * BCC * BLOCK NR ) : MSG * ERR10 .
   MSG*ERR10 .

DELETE ENTRY FROM MFD * ENTRYNR - Q - A - B - C :
  EQUAL * ENTRYNR * HBN , DECR * HBN :
  ABC: ADD * ENTRYNR * 1 * Q ,READ EDIT FIL. BLOCK * Q ,
   MAKE * BCC * Q , PUT BLOCK ID * ENTRY NR ,
    WRITE BLOCK * ENTRY NR ,
     GET MFD ENTRY * Q * A * B * C ,
       PUT MFD ENTRY * ENTRY NR * A * B * C ,
        INCR * ENTRY NR , EQUAL * ENTRY NR * HBN , DECR * HBN :
                          !ABC .

INSERT ENTRY IN MFD * ENTRY NR*X*Y*Z-T-BQ-BN-A-B-C :
  ADD * HBN * 1 * T ,
(LSEQ*T*MHBN,EQUAL*ENTRYNR*T,PUT MFD ENTRY*T*X*Y*Z,MAKE*HBN*T:
 LESS*T*ENTRYNR,MSG*EE2:
 MAKE*BQ*HBN,
  ABC: ADD*BQ*1*BN,READ EDIT FILE BLOCK*BQ,
     MAKE*BCC*BQ,PUT BLOCK ID*BN,WRITE BLOCK*BN,
     GET MFD ENTRY*BQ*A*B*C,PUT MFD ENTRY*BN*A*B*C,
     EQUAL*BQ*ENTRYNR,PUT MFD ENTRY*BQ*X*Y*Z,
     MAKE*HBN*T:
     DECR*BQ, !ABC) .
```

```
{  5. BLOCK MANAGEMENT  }

{  5.1. BLOCK DIRECTORY MANAGEMENT  }

{  5.1.1. BASIC ACTIONS FOR BLOCK DIRECTORY MANAGEMENT }

*ACTION* GET LINE ENTRY IN BLOCK DIRECTORY ,
         PUT LINE ENTRY IN BLOCK DIRECTORY ,
         PUT BLOCK ID ,
         ADD A LINE IN A PARTIALLY FILLED BLOCK DIRECTORY ,
         DELETE LOGICALLY LINE FROM BCC ,
         WRITE LINE ENTRY IN NEXT AVAILABLE LOCATION OF BCC DIRECTORY.

GET LINE ENTRY IN BLOCK DIRECTORY+PTR+LINENR+LINEPOINTER-X:
  LSEQ+PTR+MEN,SUBTR+PTR+1+X,ADDMULT+X+2+2+X,
     GET LINE NUMBER+X+LINENR , GET LINE POINTER+X+LINEPOINTER :
 MSG+EE4.

PUT LINE ENTRY IN BLOCK DIRECTORY+PTR+LINENR+LINEPOINTER-X:
  (LSEQ+LINENR+LLN : MAKE+LLN+LINENUMBER) ,
  LSEQ+PTR+MEN,SUBTR+PTR+1+X,ADDMULT+X+2+2+X,
     PUT LINE NUMBER+X+LINENR , PUT LINE POINTER+X+LINEPOINTER :
  MSG+EE5 .

PUT BLOCK ID + X
  PUT+BLOCK+1+X.

ADD A LINE IN A PARTIALLY FILLED BLOCK DIRECTORY+P+X+Y-A-B-S-T:
  MAKE+A+BDS,
    ABC : DECR+A,SUBTR+A+1+B,
          GET LINE ENTRY IN BLOCK DIRECTORY+B+S+T,
          PUT LINE ENTRY IN BLOCK DIRECTORY+A+S+T,
             EQUAL+A+P,PUT LINE ENTRY IN BLOCKDIRECTORY+A+X+Y ,
             !ABC .

DELETE LOGICALLY LINE FROM BCC+P-X-Y-Z :
  GET LINE ENTRY IN BLOCK DIRECTORY+P+X+Y,
    MAKE+Z+0,PUT LINE ENTRY IN BLOCK DIRECTORY+P+Z+Y,
      DIRECTORY COMPACTION.

WRITE LINE ENTRY IN NEXT AVAILABLE LOCATION OF BCC DIRECTORY:
   LSEQ+OBPTR+MEN,
     (LSEQ+LINESIZE+BFS,
        PUT LINE ENTRY IN BLOCK DIRECTORY+OBPTR+OLINE+LINEPTR,
        INCR+OBPTR ;
      REQUEST FOR A NEW BLOCK):
   REQUEST FOR A NEW BLOCK .
```

```
(  5.1.2. BLOCK DIRECTORY BINARY SEARCH ACTIONS )

*ACTION* BLOCK DIRECTORY BINARY SEARCH,
         FIND PLACE OF INSERTION IN DIRECTORY.

BLOCK DIRECTORY BINARY SEARCH*LN*Q*L1*L2*N1*N2=X-Y-R :
    MAKE*N1+1,MAKE*N2+DOS,
    ADD: SUBTR*N2*N1*M,
      EQUAL*M*1,
       (GET LINE ENTRY IN BLOCK DIRECTORY*N1*L1*Y,
           EQUAL*L1*LN,MAKE*Q*N1:
        GET LINE ENTRY IN BLOCK DIRECTORY*N2*L2*Y,
           EQUAL*L2*LN,MAKE*Q*N2:
                       MAKE*Q*0 ) :
     DIVR=M*M+2*P*R,ADD*N1*P*P,
        GET LINE ENTRY IN BLOCK DIRECTORY*P*X*Y,
           EQUAL*X*LN,MAKE*Q*P :
           LESS*X*LN,MAKE*N1*P,  :ADD :
                     MAKE*N2*P,  :ABC .

IS ENTRY THERE * LN * PTR - Q -L1 - L2 - N1 - N2 :
  BLOCK DIRECTORY BINARY SEARCH*LN*Q*L1*L2*N1*N2 ,
    LS=Q*1*Q , MAKE*PTR*Q .

FIND PLACE OF INSERTION IN BLOCK DIRECTORY*LN*PTR-Q*L1-L2-N1-N2:
  EQUAL*BOO*HSN,LESS*LLN*LN,ADD*DOPTR*1*PTR :
  BLOCK DIRECTORY BINARY SEARCH*LN*Q*L1*L2*N1*N2 ,
    EQUAL*Q*0,
      (LESS*L1*LN,LESS*LN*L2,ADD*N1*1*PTR,
  ADD A NEW LINE ENTRY IN A PARTIALLY FILLED BLOCK DIRECTORY*PTR*1*L :
      MSG*EE6):
  MAKE*PTR*Q.


(   5.2. ACTIONS FOR THE MANAGEMENT OF THE TEXT SECTION OF BLOCK )

*ACTION* ADD TO , STACK , STACK LAST ,
         READ TEXT OF LINE FROM THE TERMINAL AND STORE IT IN BLOCK.

ADD TO * XCHAR * LISTNAME * MAXLIST * CURRENT PTR :
  EQUAL*CHAR COUNT*CHARPERWD,
     STACK*WORD*LISTNAME*MAXLIST*CURRENT PTR,
     MAKE*WORD*XCHAR,MAKE*CHAR COUNT*1 :
  ADDMULT*WORD*CHAR SIZE*X CHAR*WORD,INCR*CHAR COUNT.

STACK *X * LISTNAME * MAXLIST * CURRENT PTR :
  LSEQ * CURRENT PTR * MAXLIST , PUT * LISTNAME * CURRENT PTR * X ,
                                                INCR * CURRENT PTR :
MSG*EE7.

STACK LAST * LISTNAME * MAXLIST * CURRENT PTR :
  ABC: (EQUAL * CHAR COUNT * CHARPERWD ,
        INVERT * WORD , STACK * WORD * LISTNAME * MAXLIST * CURRENT PTR,
        MAKE * WORD * 0 , MAKE * CHAR COUNT * 0 :
  ADDMULT * WORD * CHAR SIZE * FILL CHAR * WORD ,
          INCR * CHAR COUNT , :ABC) .
```

```
READ TEXT OF LINE FROM THE TERMINAL AND STORE IT IN BLOCK :
   MAKE + CHAR COUNT + 1 ,
   ABC: EQUAL + CHAR + EOL CHAR ,
        STACK LAST + BLOCK + BLKSIZE + LINEPTR ,
        DECR + BFS , INCR + LINEPTR :
      ADD TO + CHAR + BLOCK + BLKSIZE + LINEPTR ,
        NEXTCHAR + CHAR , DECR + BFS , INCR + LINE PTR , *ABC .


(  5.3. BLOCK TRAFFIC MANAGEMENT  )


*ACTION* COMPUTE OLD BLOCK DESCRIPTOR ENTRIES ,
         FILL IN OLD BLOCK DESCRIPTOR ,
         REQUEST FOR A NEW BLOCK ,
         CLOSE EDIT FILE ,
         READ EDIT FILE BLOCK .


COMPUTE OLD BLOCK DESCRIPTOR ENTRIES :
    MAKE + LLB + CLINE , SUBTR + BLKSIZE + LINEPTR + BFS .

FILL IN OLD BLOCK DESCRIPTOR :
     EQUAL + BCC + 1 ;
     COMPUTE OLD BLOCK DESCRIPTOR ENTRIES ,
        PUT MFD ENTRY FOR BCC .

REQUEST FOR A NEW BLOCK :
   FILL IN OLD BLOCK DESCRIPTOR ,
     WRITE BLOCK + BCC , INCR + BCC ,
        LESS + MHBN + BCC , MSG + EE3 ;
        MAKE + FLB + CLINE , BLOCK INITIALISATION .

CLOSE EDIT FILE :
   FILL IN OLD BLOCK DESCRIPTOR ,
     MAKE + FLN + FLINE , MAKE + LLA + LLB , WRITE BLOCK + BCC ,
    MAKE+EDIT FILE EMPTY + 1 .

READ EDIT FILE BLOCK + BLOCKID -X - Y - Z :
    GET MFD ENTRY + BLOCKID + X + Y + Z ,
       MAKE + FLB + X , MAKE + LLB + Y , MAKE + BFS + Z ,
          MAKE + BCC + BLOCKID , READ BLOCK + BLOCKID .


(  5.4. ACTIONS RELATED TO BLOCK COMPACTION )


*ACTION*  BLANK REMAINING ENTRIES IN BLOCK DIRECTORY ,
          BLOCK DIRECTORY COMPACTION ,
          COPY UP LINE OF TEXT ,
          RESTORE LINE OF TEXT ,
          CHANGE LINE POINTER IN BLOCK DIRECTORY ENTRY ,
          FILL BACK TEXT ,
          TEXT COMPACTION ,
          BLOCK COMPACTION ,
          RESTRUCTURE BLOCK .
```

```
BLANK REMAINING ENTRIES IN BLOCK DIRECTORY + X :
   ABC: LSEQ + X + BDS ,
     PUT + BLOCK + X + MINUS ONE , INCR + X , #ABC :  .

BLOCK DIRECTORY COMPACTION + Q - SP - X - Y :
   MAKE + SP + 1 , MAKE + DBPTR + 1 ,
   ABC: GET LINE ENTRY IN BLOCK DIRECTORY + DBPTR + X + Y ,
     EQUAL + X + LLB ,
       ( EQUAL + SP + DBPTR , MAKE + 1 + 0 :
                              MAKE + 0 + 1 ) ,
   ADD + DBPTR + 1 + SP , BLANK REMAINING ENTRIES IN BLOCK DIRECTORY :
    ( EQUAL + X + 1 , INCR + DBPTR , #ABC :
      EQUAL + SP + DBPTR , INCR + DBPTR , INCR + SP , #ABC :
      PUT LINE ENTRY IN BLOCK DIRECTORY + SP + Y + Y , INCR + SP ,
         INCR + DBPTR , #ABC ) .

COPY UP LINE OF TEXT + X + Y + N :
   MAKE + N + 1 ,
   ABC: GET + BLOCK + Y + WORD , PUT + NEWTEXT + X + WORD ,
     MARKED + WORD  :
     INCR + N , INCR + X , INCR + Y , #ABC .

RESTORE LINE OF TEXT + X :
   ABC: GET + NEWTEXT + X + WORD , PUT + BLOCK + LINEPTR + WORD ,
      MARKED + WORD ;
      INCR + X , INCR + LINEPTR , DECR + BFS , #ABC .

CHANGE LINE POINTER IN BLOCK DIRECTORY ENTRY + A + B - X - Y :
   GET LINE ENTRY IN BLOCK DIRECTORY + A + X + Y ,
      PUT LINE ENTRY IN BLOCK DIRECTORY + A + X + B .

FILL BACK TEXT + L - Q - X :
   ADD + BDS + 1 + LINEPTR , MAKE + Q + 1 , MAKE + DBPTR + 1 ,
     PUT POINTER IN LINE ENTRY + DBPTR + LINEPTR , MAKE +BFS+LINEPTR,
      ABC: GET + NEWTEXT + Q + WORD , PUT + BLOCK + LINEPTR + WORD ,
        MARKED + WORD ,
          ( EQUAL + Q + L :
      INCR +Q , INCR + LINEPTR , INCR + DBPTR ,#ADD + LINEPTR + 1 + X ,
      PUT POINTER IN LINE ENTRY + DBPTR + X , #ABC ) :
    INCR + Q , INCR + LINEPTR , DECR + BFS , #ABC .



TEXT COMPACTION + Q - P1 - X - Y - CT :
   EQUAL + Q + 0 ;
   MAKE + DBPTR + 1 , MAKE + P1 + 1 , MAKE + CT + 1 ,
   ABC: GET LINE ENTRY IN BLOCK DIRECTORY + DBPTR + X + Y ,
    (LESS + FLN + X , MAKE + FLB + X :     ) ,
   EQUAL + X + MINUS ONE ,
     DECR + DBPTR , GET LINE ENTRY IN BLOCK DIRECTORY + DBPTR +X +Y ,
        MAKE + LLB + K ,
      KKK: ADD + P1 + CT + P1 , SUBTR + TEXTSIZE + P1 + BFS ,
           FILL BACK TEXT + P1 , PUT ENTRY IN MFD :
   EQUAL + X + LLB , COPY UP LINE OF TEXT + P1 + Y + CT , #KKK :
   COPY UP LINE OF TEXT + P1 + Y + CT , INCR + DBPTR , #ABC .

BLOCK COMPACTION - Q :
   DIRECTORY COMPACTION + Q , TEXT COMPACTION + Q .

RESTRUCTURE BLOCK :
   EQUAL + FLN + MINUS ONE , DELETE ENTRY FROM MFD + BCC ;
   BLOCK COMPACTION . WRITE BLOCK .
```

```
(   5.5. BLOCK SPLITTING ACTIONS  )


*ACTION*    FIND PLACE OF SPLITTING ,
            FIND LINE LIMITS FOR THE NEW BLOCK ,
            SAVE LINE NUMBERS AND TEXT OF LINES ,
            RESTORE LINE NUMBERS AND TEXT OF LINES ,
            REARANGE  OLD BLOCK ,
            BLOCK SPLITTING ,


IS LINE WITHIN BLOCK CURRENTLY IN CORE * X :
   EQUAL * BCC * HDN , LSEQ * FLB * X :
   LSEQ * FLB * X , LSEQ * X * LLB .

IS IT A REQUEST TO REPLACE AN EXISTING LINE * X * Y - Z :
   IS ENTRY THERE * X * Z , MAKE * Y * Z .

FIND PLACE OF SPLITTING * P * N1 * N2 - X - Y - D - R :
   MAKE * DBPTR * 1 ,
   ABC: GET LINE ENTRY IN BLOCK DIRECTORY * DBPTR * X * Y ,
     MARKED * X , DIVREM * DBPTR * 2 * D * R , ADD * D * 1 * P ,
         ( EQUAL * R * 0 , MAKE * N2 * D :
                           MAKE * N2 * P ) ,
                              SUBTR * DBPTR * N2 * N1 :
     !ABC .

FIND LINE LIMITS FOR THE NEW BLOCK * L1 * L2 * P * N2 - X :
   GET LINE ENTRY IN BLOCK DIRECTORY * P * L1 * X ,
      ADD * P * N2 * Y ,
         GET LINE ENTRY IN BLOCK DIRECTORY * Y * L2 * X .

SAVE LINE NUMBERS AND TEXT OF LINES * P * N2 - X - Y - LN - PTR - Q :
  MAKE * X * 1 , MAKE * Y * 1 , MAKE * P1 * P ,
    ABC: GET LINE ENTRY IN BLOCK DIRECTORY * P1 * LN * PTR ,
     PUT * NEWLINE * X * LN ,COPY OF LINE OF TEXT * PTR * Y * Q ,
       EQUAL * X * N2 :
       ADD * Y * Q * Y , INCR * X , INCR * P1 ,  !ABC .

RESTORE LINE NUMBERS AND TEXT OF LINE * N2 - X - Y - LN :
   MAKE * X * 1 , MAKE * DBPTR * 1 , MAKE * Y * 1 , MAKE * FLN * CLINE ,
   ABC: GET * NEWLINE * X * LN ,
     PUT LINE ENTRY IN BLOCK DIRECTORY * DBPTR * LN * LINEPTR ,
      RESTORE TEXT OF LINE * Y ,
        EQUAL * X * N2 , MAKE * LLB * LN :
        INCR * X , INCR * DBPTR ,  !ABC .

REARANGE OLD BLOCK * P * N2 - P1 - N :
   MAKE * P1 * P , ADD * P * N2 * N ,
     ABC: DELETE LOGICALLY LINE FROM BLOCK * P1 ,
      EQUAL * P1 * N , BLOCK COMPACTION ;
      INCR * P1 , !ABC .

BLOCK SPLITING - P - N1 - N2 :
  FIND PLACE OF SPLITTING * P * N1 * N2 ,
   SAVE LINE NUMBERS AND TEXT OF LINES * P * N2 ,
     REARRANGE OLD BLOCK * P * N2 ,
      REQUEST FOR A NEW BLOCK , INCR * BCC ,
        RESTORE LINE NUMBERS AND TEXT OF LINES * N2 ,
          INSERT ENTRY IN MFD * BCC * FLB * LLB * BFS .
```

```
(  6. THE EXECUTION OF EDITOR COMMANDS. )

(  6.1. EXECUTION OF THE GROUP LIST,DELETE,TEXT REPLACEMENT,SAVE.)

*ACTION* EXECUTE , COMMAND EXECUTION , DO IT .

EXECUTE + CODE + SLINE + LLINE + SCOL + ECOL :
  EQUAL*CODE+LCHAR,LIST BLOCK+SLINE+LLINE+SCOL+ECOL :
  EQUAL*CODE+DCHAR,DELETE FROM BLOCK+SLINE+LLINE+SCOL+ECOL :
  EQUAL*CODE+TCHAR,REPLACE TEXT IN BLOCK+SLINE+LLINE+SCOL+ECOL :
  EQUAL*CODE+SCHAR,SAVE BLOCK+SLINE+LLINE+SCOL+ECOL :
  MSG+EE1 .

COMMAND EXECUTION+CODE+SL+EL+SC+EC-LN-X-Y-BLOCKNR :
  EQUAL*ERRORSW+1 :
  MAKE+LN+SL , FIND BLOCK+BLOCKNR+LN ,
     ABC: READ EDIT FILE BLOCK+BLOCKNR ,
          EXECUTE+CODE+SL+EL+SC+EC ,
             LSEQ+EL+LLB :
             INCR+BLOCKNR,GET NEO ENTRY FOR BLOCK+BLOCKNR+LN+X+Y,
             !ABC .

DO IT + CODE - SL -EL - SC - EC :
  READ SELECTIVE PARAMETERS OF COMMAND+SL+EL+SC+EC+CODE,
    COMMAND EXECUTION+CODE+SL+EL+SC+EC .


(  6.1.1.  LIST COMMAND EXECUTION  )

*ACTION* LIST LINE NUMBER,LIST TEXT OF LINE,LIST BLOCK.

IS SEARCH TEXT PRESENT :
  EQUAL*INPSW+1 .

DOES LINE CONTAIN SEARCH TEXT+A+B+C-X-Y-M-CNT-LCNT-W1-W2-C1-C2-CH1-CH2 :
  DIVREM+B+9+X+Y,MAKE+CNT+1,MAKE+LCNT+1,ADD+X+A+W1,MAKE+W2+1,MAKE+C1+1,
  (EQUAL*Y*0,MAKE+C1+9,DECR+W1:
                    MAKE+C1+Y),
  A1: DECODE WORD INTO CHARACTERS+BLOCK+W1+CHARACTERS1,
      DECODE WORD INTO CHARACTERS+INPTEXT+W2+CHARACTERS2,
      ABC: GET+CHARACTERS1+C1+CH1 , GET+CHARACTERS2+C2+CH2 ,
        EQUAL*CH1+ECOLCHAR,
             LESS+CNT+INCNT , FALSE :
        ADD+CNT+LINECNT+M, LESS+EC+M, FALSE :
        (EQUAL*CH1*CH2,INCR+C1,INCR+C2,INCR+CNT,
        (LESS+INCNT+CNT, TRUE:
          (C1: LESS+C1+9,INCR+C1,INCR+LINECNT,
            B1: (LESS+C2+9,INCR+C2, !ABC:
                 MAKE+C2+1,INCR+W2, !A1):
            MAKE+C1+1,INCR+W1, !B1)):
            MAKE+C2+1,MAKE+W2+1,MAKE+CNT+1, !C1).

IS LINE TO BE CONSIDERED *POINTER+SCOL+ECOL :
  IS SEARCH TEXT PRESENT ,
     DOES LINE CONTAIN SEARCH TEXT+POINTER+SCOL+ECOL .
```

```
LIST LINE NUMBER + LN :
  EQUAL + PSUP + 1 :
    OUTPUT LINE NUMBER + LN .

LIST TEXT OF LINE + TXPTR - Q - CNT :
  MAKE+CNT+1,
  ABC: DECODE WORD INTO CHARACTERS+BLOCK+TXPTR+CHARACTERS1,
       MAKE+Q+1,
       NXT: GET+CHARACTERS1+Q+CHAR,
            EQUAL+CHAR+DOLCHAR :
            PRSYM+CHAR,INC:+CNT,INC:+TXPTR,
                 EQUAL+1+9, !ABC :
                 INCR+Q,
                  (LSEQ+CNT+LINESIZE, !NXT:
                   MSG+ERR 35),

LIST LINE + LINE NUMBER + TXPTR + SCOL (ECOL :
   IS LINE TO BE CONSIDERED+TXPTR+SCOL+ECOL,
      LIST LINE NUMBER+LINE NUMBER,
        LIST TEXT OF LINE+TXTPTR .

LIST BLOCK+SL+EL+SC+EC-LINENB+POINTER:
  (EQUAL+EDIT FILE EMPTY+0,MSG+ERR15: ) ,
  (IS ENTRY THERE+SL+DBPTR:
   MSG+ERR31),
    ABC: GET LINE ENTRY IN BLOCK DIRECTORY+DBPTR+LINENB+POINTER,
         LIST LINE+LINENB+POINTER+SC+EC,
            LESS+LINENB+EL,LESS+LINENB+LL3,
               INCR+DBPTR, !ABC:

VETO PROCESSING+TXTPTR+LN:
  EQUAL+PVETO+1,LIST LINE NUMBER+LN,LIST TEXT OF LINE+TXTPTR,
     WAIT REPLY, EQUAL+REPLY+YES.


(  6.1.2. DELETE COMMAND EXECUTION   )


*ACTION*  DELETE LINE , DELETE FROM BLOCK .


DELETE LINE+P+COL1+COL2+LN-Y :
   GET LINE ENTRY IN BLOCK DIRECTORY+P+LN+Y,
     IS LINE TO BE CONSIDERED+Y+COL1+COL2,
       VETO PROCESSING+P+LN,
         DELETE LOGICALLY LINE FROM BCC:  .


DELETE FROM BLOCK+SL+EL+SC+EC-LN :
    (IS ENTRY THERE+SL+DBPTR:
      MSG+ERR31) ,
  ABC:  DELETE LINE+DBPTR+SC+EC++LN ,
    LESS+LN+LLB,
      (LESS+LN+EL,INCR+DBPTR,  !ABC :
       NXT: RESTRUCTURE BLOCK) :
   !NXT .
```

```
(   6.1.3. TEXT REPLACEMENT COMMAND EXECUTION   )


*ACTION*  REPLACE TEXT IN BLOCK,REPLACE TEXT IN LINE .


REPLACE TEXT IN LINE*POINTER*SC*EC*LN:
 IS LINE TO BE CONSIDERED*POINTER*SC*EC,
    VETO PROCESSING*POINTER*LN ,
      ACTUAL TEXT REPLACEMENT*POINTER*SC*EC.


REPLACE TEXT IN BLOCK*SL*EL*SC*EC-LINENB-POINTER:
  (IS ENTRY THERE*SL*DBPTR;
   MSG*ERR31 ),
 ABC:  GET LINE DESCRIPTOR*DBPTR*LINENB*POINTER,
       REPLACE TEXT IN LINE*POINTER*SC*EC*LINENB,
       LSEQ*LINENB*LLB,LSEQ*LINENB*EL,INCR*DBPTR,!ABC;  .


(   6.1.4. SAVE COMMAND EXECUTION   )


*ACTION* TRANSFER LINE FROM BLOCK TO BUFFER ,
         SAVE BLOCK .


TRANSFER LINE FROM BLOCK TO BUFFER*POINTER*LN*SC*EC-X-Y:
  IS LINE TO BE CONSIDERED*POINTER*SC*EC,
    VETO PROCESSING*POINTER*LN,
      SUBTR*INPUTRECORDSIZE*PBUFFER*X,
LSEQ*LINESIZE*X,
 (ABC: SAVE LINE NUMBER*LN,MAKE*Y*POINTER,
   NXT:  GET*BLOCK*Y*CHAR1,PUT*BUFFER*POINTER*CHAR1,
     MARKED*CHAR1,MAKE*POINTER*X;
     INCR*Y,INCR!PBUFFER, !NXT) ;
 WRITE OUTPUT RECORD*RECNBR,INC*RECNBR,MAK *PBUFFER*1,
    !ABC .


SAVE BLOCK*SL*EL*SC*EC-POINTER-LINENB :
 (IS ENTRY THERE*SL*DBPTR!MSG*ERR31) ,
  MAKE*PBUFFER*1,
 ABC: GET LINE ENTRY IN BLOCK DIRECTORY*DBPTR*LINENB*POINTER,
  TRANSFER LINE FROM BLOCK TO BUFFER*POINTER*LINENB*SC*EC,
  LESS*LINENB*LLB,LESS*LINENB*EL,INCR*DBPTR, !ABC ;
  EQUAL*LINENB*EL;
  EQUAL*LINENB*LLB,MAKE*DBPTR*1.
```

```
(   6.2.  EDIT COMMAND EXECUTION      )


*ACTION* TRANSFER LINE FROM INPUT BUFFER TO BLOCK,
         EXECUTE EDIT COMMAND .


TRANSFER LINE FROM INPUT BUFFER TO BLOCK :
 AAA: EQUAL*PSEQUENCE*1,
  (ABC:  LSEQ*DBPTR*MEM,LSEQ*LINESIZE*BFS,
     (PUT LINE ENTRY IN BLOCK DIRECTORY*DBPTR*CLINE*LINEPTR,
     NXT: GET*BUFFER*PBUFFER*CHAR1,INCR*PBUFFER,
          EQUAL*CHAR1*EOL CHAR,
            STACK LAST*BLOCK*BLKSIZE*LINEPTR,DECR*BFS,
               INCREMENT LINE NUMBER,INCR*DBPTR :
          ADD TO*CHAR1*BLOCK*BLKSIZE*LINEPTR,DECR*BFS, !NXT)!
   REQUEST FOR A NEW BLOCK, !AAA)!
      READ THE SEQUENCE NUMBER OF LINE*CLIN. , !ABC :
      MSG*ERR36 .


EXECUTE EDIT COMMAND
  HAS EDIT FILE SAVED,
    (TURN OFF SAVED FLAG,MAKE*FLINE*OFLINE,MAKE*BOD*L,
    BLOCK INITIALISATION,MAKE*Q*1,MAKE*EDIT FILE EMPTY*1,
     NXT: READ INPUT FILE RECORD*U,
       IS IT END OF FILE,CLOSE EDIT FILE!
       INCR*Q,
        ABC: TRANSFER LINE FROM INPUT BUFFER TO BLOCK,
             EQUAL*PBUFFER*INPUTRECORDSIZE, !NXT :
             INCR*Q, !ABC!:
  MSG*ERR35 , TURN ON SAVED FLAG.
```

```
(   6.3.  ADD COMMAND EXECUTION   )


*ACTION*  PROCESS THE REQUEST TO ADD A LINE WITH A GIVEN LINE NUMBER,
          EXECUTE ADD COMMAND .


PROCESS THE REQUEST TO ADD A LINE WITH A GIVEN LINE NUMBER:
    MAKE*EDIT FILE EMPTY * 1,
 STR: IS LINE WITHIN BLOCK CURRENTLY IN CORE*CLINE,
 (QQQ: IS IT A REQUEST TO REPLACE AN EXISTING LINE*CLINE*PTR,
   (LSEQ*LINESIZE*BFS,
     ABC: PUT LINE ENTRY IN BLOCK DIRECTORY*PTR*CLINE*LINEPTR,
          READ TEXT OF LINE AND STORE IT IN BLOCK :
    DELETE LOGICALLY LINE FROM BCC*PTR,BLOCK COMPACTION,
    (NXT: LSEQ*LINESIZE*BFS,
      FIND PLACE OF INSERTION IN DIRECTORY*CLINE*PTR, :ABC :
     BLOCK SPLITTING, :STR )):
     :NXT ):
 WRITE EDIT FILE BLOCK,FIND BLOCK*BLKID*CLINE,
   READ EDIT FILE BLOCK*BLOCKID , :STR .


EXECUTE ADD COMMAND*ALINE*AINCR:
 (EQUAL*PINCR*1,MAKE*INCR*A.NCR: ) ,
   MAKE*CLINE*ALINE,
 ABC: IS IT A REQUEST TO REPLACE AN EXISTING LINE*CLINE*Y,
       MSG*ERR1 ;
     IS END OF CREATE MODE :
     OUTPUT LINE NUMBER*CLINE,
         PROCESS THE REQUEST TO ADD A LINE WITH A GIVEN LINE NUMBER,
           INCREMENT LINE NUMBER, :ABC .


(    6.4.  RESEQUENCE COMMAND EXECUTION    )


*ACTION*  EXECUTE RESEQUENCE COMMAND .


EXECUTE RESEQUENCE COMMAND*RLINE*RINCR-X-Y-BLKID:
  MAKE*BLKID*1,MAKE*CLINE*RLINE,MAKE*INCR*RINCR,MAKE*FLN*CLINE,
  ABC: READ EDIT FILE BLOCK*BLKID,MAKE*OBPTR*1,MAKE*FLB*CLINE,
   NXT: GET LINE ENTRY IN BLOCK DIRECTORY*OBPTR*X*Y,
        PUT LINE ENTRY IN BLOCK DIRECTORY*OBPTR*CLINE*Y,
        EQUAL*X*LLB,MAKE*LLB*CLINE,PUT MFD ENTRY FOR BCC,
         (EQUAL*BLKID*HBN,MAKE*LLN*CLINE,WRITE BLOCK*BLKID:
          INCR*BLKID,INCREMENT LINE NUMBER,WRITE BLOCK*BLKID,:ABC):
        INCR*OBPTR,INCREMENT LINE NUMBER,:NXT .


(  6.5. RUN COMMAND EXECUTION  )


*ACTION* EXECUTE RUN COMMAND.


EXECUTE RUN COMMAND :
 (EQUAL*PFILE*0,MAKE*FILE NAME * EDIT: ) ,  RUN PROGRAM .
```

```
(  7.  EDITOR NUCLEUS     )

(  7.1. COMMAND PROCESSING  )

CREATE COMMAND :
   IS KEYWORD + CREATE ,
      READ OPTIONAL PARAMETERS OF CREATE COMMAND .

ADD COMMAND - ALINE - AINCR :
   IS KEYWORD + ADD ,
      READ OPTIONAL PARAMETERS OF ADD COMMAND + ALINE + AINCR ,
        EXECUTE ADD COMMAND.

LIST COMMAND - W :
   IS KEYWORD + LIST ,
      MAKE + W + LCHAR , DO IT + W .

DELETE COMMAND - W :
   IS KEYWORD + DELETE ,
      MAKE + W + DCHAR , DO IT + W .

SAVE COMMAND - W :
   IS KEYWORD + SAVE ,
      MAKE + W + SCHAR , DO IT + W .

TEXT REPLACEMENT COMMAND - W :
   IS TEXT REPLACEMENT ,
      MAKE + W + TCHAR , DO IT + W .

EDIT COMMAND :
   IS KEYWORD + EDIT ,
     READ NAME OF SOURCE FILE ,
         READ OPTIONAL PARAMETERS OF EDIT COMMAND ,
            EXECUTE EDIT COMMAND .

FORMAT COMMAND :
    IS KEYWORD + FORMAT ,
      READ OPTIONAL PARAMETERS OF FORMAT COMMAND .

RESEQUENCE COMMAND - RLINE - RINCR :
    IS KEYWORD + RESEQUENCE ,
      READ OPTIONAL PARAMETERS OF RESEQUENCE COMMAND +RLINE + RINCR ,
        EXECUTE RESEQUENCE COMMAND + RLINE + RINCR .

RUN COMMAND :
    IS KEYWORD + RUN ,
       READ NAME OF PROCESSOR ,
          READ OPTIONAL PARAMETERS OF RUN COMMAND ,
             EXECUTE RUN COMMAND .

OTHER EDITOR COMMANDS :
   LIST COMMAND ;
   SAVE COMMAND ;
   DELETE COMMAND ;
   EDIT COMMAND ;
   TEXT REPLACEMENT COMMAND ;
   ADD COMMAND ;
   RESEQUENCE COMMAND ;
   RUN COMMAND ;
   FORMAT COMMAND .
```

```
EDITOR TERMINATION :
    IS KEYWORD + BY .

INTERCOM COMMAND :
   ANY OF THE INTERCOM KEYWORDS , INTERCOM ACTION .

ADD A LINE TO THE EDIT FILE :
  IS A VALID LINE NUMBER + X , MAKE + CLINE + X ,
     PROCESS THE REQUEST TO ADD A LINE WITH A GIVEN LINE NUMBER .


(  7.2. THE HEART OF THE EDITOR )

*ACTION*  START EDITOR , COMMAND MODE , CREATE MODE .

START EDITOR :
   MAKE*EDIT FILE EMPTY *0,TURN ON SAVED FLAG,
      MAKE*BCC*0.
CREATE MODE :
 WAS EDIT FILE SAVED ,
    ( TURN OFF SAVED FLAG , CREATE MODE INITIALISATION ,
    ABC: OUTPUT LINE NUMBER + CLINE ,
      IS A REQUEST TO ADD A LINE WITH A GIVEN LINE NUMBER + X ,
        SAVE CURRENT LINE NUMBER + X ,
          PROCESS THE REQUEST TO ADD A LINE WITH GIVEN LINE NUMBER ,
            RESTORE CURRENT LINE NUMBER , !ABC) !
      IS END OF CREATE MODE , CLOSE EDIT FILE ;
      WRITE ENTRY IN NEXT AVAILABLE LOCATION OF BCC DIRECTORY ,
         READ TEXT OF LINE FROM THE TERMINAL AND STORE IT IN BLOCK ,
           INCREMENT LINE NUMBER, !ABC) !
MSG * ERR1  , TURN ON SAVED FLAG .


COMMAND MODE :
   NXT: READY FOR COMMAND ,
     ABC: READ KEYWORD , COMMAND MODE INITIALISATION ,
       CREATE COMMAND ;
       ADD A LINE TO THE EDIT FILE , !ABC ;
       OTHER EDITOR COMMAND , !NXT ;
       INTERCOM COMMAND , !NXT ;
       EDITOR TERMINATION , EXIT TO INTERCOM MODE ;
       MSG * ERR18  , !NXT .


EDITOR :
   START EDITOR ,
      NXT: COMMAND MODE ,
          CREATE MODE , !NXT .

*RESULT* EDITOR .
```

Received by Publishing Department
on March 30, 1977.