nMOS

L1 MOS

SORT/MERGE Utility
Reference Manual

olivetti

PREFACE

This manual provides the necessary information for the execution of the SORT/MERGE utility. It describes the use of utility commands and explains how to run the utility.

For a detailed description of the utility's features, see "Introduction to MOS" manual.

SUMMARY

The manual contains a General Introduction in which advice is given on operational aspects and on how to make best use of the utility.

It contains a reference guide to the commands that make up the utility language and the two Shell commands that activate the two utility phases.

Examples for the use of the utility and the full list of error messages complete the manual: both language and sort/merge error messages are given.

REFERENCES

Read first...

Introduction to MOS
Code 4002130 G (vol. 2)

MOS SHELL Commands Reference Manual
Code 4002770 Q

For further information, read...

Glossary/Glossario
Code 4002140 H (vol. 1)

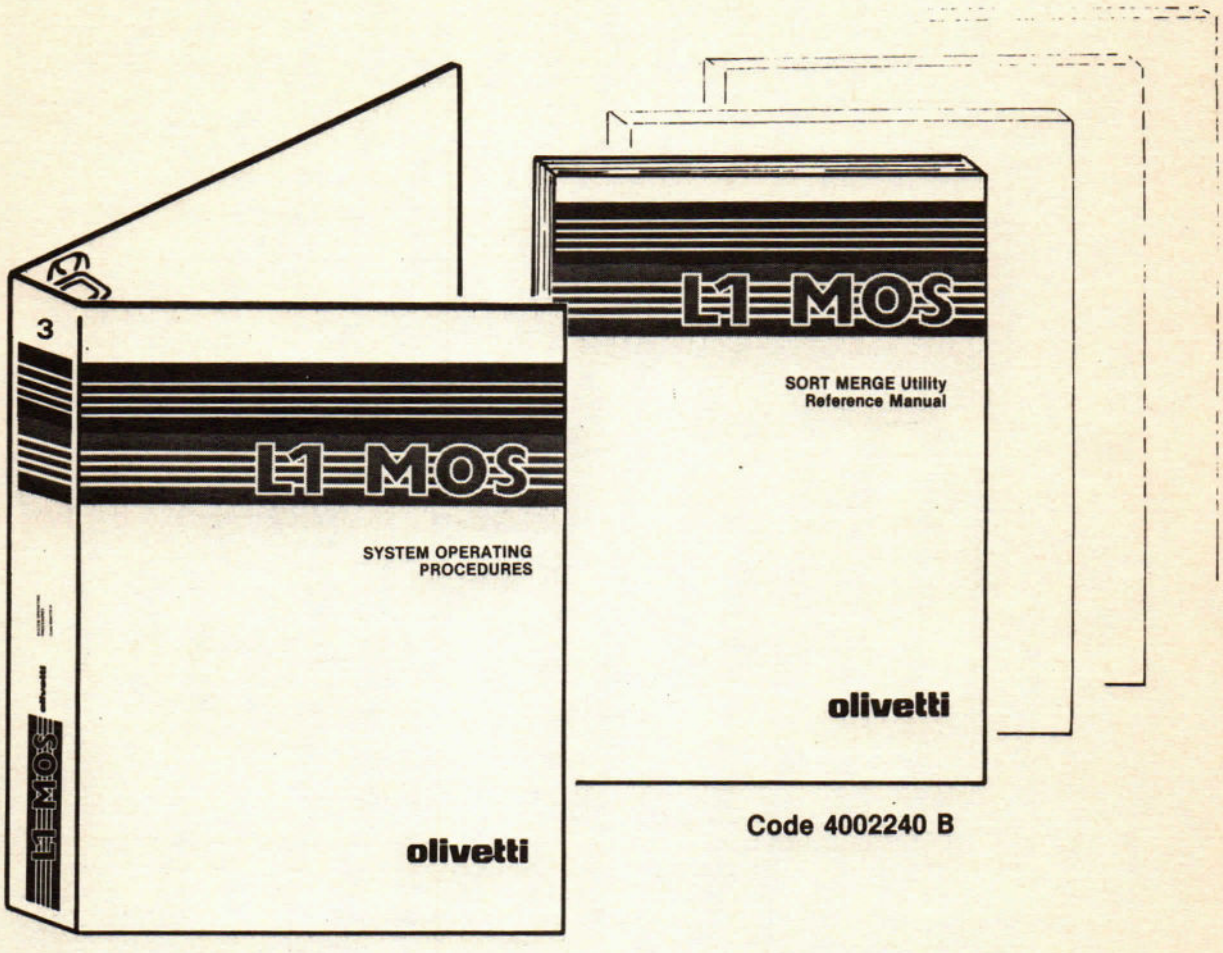
First Edition: January, 1983
Release 1.Ø

First Update: October, 1983
Release 2.Ø

Second Update: April, 1984
Release 3.0

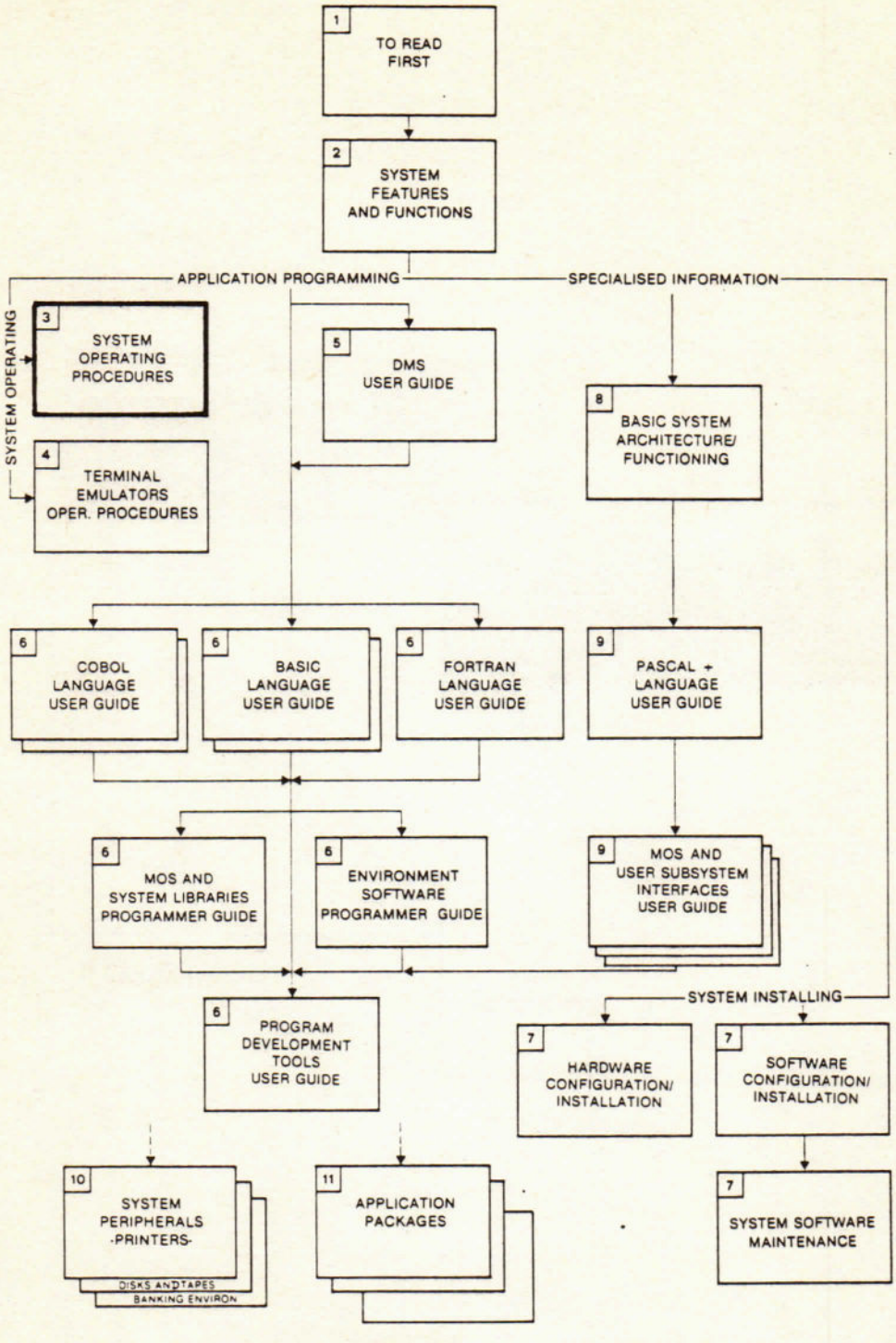
PUBLICATION ISSUED BY:

Ing. C. Olivetti & C., S.p.A.
Direzione Documentazione
77, Via Jervis - 10015 IVREA (Italy)



Code 4002240 B

Code 4004770 H


1
TO READ FIRST
2
SYSTEM FEATURES AND FUNCTIONS
APPLICATION PROGRAMMING
SPECIALISED INFORMATION
SYSTEM OPERATING
3
SYSTEM OPERATING PROCEDURES
4
TERMINAL EMULATORS OPER. PROCEDURES
5
DMS USER GUIDE
8
BASIC SYSTEM ARCHITECTURE/ FUNCTIONING
6
COBOL LANGUAGE USER GUIDE
6
BASIC LANGUAGE USER GUIDE
6
FORTRAN LANGUAGE USER GUIDE
9
PASCAL + LANGUAGE USER GUIDE
6
MOS AND SYSTEM LIBRARIES PROGRAMMER GUIDE
6
ENVIRONMENT SOFTWARE PROGRAMMER GUIDE
9
MOS AND USER SUBSYSTEM INTERFACES USER GUIDE
6
PROGRAM DEVELOPMENT TOOLS USER GUIDE
SYSTEM INSTALLING
7
HARDWARE CONFIGURATION/ INSTALLATION
7
SOFTWARE CONFIGURATION/ INSTALLATION
10
SYSTEM PERIPHERALS -PRINTERS-
DISKS AND TAPES
BANKING ENVIRON
11
APPLICATION PACKAGES
7
SYSTEM SOFTWARE MAINTENANCE

Title: L1 MOS SORT/MERGE Utility Reference Manual

Newsletter Code:	4002242 Q
Date:	April, 1984
Publication Code:	4002240 B (0)
Previous Newsletters:	4002241 C

This Newsletter provides updated pages for the subject publication.

The last level completed on the attached form, Updating Status, indicates the pages to be added, removed, or replaced, the number of pages included, and the Newsletter Code. Pages marked with an asterisk should be removed from the publication. The form should be filed at the back of the publication as a permanent record of amended pages.

Each amended page is identified by the Newsletter Code shown above.
Amended pages remain valid unless otherwise noted in a subsequent Newsletter. Modifications to text, figures, or tables are indicated by a vertical bar in the outside margin next to the change.

Summary of Amendments:

- The SORTRUN command now includes completion code handling.
- There are additions and changes in Appendix B: SORT/MERGE error messages.

Ing. C. Olivetti & C., S.p.A. - Direzione Documentazione - Via Jervis 77 10015 IVREA (TO) Italy

Printed in Italy

olivetti

UPDATING STATUS

LEVEL	DATE	UPDATED PAGES	PAGES	CODE
0	83.01.15		80	4002240 B
1	83.10.01	Preface, i, ii, Gen.Intr., 0.1.1,0.3.1, 1.0.2,1.0.3,1.3.1,1.3.4,1.3.7,1.3.12, 1.4.1,1.5.1,1.7.1,1.7.2,1.8.1,1.10.1,2.0.1, 2.0.2,2.2.1,2.2.2,A.0.3,A.0.5,A.0.6,A.0.8, A.0.9,A.0.12,A.0.15 ÷ A.0.18,B.0.1 ÷ B.0.5.	37	4002241 C
2	84.04.09	Preface, 2.0.1, 2.0.2, B.0.2 ÷ B.0.6.	8	4002242 Q
3				
4				
5				
6				
7				
8				

Pages marked * must be suppressed

GENERAL INTRODUCTION

GENERAL INTRODUCTION

SORT/MERGE is a utility provided for:

- ordering one or more positional input files, according to keys and criteria decided by the user, so as to produce a single ordered file, which is also positional (SORT).

- merging up to four input files, ordered by the same keys, in a single output positional file (MERGE).

The utility is executed in two phases:

- interactive phase (invoked by the Shell command LANGUAGE) in which the SORT/MERGE commands keyed in by the operator are analysed by a Language Analyzer from a formal and syntactical point of view. Possible errors are pointed out at the end of each command or line (language errors) and the user is generally given the chance to correct the mistake at once and continue keying commands.

- execution phase (invoked by the Shell command SORTRUN) in which the algorithm (SORT or MERGE) chosen by the operator in the interactive phase is run according to the user defined parameters and options. Some errors made in the preceding phase, which only can be detected at run time, are pointed out here, together with proper Sort errors and errors passsed by File System. In these cases the utility normally aborts, unless there is a different warning.

REQUIREMENTS

Configuration

SORT/MERGE can run on every L1 configuration in which are present :

- two disk devices (1 for O.S. and SORT, 1 for the user files)
- a console

Memory

Memory requirements of the program (not including O.S.) vary for interactive and execution phase:

'LANGUAGE' The interactive phase, activated by LANGUAGE command, requires at least 48k of memory to run

'SORTRUN' The execution phase, activated by SORTRUN command, requires at least 42k for SORT; the requirement for MERGE depends on the number of files to be merged but it is at least 42k.

SORT performance will improve with any increase in memory over the minimum required. There is no maximum memory limit; however, beyond a certain ratio between memory and file size, the marginal increase in performance due to the addition of further memory will lessen; extra memory above 64K will not be utilized.

Apart from the sort/merge of files, the utility provides a large selection of extra functions that are described in the manual entitled 'Introduction to MOS".

A brief explanation follows on how to employ the utility to perform some specific functions. In the examples only the commands (or parameters) which actually carry out the requested functions are described, and not those which are not concerned with that function, even if they are compulsory in an execution (like INPUT, or Key-Def).
To make the best use of this paragraph, the reader should look for a specific function (e.g. Record Selection), read the text and see which command or parameter carries out that function. He should then read in the "Commands" chapter the description of the command or parameter involved, and at the end read the example.
This familiarises the reader with the concepts and facilities of the utility before trying to use it.

Record Selection

A file containing three different kinds of records is to be sorted; only two of these must be written in output.
Selection will be carried out by using the Rec-Type INPUT command in which the conditions that identify a record as belonging to each record-type are defined.

The parameter will be used to define only two types of record (e.g. 1 and 3).

```
INPUT : 1 20='IMP'
      : 3 20='OPE'
```

As Rec-Type 2 is not defined, it will be excluded from output.

Record Retrieval

A similar case is that of a file containing a record with only one layout, from which a subset of records must be retrieved and sorted; e.g. the records pertaining to children born in 1968, from a statistics data file. Rec-Type parameter will again

be used to define only one record.

```
INPUT : 1 70='1968'
```

This will exclude any other record from output.

Keys in Different Positions

A file made up of records with sorting keys in different positions can be sorted by using INPUT command blocks Rec-Type and Key-Def.
Rec-Type defines the types of records to be sorted and Key-Def the key position in each one of these.

```
INPUT : 1 10=45(10)
          K1 1-5 E
      : 2 10<>45(10)
          K1 17-21 E
```

In both kinds of record the key is a 5 character EBCDIC field : in the first type it is in positions 1-5, whilst in the second it is in positions 17-21. Note that the second type could also have been defined as a result as

```
:2
```

without any condition.

Field Selection

If the user wants to sort a Personnel data file composed of records of only 1 type and exclude from the output layout the 'paternity' field (positions 20-29 in the input record) he must use the INPUT command block Field-Def.

```
INPUT : 1
        F1 1-19
        F2 30-200
```

The unique record-type is unconditionally defined, as no record selection is desired.

As field 20-29 is not defined it will be excluded automatically from the output layout which will therefore be 190 characters long.

Apart from the sort/merge of files, the utility provides a large selection of extra functions that are described in the manual entitled 'Introduction to MOS".

A brief explanation follows on how to employ the utility to perform some specific functions. In the examples only the commands (or parameters) which actually carry out the requested functions are described, and not those which are not concerned with that function, even if they are compulsory in an execution (like INPUT, or Key-Def).
To make the best use of this paragraph, the reader should look for a specific function (e.g. Record Selection), read the text and see which command or parameter carries out that function. He should then read in the "Commands" chapter the description of the command or parameter involved, and at the end read the example.
This familiarises the reader with the concepts and facilities of the utility before trying to use it.

Record Selection

A file containing three different kinds of records is to be sorted; only two of these must be written in output.
Selection will be carried out by using the Rec-Type INPUT command in which the conditions that identify a record as belonging to each record-type are defined.

The parameter will be used to define only two types of record (e.g. 1 and 3).

```
INPUT : 1 20='IMP'
      : 3 20='OPE'
```

As Rec-Type 2 is not defined, it will be excluded from output.

Record Retrieval

A similar case is that of a file containing a record with only one layout, from which a subset of records must be retrieved and sorted; e.g. the records pertaining to children born in 1968, from a statistics data file. Rec-Type parameter will again

be used to define only one record.

```
INPUT : 1 70='1968'
```

This will exclude any other record from output.

Keys in Different Positions

A file made up of records with sorting keys in different positions can be sorted by using INPUT command blocks Rec-Type and Key-Def.
Rec-Type defines the types of records to be sorted and Key-Def the key position in each one of these.

```
INPUT : 1 10=45(10)
          K1 1-5 E
      : 2 10<>45(10)
          K1 17-21 E
```

In both kinds of record the key is a 5 character EBCDIC field : in the first type it is in positions 1-5, whilst in the second it is in positions 17-21. Note that the second type could also have been defined as a result as

```
:2
```

without any condition.

Field Selection

If the user wants to sort a Personnel data file composed of records of only 1 type and exclude from the output layout the 'paternity' field (positions 20-29 in the input record) he must use the INPUT command block Field-Def.

```
INPUT : 1
        F1 1-19
        F2 30-200
```

The unique record-type is unconditionally defined, as no record selection is desired.

As field 20-29 is not defined it will be excluded automatically from the output layout which will therefore be 190 characters long.

Totalisation

An invoice file is to be sorted by customer code (positions 1-6 in the record) outputting only one record for each customer, so that the sum of all the amounts in all the records (invoices) belonging to that customer are written in the "amount" field (positions 30-40) of that unique record.
The INPUT command Field-Def block and the SUMMARY command are used for this purpose.

```
INPUT : 1
        K1 1-6 E
        F1 30-40
SUMMARY : F1(12) PD
```

The amount field is defined as a totalisation field in which all the amounts that have the same customer code are added together. In the output record, the field has been lengthened to 12 bytes to avoid overflow.

Collating Sequence

Sorting can be carried out by using two kinds of collating sequences (ASCII or EBCDIC character sets) for character string keys. If, for some reason, the user wants to use different sorting criteria, he can define his own collating sequence by using the ALTSEQ command.
The command will only refer to the name of a file which contains the desired character sequence that the user will have previously created, outside the SORT utility, in byte-stream format.
The one-record, 256 bytes file contains the set of characters which is defined as a collating sequence, in the order decided upon by the user. In this way each character position in the file determines the binary weight that it will have in the SORT. If for instance, the character "?" is in position 64 (hex. '40') in a 256 character string, it will have a binary weight which is equal to '01000000'.

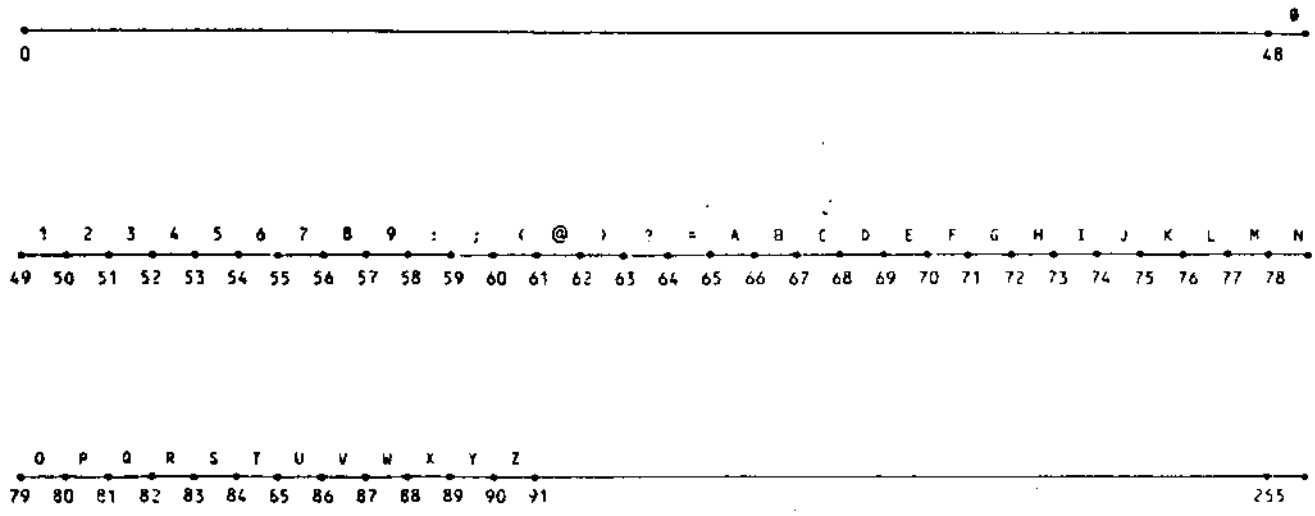


Fig. 0. 1 - User Defined Collating Sequence

This is an example of an alternative collating sequence. For simplicity, only the characters in the 256 bytes file that are of interest to the above example are described.
The figure represents a subset of the ASCII character set, which has been modified so as to reverse the positions (and binary weight) of characters '@' and '='.
The result of this will be that, after SORT, records having their key equal to '@' will precede those with '=', which is the opposite to what usually happens.

Reverse Comparison

For the collating sequences available for character-string keys (ASCII, EBCDIC, or user defined C) reverse comparison is allowed ; that is, the comparison is carried out from right to left. (see R parameter in Key-Def, INPUT command). The result is that, in reverse mode, a character-string key, like ASCII 'AAAZ' will be sorted after the key 'ZZZA'.

Reformatting of Output Layouts

Restructuring of the output layout in one or more types of records can be carried out by using the OUTPUT command and the INPUT command Field-Def block.
Characters 40-45 in the input record are to be moved at the end of the output record and the field key

from positions 20-25 to positions 1-5 with the constant 'IVA' added. This is done by:

```
INPUT : 1
        K1 20-25
        F1  1-19
        F2 26-39
        F3 40-45
        F4 46-100
OUTPUT : 1 K1,'IVA',F1,F2,F4,F3
```

In this way the output record layout can be very different from the original one.

1. COMMANDS

The user interface is a series of JCL-like commands. This chapter contains a brief explanation of the criteria used to describe the commands, a synthesis of the related basic concepts and a list of the commands themselves.

Description Criteria

Description of the commands is in alphabetical order and is structured as follows:

- the name of the command (in the right hand corner).
- the function.
- a syntax diagram using Pascal notation.
- a description of parameters.
- notes.
- examples.

If the command structure is particularly complex (e.g. the INPUT command), the diagram is described using a top-down criterion, so that its elements are expanded and simplified at each step. The parameters are key-words (usually a capital letter followed by a number), whilst their attributes are positional, that is, their meaning is interpreted according to their position inside the parameter.
The separator character inside the attributes is usually '-' (minus); the attributes themselves are always separated by blanks (shown as bl) except when indicated. For some parameters 'new line' must be specified: this is obtained by using ↵ (carriage return sign).
When the command is particularly complex and needs more than one line, the 'line continuation' character '&' must be used, which :

. is a reserved character for the utility

. must be the last character in the line, only

followed by '↵'

. cannot be used to continue a keyword.

note The examples provided illustrate the features and the use of the single command. Recapitulatory examples aimed at highlighting the global use of the utility are given in the appropriate Chapter (Chapter 3).

Basic Concepts The user has at his disposal two types of commands:

- Shell commands (LANGUAGE and SORTRUN) to request execution of each of the two phases (interactive and execution) which form the utility.
- proper SORT/MERGE commands, which form the user interface of the utility. They are used in the interactive phase to supply the Language Analyzer with the execution parameters of the program.

SORT/MERGE
INPUT
OUTPUT
ALTSEQ
MEMORY
SUMMARY
REPEAT
END

Each of these, if used, must be defined only once.

compulsory commands Of these, only

SORT/MERGE
INPUT
END

are compulsory.

command block The subset of commands specified for an execution form the command block of that execution. It can be named, so as to be called later, for re-executing the same utility (see command REPEAT).

command sequence The order in which the commands are defined is compulsory for:

SORT/MERGE
INPUT

OUTPUT
SUMMARY
REPEAT
END

These must be provided in this order if they are present.

The following shows a command sequence with an example of command block:

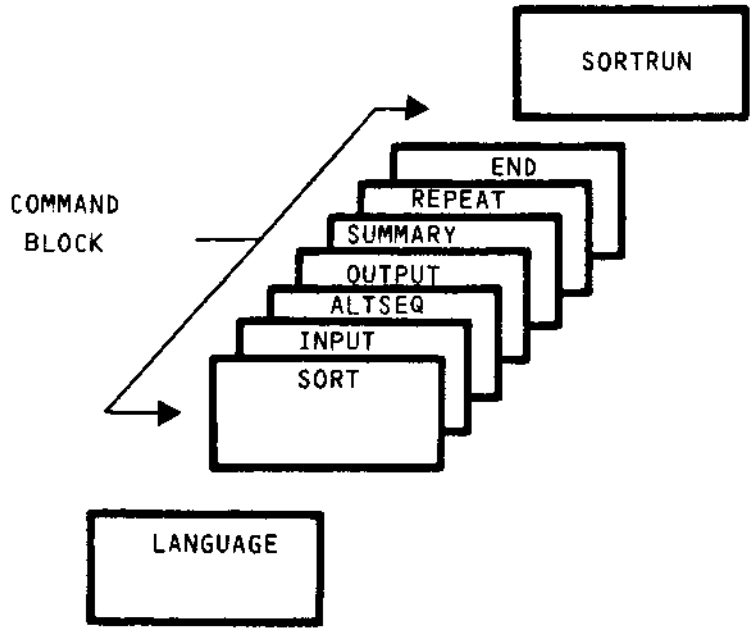
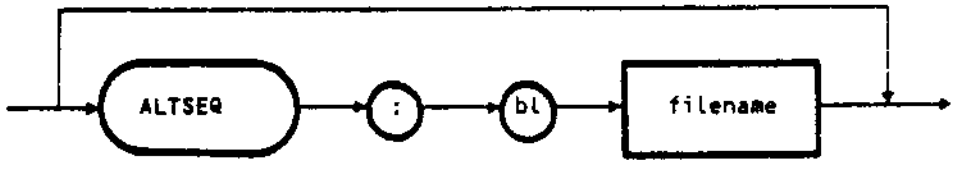


Fig. 1. 1 - Example Command Sequence.

end of block	The command END must be the last in the command block.
Reference Index	The list of SORT/MERGE commands is supplied here, in alphabetical order: a description of the Shell commands is given in chapter 2.

Provides the program with the name of the file containing the collating sequence chosen by the user and used instead of the two standard types that are available for character-string keys (that is, the ASCII or EBCIDIC character set). The command is compulsory if at least one 'C' type key has been defined in INPUT (see the Key-Def block in the INPUT command)



where:

filename — is the name of a 256 bytes, one record file which contains the set of characters that the user provides the program with, as a reference for sorting. This file must have been previously created by the user in byte-stream format.
The 256 positions in the string are in ascending order of binary weight (from hex '00' to hex 'FF'). Therefore the position of each character in the file defines the binary weight that is assigned to it in the sorting sequence. All of the 256 positions must be defined, and each must be different from the other. If only a subset of them is to be used for sorting, the other character positions can be defined with their binary weights.

note 1 — The alterations in the sequence made by ALTSEQ only affect sorting; the output record is unaltered. This means that ALTSEQ does not translate.

example 1

Use of a collating sequence created by the user is described below.

A file containing only one 256 bytes record is defined, and for simplicity the characters that are not to be used are excluded (their place is filled with a continuous line without dots).

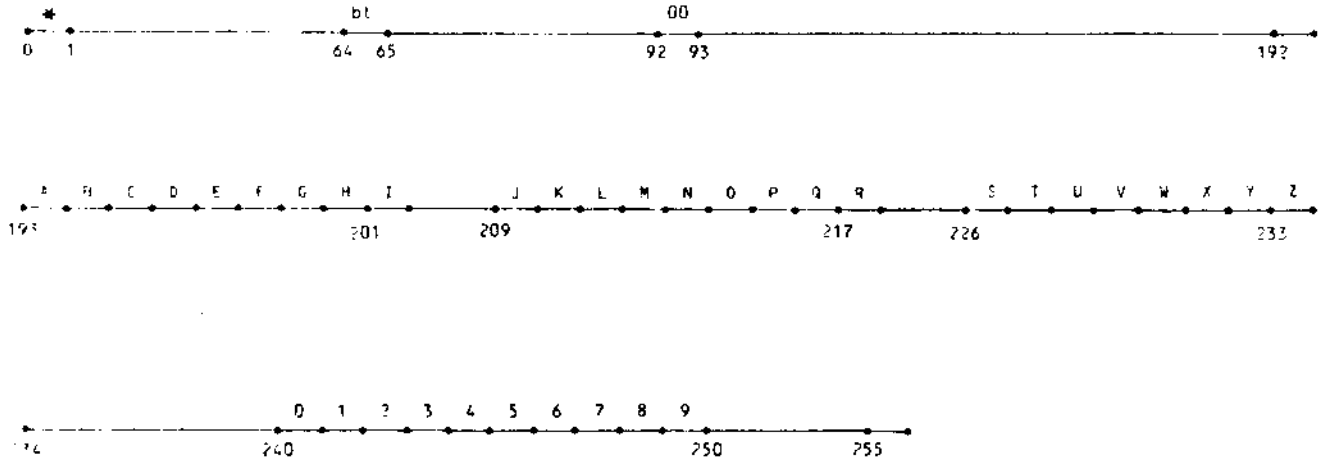


Fig. 1. 2 - Description of an Alternative Collating Sequence

This is the EBCDIC alphanumeric character set, which has been altered so as to reverse the positions of characters 'asterisk' (position 92 in the usual sequence, that is, binary weight hex '5C') and 'binary zeros' (position 0, that is binary weight hex '00' in the usual sequence).
All the other characters stay in their original position and consequently keep their binary weight.
With this weight sequence the SORT order will be:

1. Asterisk

2. Blank

3. Binary zeros

4. The usual alphanumeric EBCDIC sequence.

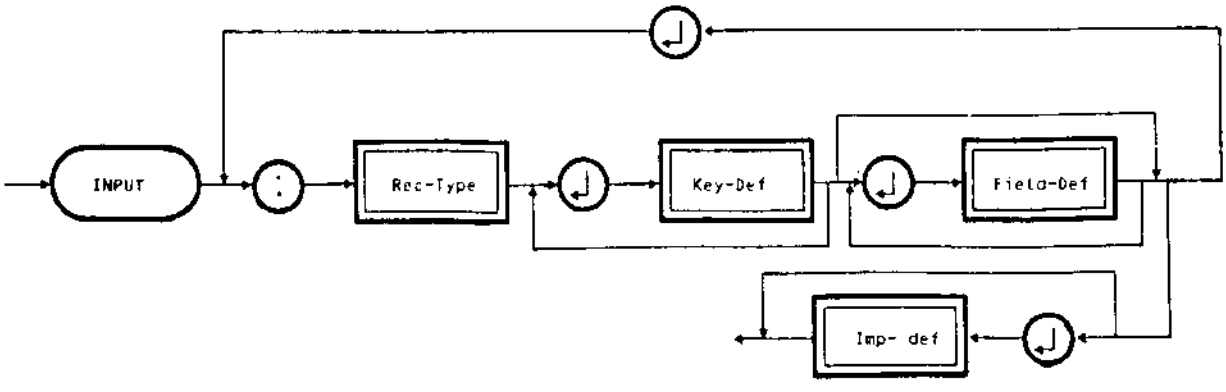
The result of this will be that after SORT, records with asterisks in the key will come before those

with blanks, that is, the opposite to the usual EBCDIC sequence.

END (END OF COMMAND BLOCK)

Defines the end of the command block and must be the last in it.
It is a COMPULSORY command.

Describes the type or types of input records, the sorting keys and the fields to be written in output. This is a COMPULSORY command which must immediately follow that of the SORT/MERGE.



where:

Rec-Type — allows classification of input records as record-types.

Key-Def — describes the format, position, hierarchy and sorting order (ascending/descending and possibly reverse) for the sorting keys.

Field-Def — explicitly describes the position and length of the fields to be written in output.

Imp-Def — implicitly defines the fields to be written in output.
Can only be defined once in the command.

note 1 — The group of 4 blocks define the "record-type block" (preceded by a colon) which contains the characteristics of each record-type.
The 3 blocks Field-Def, Key-Def and Imp-Def form the 'description' of Rec-Type ; their order is not relevant.

note 2 — The total number of fields and/or keys defined in a Rec-Type description must not exceed 128 : for more details see notes 12 and 13.

note 3 — Each record must not be longer than 4095 bytes; all records must have the same length.

note 4 — Key-Defs and/or Field-Defs may not overlap.

Rec-Type

Input records are classified as 'record-type' using conditions; that is, by comparing record fields with other ones in the same record or with constants, using relational operators. Several conditions (up to 128) can be related to each other by logical operators, forming the complex condition.

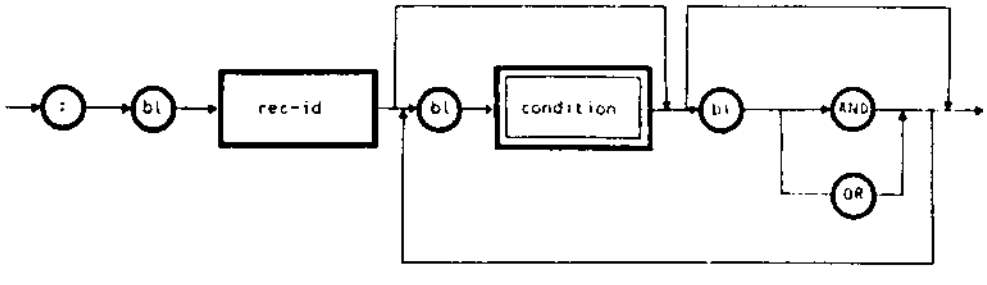
This classification allows :

- differentiated processing of records in different formats
- records reformatting in output (see OUTPUT command)
- the (explicit) inclusion of records in the SORT. Any records that do not belong to the type(s) defined are omitted.

Up to 16 'Rec-Types' can be defined; one is compulsory.

If all the input records are in the same format (or the format itself is irrelevant) and all of them are to be written in output, a record-type must be defined, even though it does not depend on a condition.

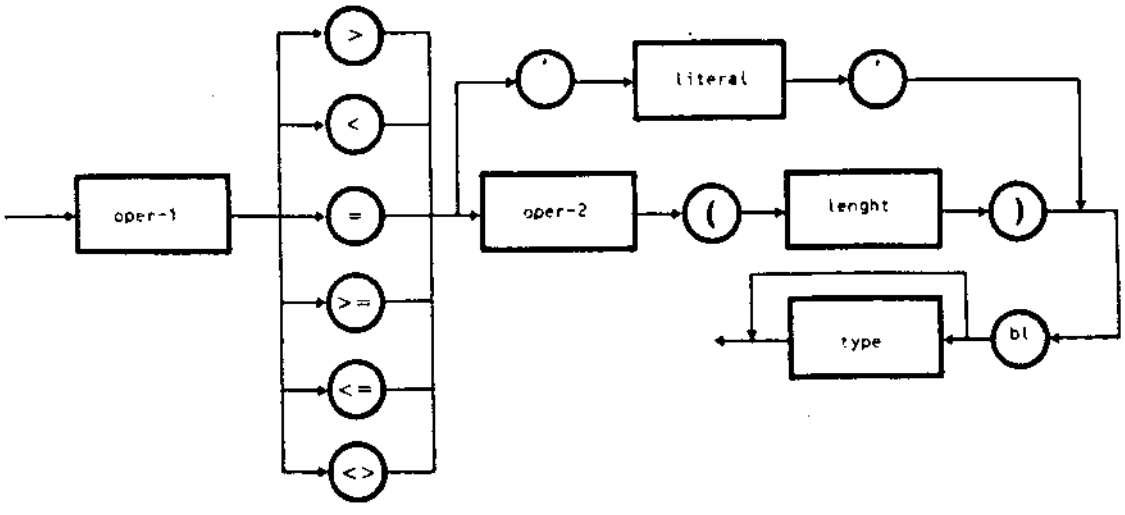
if a Rec-Type is defined without any condition, it must be the last in the INPUT command. The implicit meaning of this by default definition is that all the records not coming into the other Rec-Types fall in the unconditionally defined Rec-Type.



where:

rec-id Identifies the record-type. It is a number from 1 to 16. The numbers need not be in sequence.

Condition



where:

oper-1 is an input record field that is analysed in order to determine the record attribution to the 'rec-id' type.
It is a number whose value indicates the field start position in the record (the first byte is in position 1).

oper-2 is another field in the same record as oper-1, which is used for comparison with the first one.
It is a number whose value indicates its position in the record, as for oper-1.

length The length of the comparison between oper-1 and oper-2.
It is a number in brackets whose value must not be greater than 256. If omitted, the default value is 1 byte, in every case except FP (default value 4 bytes).

literal An alphanumeric constant in inverted commas used as a comparison field. Its length (implicit) determines the length of the comparison, that starts from position oper-1.
The maximum length allowed for a literal is 256

bytes; the global length of the literals defined in a command-block is 800 bytes.

type — Defines the type of fields to be compared, which must be the same for both.
It can assume the following values:

A: ASCII character set (default value)

B: binary integer

E: EDCDIC character set

PD: packed decimal

UD: unpacked decimal

ZO: zone (the four most significant bits in the byte). Permitted length : 1 byte.

DG: digit (the four least significant bits in the byte). Permitted length : 1 byte.

FP: floating-point (single or double precision, i.e. 4 or 8 bytes respectively)

C: user defined collating sequence (see ALTSEQ command)

note 5

The order in which the logical operations on conditions are carried out is indicated by brackets; if there are none, operations are performed from left to right as follows:

```
A OR B AND C
```

will be carried out as

```
(A OR B) AND C
```

note 6

Analysis of the record-type is carried out in the same order as the record type definitions are provided; e.g. if two record-types are defined in an INPUT command.

```
INPUT : 2 comp-cond
        key-def
        field-def
      : 1 comp-cond
        key-def
```

The conditions that classify the record as type 2 (comp-cond) are first analysed; if these are not true (i.e. the record does not belong to type 2) the program analyses the conditions that classify the record as type 1. If this is not true either, the record will be omitted.

note 7

The logical operators (AND,OR) in the complex condition must be preceded and followed by a blank.

note 8

The number of conditions that make up the complex condition must not be over 128. There is also a logical limit related to efficiency considerations.

The large number of conditions may make it necessary to continue on the next line. In this case the character to be used is '&', which, when used in a condition, has some peculiarities, as :

- does not have to respect the integrity of the conditions and parameters (it can divide a condition that is in brackets) but, as said, cannot be used to continue a keyword (e.g. INP & UT is illegal)

- the continuation line must not begin with leading zeroes.

note 9

SORT uses the minimum number of bytes necessary to store the 'binary integer' type literals. For instance a literal that is indicated by the user as

'123' B

is stored by SORT in 1 byte.
If on the other hand the user wants the literal to fill a different number of bytes he will have to write it as preceded by leading zeroes.

To establish the number of zeroes to be inserted, the user must search in the table below the number of bytes he wants to fill and, corresponding to this, the minimum representable number formed only by 9s.
The number of leading zeroes to be added to the literal is given by the difference in digits between this number and the literal itself.

minimum representable negative value	# bytes	maximum representable positive value
- 128	1	127
- 32 768	2	32 767
- 8 388 608	3	8 388 607
- 2 147 483 648	4	2 147 483 647
- 579 755 813 888	5	579 755 813 887
- 140 737 488 355 328	6	140 737 488 355 327
- 36 028 797 012 963 968	7	36 028 797 012 963 967

Tab. 1. 3 - Range of Integer Numbers Representable in n Bytes

If, for instance, the literal '278' B , which would normally occupy 2 bytes, is to occupy 3, it has to be written as

'00278' B

In fact 99999 is the minimum positive integer formed only by 9s which can be represented in 3 bytes : as it occupies 5 digits and the supplied literal only 3, the literal itself will be supplied to the SORT as preceded by two non-significant leading zeroes.

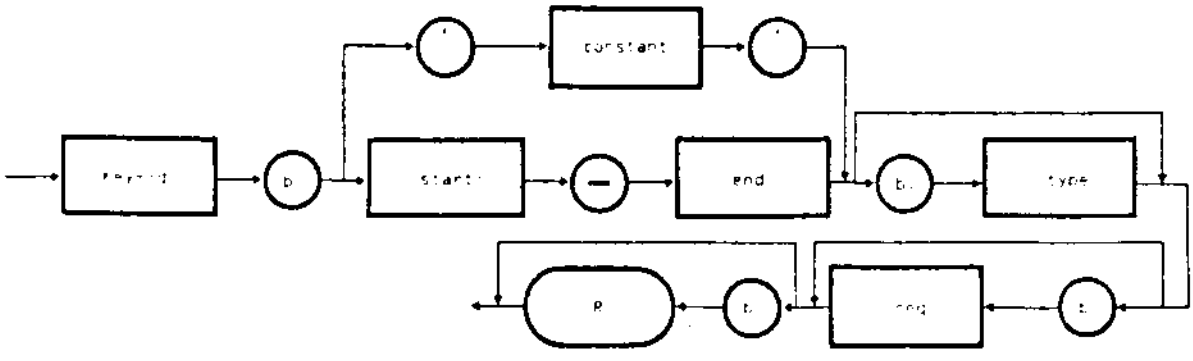
note 10 The FP literals are always stored by SORT in double precision (8 bytes).

Key-Def

Provides the utility with all information relative to the sorting keys, e.g. the record fields chosen by the user to control sorting.

At least one Key-Def must be included in each definition. The number of keys that can be defined in a description is limited by the fact that the total length must not exceed 256 characters. (See also note 4).

Each Key-Def must stay in its own line.



where:

Key-id

Identifies the key .

It is a capital K followed by a number (Kn). Number (n) determines the keys hierarchy: the order in which the keys are defined is not relevant. If the keys are defined in the following order:

K3,K1,K5

the relative hierarchy will be:

K1 primary key

K3 secondary key

K5 tertiary key

start-end — Defines initial (start) and final (end) positions of the key in the record, separated by minus (-).

The first character in the record is in position 1.

The keys must not overlap: the start position of a key must come after the end position of another one, unless it is the first one.

type — See the 'type' attribute values in parameter 'Rec-Type'.

The default value is A (ASCII).

seq — The sorting sequence for Kn key . It can be ascending (A) or descending (D).

The default value is A.

There can be ascending and descending keys in the same record.

R — Indicates reverse comparison, that is, for a character string key (whose collating sequence is ASCII, EBCDIC or user defined C) comparison is to be carried out from right to left.

It is indicated with a capital R added at the end of Key-Def.

It is an optional parameter, which can be added (not substituted) to 'seq'.

constant — A constant to be used as a key can be defined (forced) in the input record.
The notation will be

Kn 'constant'

that is the key identifier (key-id) followed by a blank and the constant in inverted commas.

The position of this extra key as defined in the record-type description can be changed in the OUTPUT command.

note 11 — The keys defined for different record-types must correspond in :

- type
- length and
- hierarchy level

K1 in record-type 1 must be a field of the same type and length as the K1 in record-type 2: they do not have to be in the same position in the record.

The same number of keys must be defined for each record-type.

Field Definition

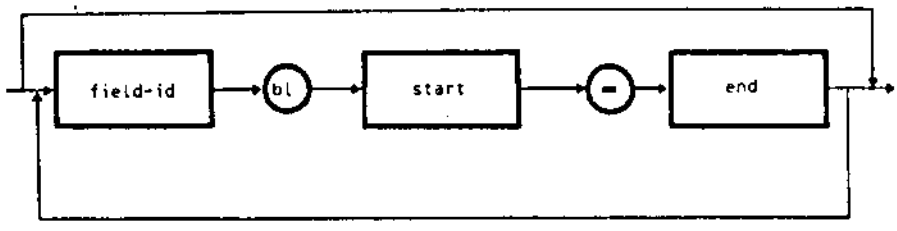
The input record fields to be included in sorting can be defined :

- explicitly, by using parameters field-id, start-end (Field-Def)
- implicitly, by using ALL parameter (Imp-Def).

All the fields that are not defined are omitted.

Each field definition must occupy its own line.

Field-Def

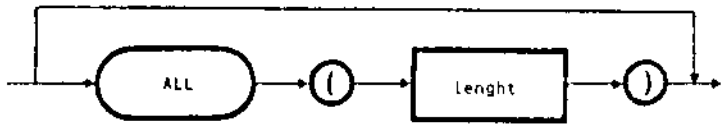


where:

field-id — Identifies the field. It is a capital F followed by a number (Fn).

start-end — Defines the first position (start) and the last one (end) of the field in the record, divided by a minus (-).

As for keys, overlap is not allowed.



where:

ALL(length) Defines the output record length and, implicitly, its layout. It is composed of ALL (key-word) followed by a number in brackets (length).

If present, it must be defined only once in the command.

With this option all the fields that have not been defined as keys, summary or fields are included in SORT, for a length equal to the number in brackets. This must be less than or equal to the length of the input record, and is the same as 'include n characters in SORT, starting from position 1 in the record'.

This parameter simply defines the output record length, and does not delimit the fields which may be referred to for record recognition.

The use of this option for fields definition implies that the output record layout will be exactly the same as for the input : therefore no output layout reformatting is allowed.

If in INPUT command with ALL option a constant is forced as a key (see 'constant' parameter in Key-Def sub-paragraph) the utility will put it at the head of the output record.

note 12

The number of fields that can be defined implicitly using the ALL option is equal to the number of contiguous groups of bytes between 1 and length which are not defined explicitly as keys, constants or fields.
So if the record type block is:

```
: 1  1='GROUP'
K1   1-15
F1   16-20
ALL(40)
```

The number of fields defined by ALL is 1 (contiguous group of bytes between 21 and 40).

If the record definiton is:

```
: 1   1='GROUP'
K1  10-15
F1  20-30
ALL(40)
```

Three fields are defined implicitly by ALL (the groups of bytes between 1 and 9, 16 and 19 and 31 and 40).
This must be borne in mind when calculating the number of fields that have been defined; that cannot be over 128 as previously mentioned (see note 2).

note 13

The FP, ZO, DG type keys are equivalent to a key plus a field, as far as the limits in key/fields definition are concerned (see note 2).

example 1

```
INPUT :  1 20='IMP'
     K1   10-14 E
     ALL(190)
     : 2  12=6(2) PD
     K1 15-19 E
     F1 1-9
     F2 10-14
     F3 20-190
```

Two record-types are defined for a 190 character record:
Type 1 can be seen to have constant 'IMP' starting from position 20.
Type 2 must have the packed field beginning in position 12 equal to the packed field which begins in position 6. The comparison is made for a 2

character length.

Any other records that do not belong to types 1 or 2 are omitted.

The sort key is an EBCDIC field 5 characters long: in record-type 1 it starts at position 10, while in record-type 2 it starts at position 15.
Sorting is done in ascending order.

example 2

```
INPUT 1 180=45(6) AND 1='A'
      K1 10-14 E R
      ALL(200)
```

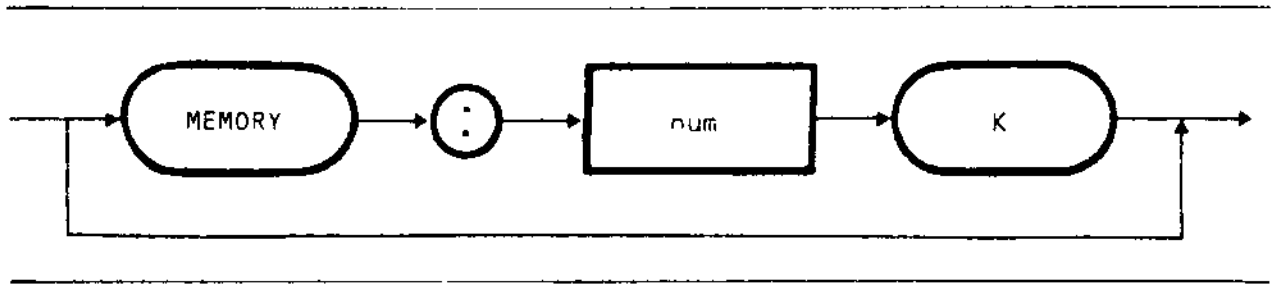
Only one record type is defined : every record which does not correspond to the defined condition is omitted.
The key is an EBCDIC string of 5 characters, for which the reverse comparison is to be carried out; order is ascending (default). With this definition, keys :

'ABCDE' , 'AAAAE' , 'EDCBA'

will be sorted in this order:

EDCBA
AAAAE
ABCDE

Defines the number of Ks of storage made available to the SORT program as data space. An increase in data space improves SORT efficiency (MERGE is not affected).



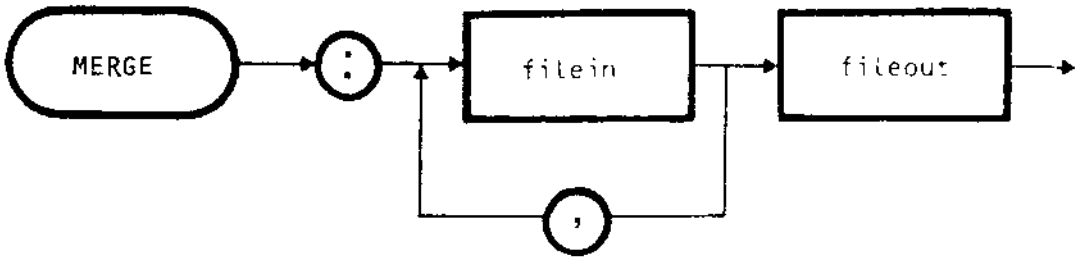
where;

num Is the number of storage Ks made available to SORT.

The minimum value is 10, but, in any case, if a lower amount is defined by the user, the utility takes the default value of 10.
The maximum value is 64 : any amount of space supplied above this limit will not be utilized.

K (Key-word) must follow 'num' without any intermediate blank.

Defines the input and output files to the MERGE program. This performs the merging of files already ordered according to the same keys.



where:

filein — One of the input files (max 4) which are divided by commas. The filein order in the list indicates priority among the files (which are all opened at the same time). This means that records in different files which have the same key will be output in the sequence given by the list.
The filein names must be all different.

fileout — The output file, which must be the last. By default, the last file in the list is assumed to be the output file : if not, it is used as such by the program , and its contents will be lost.
It cannot be one of fileins.

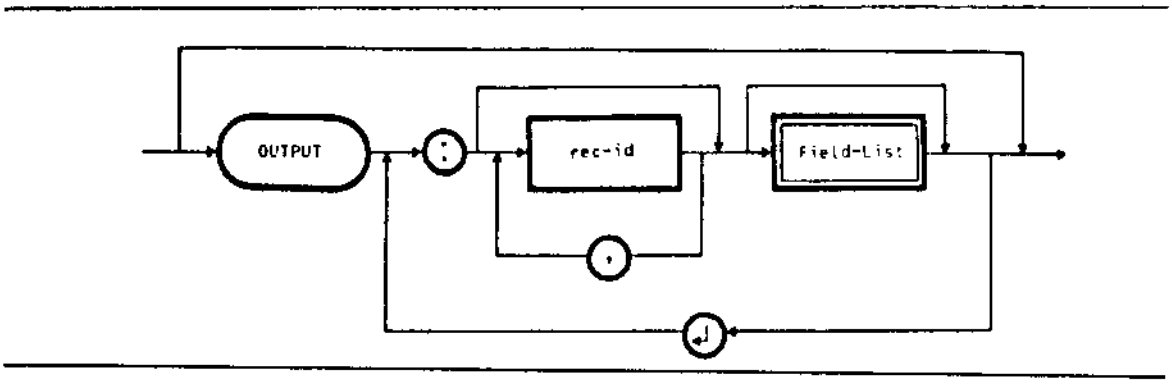
note 1 — The MERGE algorithm could be preferred to SORT, when possible, because its execution time is normally less than that of the SORT, under the same conditions. Besides this, in MERGE the order of equal keyed input records is preserved in output. Therefore it is worth exactly defining the algorithm to be used, if SORT or MERGE.
Care must be taken to have the same algorithm defined in the interactive phase (SORT or MERGE command) and in the execution phase (SORTRUN command) either explicitly or implicitly.

For each record-type that is defined in INPUT, OUTPUT permits determination of a new layout by linking in the required way the fields and keys that were defined in the relative record-type block; if necessary these are integrated with constants.

If reformatting of layouts is not required the command can be omitted.

The omission of this command has different meanings wether parameter ALL was used or not in INPUT command:

- if ALL was used in INPUT, omitting OUTPUT means that output layout must be the same as the input record;
- if ALL was not used in INPUT, omitting OUTPUT means that the output layout must be exactly the one described in INPUT command .
  This can be different from the input record layout, as fields may have been inserted or deleted by means of INPUT command features.



where:

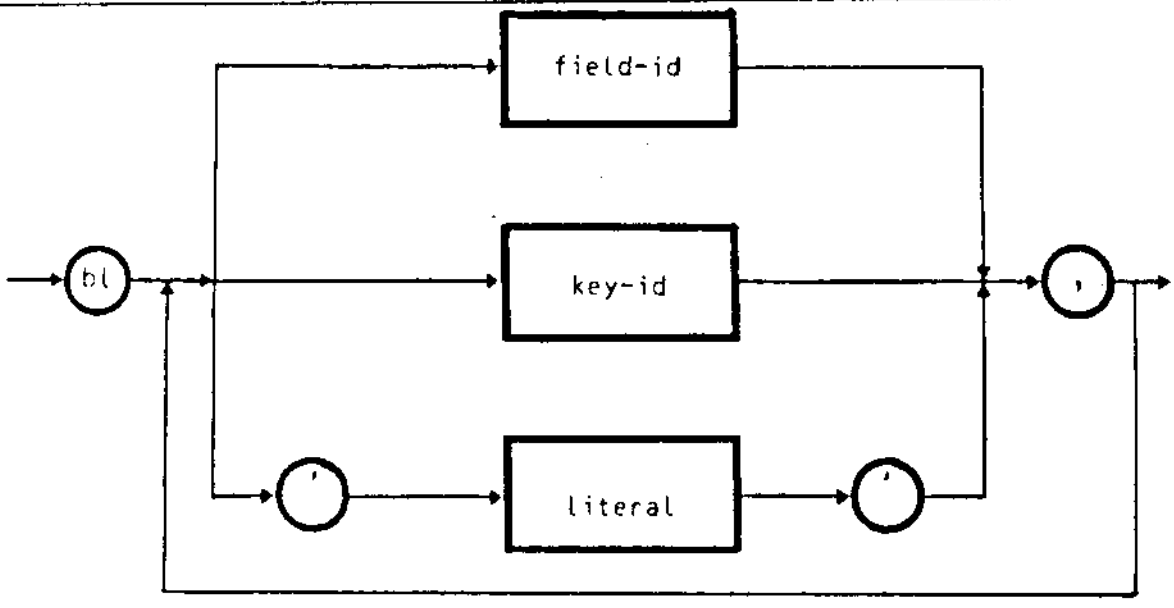
rec-id Is one of the record-types defined in INPUT. A number of rec-ids form the Number-List.
Not all the record-types described in INPUT have corresponding OUTPUT rec-ids. If so, the record will be written in output without being reformatted.
However, any rec-id that appears in this output definition must have been defined in INPUT.

If this parameter is omitted, every Rec-Type defined in INPUT command which is not included in any Number-List will have in output the layout defined in Field-List.

Field-List Is the output layout, defined for each record-type that is to be modified (rec-id).
If the record is to be left in its original format, Field-List can be omitted.

A Field-List not preceded by at least a rec-id must be the last one in the OUTPUT command.

Field-List



where:

field-id is a field identifier defined in INPUT within the rec-id description.

key-id is a key identifier defined in INPUT within the rec-id description.

literal is a constant in inverted commas that is forced into the output layout.

note 1

The above parameters can be mixed; the separating character is a comma, with or without blanks.

note 2

If several rec-ids are defined in OUTPUT the relative layouts must be of the same total length. If this is not done the utility aligns the shorter layouts to the longer ones by adding blanks at the end.
This is considered as an extra field, which is reckoned in Language Analyzer internal tables. Therefore the relative limits have to be taken into account : see command INPUT, note 2, and literal definition in Rec-Type sub-paragraph.

example 1

```
OUTPUT : 1
       : 2 F1,K1,F2,F3,'#'
```

In this example the layout that was defined in example 1 in the INPUT command is reformatted by aligning type 2 records to the record-type 1 format. Constant # is added in queue, so the OUTPUT record will be 191 characters long.
As far as the layout is concerned, record-type 1 is unchanged, but it is lengthened by a blank at the end so as to align it to record-type 2 length.

example 2

```
OUTPUT: 1, 3  F1, K1, F2
```

According to this notation, record types 1 and 3 wil have in output the described layout. If record type 2 was defined in INPUT, it will keep its orignal layout.

example 3

```
OUTPUT: F1, K1, F2, F3
```

In this example every Rec-Type described in INPUT will have in output this layout.

example 4

```
OUTPUT:
```

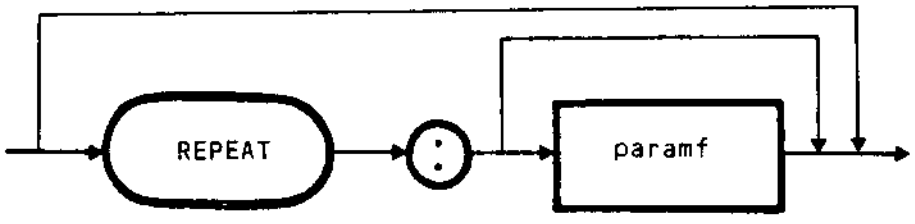
With this notation every Rec-Type defined in INPUT will keep in output:

- the original layout of the input record, if parameter ALL was used in INPUT.
- the layout defined in INPUT command, if parameter ALL was not used in that command.

Gives a name to the parameter file which is created by the utility using the command block for the current execution.

With this name the same execution can be run again later via the SORTRUN command, using the same input, output and parameters.

It must immediately precede END command in the command-block.



Where:

paramf — is the name given to the parameter file generated for the current execution.
If it is omitted, the parameter file receives SORTPARAM as default name.

If the whole command is omitted the parameter file will be deleted at the end of execution, and no re-execution of SORT/MERGE will be possible.

note 1 — The name paramf (or SORTPARAM) must not be already existent (that is, known to the File System).

example 1

```
SORT : file1, file2, file3

INPUT : 1
        K1 1-5 UD
        ALL(180)
REPEAT : paramf
END
```

In this example file1 and file2 are sorted by the

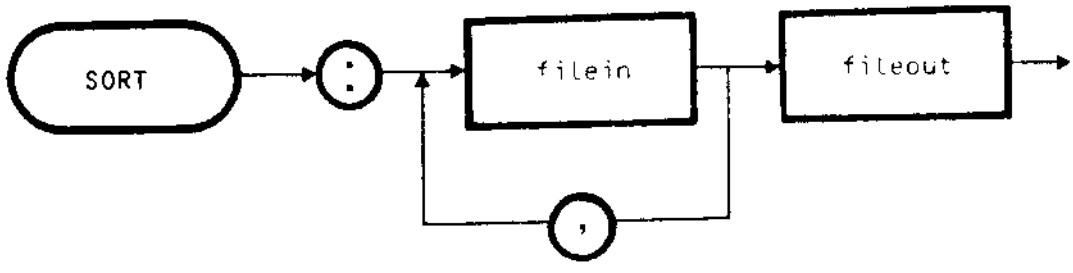
first 5 characters in the record (that are in unpacked decimal format) in ascending order and written on file3 without modifications.

A subsequent command

```
SORTRUN paramf
```

will cause execution of exactly the same SORT program : the input and output files and the parameters will be the same.

Defines to the SORT program the input and output files.
It must be the first element in the command-block.
It is a COMPULSORY command.



where:

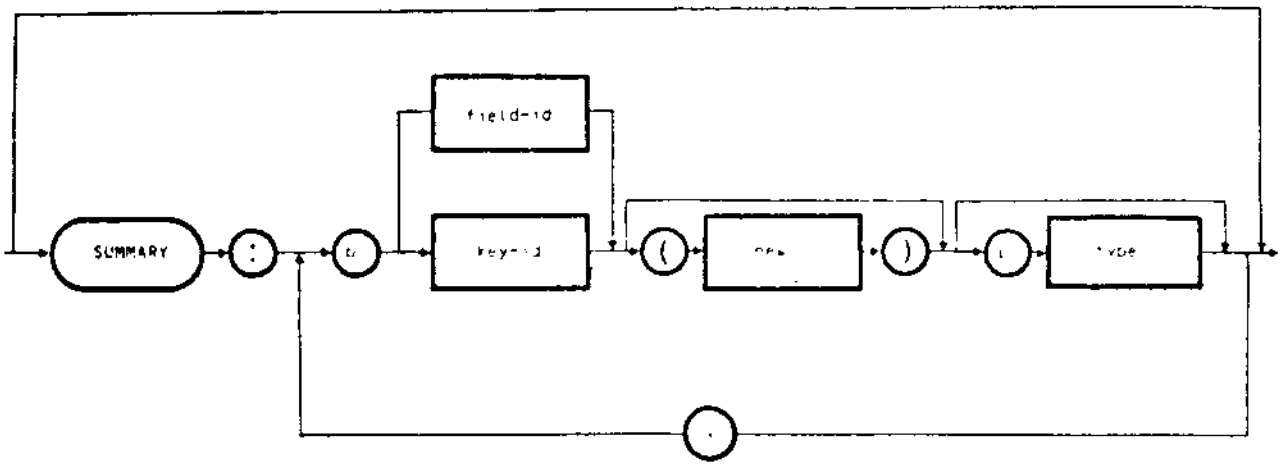
filein is the name of one of the input files (max 16) which are divided by commas.
The order of fileins in the list determines the opening order for the files (which are opened one at a time).
The names of the files must be different.
The type of files must be positional.

fileout is the name of the output file.
It must be the last one. By default, the last file in the list is assumed to be the output file : if not, the program uses it as such, and its contents will be lost.
It must be different from every filein.
The file type is positional.

Defines the fields that must be summarised in records having the same key.

The summary fields can be defined as longer than the original one so as to avoid overflow.

If the OUTPUT command is defined in the command block, SUMMARY must follow it in the order.



Where:

field-id is a field identifier defined in INPUT command.

key-id is a key identifier defined in INPUT command.

newl is a number in brackets (max value 32) which defines the new summary field length.

type Can only be :

B binary integer (length from 1 to 8 bytes)

FP floating-point (single or double precision,

that is 4 or 8 bytes)

PD packed decimal (max. length 63 digits, i.e.32 bytes)

UD unpacked decimal (max. length 32 bytes).

The default value is UD.

note 1

SORT does not keep the same input order for groups of records that have the same key. So any record with that key can be the first one on which the others are totalised.
This means that the non-summary fields that appear in the output record can belong to any one of the records in the group. On the contrary, MERGE keeps the same order that was defined for files in the MERGE command.

note 2

The character used to divide the elements in the list of the summary fields is a comma.

note 3

If several record-types have been defined in INPUT, any summary fields defined must be present in every Rec-Type.

note 4

If several layouts have been defined in OUTPUT, care must be taken when using the SUMMARY command.

In fact a SUMMARY cannot be defined for each record type : as a result, records with the same key, but of different types will be synthesized into one.

This can be overcome by using the keys as in example 3 in chapter 3 (Partial Totalisations).

example 1

```
SORT : file1,file2

INPUT : 1 39='BB'
        K1  1-5
        F1 10-15
        F2 16-40

SUMMARY : F1(7) PD

END
```

In this example field F1 is defined as a summary field and lengthened so as to avoid overflow.

When there are several records which have the same

key, only the first one will be taken as a whole. Of the other records in the group only field F1 (characters 1-15) will be added to F1 in the first record, whilst the other field values will be lost.

As one can never know which of the group records will appear first, one cannot know either which non-summary fields will appear in the output record.

Note that columns 6-9 of input record do not appear in output, as not defined in INPUT.

2. SORT/MERGE EXECUTION

INTERACTIVE

SORT/MERGE is designed to be executed directly from the SHELL environment using the SHELL commands LANGUAGE and SORTRUN which invoke the two phases of the utility:

- LANGUAGE, activates the Interactive Phase (the Language Analyser).

- SORTRUN, activates the Execution Phase (the SORT or MERGE algorythm).

The operation of these commands is described on the following pages.

BATCH MODE

Alternatively, the utility may be activated from within a SHELL procedure, which may be executed in the Batch Environment.

Redirected Standard I/O

In this case, the standard input must be redirected from a file containing the command block required by LANGUAGE.

The standard output should also be redirected to a file which is to receive any messages generated by the utility, which would otherwise appear on the screen.

Completion Codes

The utility returns a code in the SHELL variable %STATUS to indicate the outcome of the sortrun. This code may be used by the activating procedure to determine what action should be taken next.

The completion codes and their meanings are as follows:

0	SORTRUN completed
1	memory requested not available
2	insufficient memory requested
3	file has not been sorted
4	sequence error
5	illegal file type
6	invalid pathname

7	filename not found
8	file already exists
9	filenames the same
10	arithmetic overflow
11	parameter file incompatible with call
12	illegal input record length
13	input files have different record lengths
14	record count discrepancy
15	security violation
16	out of disk space
17	hardware error
18	device not ready
19	out of bounds
20	system error
21	timeout error
22	sort interrupted
23	program can't run

These codes have the same meanings as the standard SORT/MERGE error messages which are fully described in Appendix B.

Example

An example of how SORT/MERGE might be executed from a SHELL procedure is as follows:

```
LANGUAGE < COMMANDFILE > MSGFILE ;
SORTRUN                > MSGFILE
```

where:

COMMANDFILE is a bytestream file which contains the command block required by LANGUAGE.

MSGFILE is a bytestream file which is to receive the messages and screen display commands generated by the interactive phase.

Note

As the standard I/O cannot be redirected for the optional parameter file and "OUT=" parameter needed by SORTRUN, the utility can only be executed from within a SHELL procedure if the SORTRUN command has no input parameters.

Calls the Language Analyzer for executing the utility Interactive Phase.

The called Language Analyzer displays the SORT (or MERGE) prompt on the application window of the screen:

SORT* >

The operator then keys in the command block. At the end of each command, parameter or line ('carriage return' key) Analyzer points out possible formal or syntactical mistakes made by the operator. He is generally given the choice between interrupting the run or correcting at once the mistake, on the basis of the message given by the Analyzer.

At the end of an error free command block, the operator can request execution of the algorithm defined there (SORT or MERGE) by means of the Shell command SORTRUN.

note 1

Care must be taken when using REPEAT command (in the interactive phase) and SORTRUN (in the execution phase) without explicitly mentioning 'paramf' parameter : this could bring to an undesired execution of an old program.

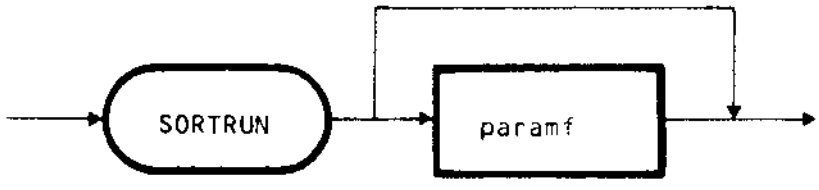
note 2

If 'paramf' pathname is absolute the file is searched starting from the root directory ; if relative, the file is searched starting from the current working directory. If this is not the same under which the file is defined, an error message will be received, namely the SORT/MERGE error message 13.

Causes execution of the SORT or MERGE algorithm, according to the command specified in the command-block during the Interactive Phase.

Execution can be:

- the first one
- one of many repeated ones, if, in a previous execution, the parameter file created by the Language Analyzer was stored by means of the REPEAT command.



where:

paramf is the name of the parameter file stored using a REPEAT command in a previous execution: its presence means that a re-execution of the previous SORT is desired. If omitted, the meaning may be :

- this is the first execution of the utility
- this is a re-execution of a previously run SORT/MERGE whose command block contained the REPEAT command without definition of paramf. In this case the parameter file was given the the default name SORTPARAM which can be omitted here.

3. EXAMPLES

This chapter gives a few examples which illustrate the use of the utility in its entirety: it does not simply refer to the individual functions or to the individual commands, as shown in the previous chapters.

A straightforward use of SORT

The example illustrates how to sort a file formed by records of the same type, without carrying out any record or field exclusion, or any reformatting of output layout. The elements to be defined are: input and output files, the sort key (default options: ASCII and ascending) and the number of bytes of the record to be sorted.

```
SORT  :    inpfile,outfile
INPUT      :   1
           K1  10-15
           ALL(200)
           END
```

This is an example of a very simple command-block.

Use of an additional constant as a key

The input file is made up of two record-types which must be sorted on the same key (an EBCDIC field of 9 characters) in descending key order, but in such a way that all type 2 records precede those of type 1. This can be achieved by adding a primary key of a low binary value to type 2 records, (e.g. a blank) and one of a higher binary value to type 1 records (e.g. a 0).
The two formats must also be made equal, according to the type 1 record layout.

```
SORT :     inpfile,outfile
INPUT      :1  15='1000'
           K1 '0' E
           K2 10-18 E D
           ALL(100)
           :2
           K1 ' ' E
           K2 1-9 E D
           F1 10-18
```

```
                F2 19-100
OUTPUT    : 2 K1, F1, K2, F2
            END
```

The two added primary keys will split the output file into two groups, the first of which will be made of all the records of type 2, while the second will be made of all the records of type 1.
Inside the two groups, records will be ordered by the secondary key.

Use of the ALL option, together with the OUTPUT command, does not contradict what has been said in the description of the ALL parameter itself. The type 1 record in fact, is not reformatted (except for the addition of K1 at the head of the record, which is determined by the use of ALL). Only the type 2 record is reformatted and the definition of the layout is only significant for this record (in which the ALL option is not used).

Partial totalizations

In a SUMMARY command, it is not possible to differentiate the totalization fields by record-type.
However, if several record-types with different layouts have been defined for output (explicitly or implicitly) , and if separate accumulations are desired for each of them, the same action has to be taken as in the previous example.

A different constant will be added to each record type, to be used as a secondary key. This means that records with the same primary keys belonging to different types, which would normally have been accumulated, will not, as they are differentiated by the added secondary key.
This allows a 'de facto' partial totalization for each record type.
In fact, for each group of records with the same primary key, two records instead of one will be output and summarized, if the group contains records of the two different types.

```
SORT :  inpfile,outfile
INPUT  : 1  99=107(5) UD
         K1  1-6
         F1  7-12
         F2  13-100
         K2  '1'
         :2
         K1  7-12
         F1  1-6
         F2  13-100
         K2  '2'
```

```
SUMMARY: F1(8) UD
END
```

As an OUTPUT command has not been defined, each of the 2 record types will be output with the format described in INPUT, and with the added key in the position also defined for each of them in INPUT. This is also an example of implicit definition of the output layout.

Diversifying in output records of the same format

This is made possible by defining 2 record types and by adding a constant, different for each type, to be used as a secondary key. This added key must not appear in output.

```
SORT   :  inpfile, outfile
INPUT  : 1  50='000126' B
         F1 1-25
         K1 26-30
         F2 31-49
         K2 '*'
         F3 50-100
         :2
         F1 1-25
         K1 26-30
         F2 31-49
         K2 '+'
         F3 50-100
OUTPUT   :1 F1, K1, F2, F3
         :2 K1, F1, F2, F3
         END
```

Alternative collating sequence

In this example, the command block is given for the execution of the SORT program described in example 1 of the ALTSEQ command (chapter 1).

Execution of the SORT function will be preceded by the creation of 'filen', containing the alternative collating sequence as a one record file of 256 bytes in byte-stream format. After this, SORT can be executed.

```
SORT    : inpfile,outfile
INPUT   : 1
          K1 21-22 C
          K2 26-30 E
          ALL(200)
          ALTSEQ filen
          END
```

The primary key, which has been sorted according to the alternative character set defined in filen, will ensure that the records containing asterisks in that key precede those with a primary key set at blank, as requested.
Within the two groups, records will be ordered according to the secondary key, in the normal EBCDIC sequence.

Complex condition

The utility can solve extremely complex conditions (comprising up to 128 sub-conditions) to define a Rec-type. An example of this could be:

```
SORT :  inpfile,outfile
INPUT : 1 (11=20(5) PD AND 21=&
'EVENT') OR (11=41(5) &
PD AND 150='00123' B)
        K1 95-100 UD
        ALL(200)
END
```

Only the records satisfying the defined complex condition are extracted and sorted; all the others are rejected.

The example also illustrates the use of the character 'line continuation' &, as well as the user's definition of the length of the binary-type literal. This determines the length of the comparison between the literal itself and the field in position 150: it is 3 characters instead of the single character which would have been assumed as default by SORT.

Merging four files

The example illustrates how to merge into one, four files which have been previously sorted on the same key, and reformat them in output, adding the 'ANAG' constant to the record tail. Field F1 must also be totalized so as to release a single record for every group of records with the same key.

```
MERGE :   file1,file2,file3,&
          file4,fileout
INPUT     :1
          K1   20-27 PD
          F1   1-9
          K2   10-19 E
          F2   28-90
```

```
OUTPUT    : F1, K1, K2, F2, 'ANAG'
SUMMARY   : F1(12) UD
END
```

By this notation, the output layout will be changed from the INPUT defined format.
This example is to show how every feature of the parametric commands is also valid for MERGE, with the sole exception of MEMORY.

4. ERROR MESSAGES

The error messages produced by the utility are two types, related to:

- Language errors: the message is displayed by the Language Analyzer during the interactive phase, at the end of a keyed in command or line ("carriage return" key: ↵)

- SORT/MERGE errors: the message is displayed by the algorithm concerned during the execution phase. At this point errors made in the preceding phase can also be displayed, which can only be detected at run time (e.g. error 4 in the "SORT/MERGE errors" list) together with File System passed on errors.

ERROR INTERACTIVE HANDLING

Every message, visualised in the screen application window, is followed by the action taken by the utility, which can be:

- SORT ABORTS
- SORT CONTINUES
- ABORT? (y/n)

In the last case, if the operator chooses to go on (keying 'n') the utility again signals the action it takes, or the action suggested to the operator. The possible cases are:

- SORT ACCEPTS THE SECOND COMMAND; if two equal commands are keyed in, the utility assumes that the first is wrong. If this is not true, the operator can key in the right command again.
- SORT IGNORES THE LINE: a misplaced line is not taken into account by the utility
- REPEAT COMMAND: the operator must rekey the whole command
- REPEAT THE LINE: the operator only has to re-key the last keyed line
- REPEAT LAST DESCRIPTION: the operator must re-key the last 'Rec-Type description' (see note-1 in INPUT command)
- INSERT INPUT COMMAND: the operator must key in the mandatory INPUT command.

At any time the operator can re-key a command which overrides the first one. The utility will advise that the second command is accepted.

NOTES

1. When an error message is followed by 'SORT ABORTS' the utility, before aborting, gives control to SHELL, and the following remain displayed on the screen

- the SORT error message

- the Shell prompt, i.e.>

2. When an error message is followed by 'SORT CONTINUES', it is a simple warning, that is, correction of the error is not mandatory.

3. If more than one error is made in a single line, the Language Analyzer only signals the first one.

4. If the action suggested to the operator 'REPEAT THE LINE' refers to a continuation line, both this and the preceding one (ending with '&') must be repeated.

APPENDIX A. LANGUAGE ERROR MESSAGES

1. <K> EXPECTED	In the MEMORY command there is no K after the number. SORT continues.
2. <:> EXPECTED	An operating code is not followed by the colon. SORT continues.
3. <'> EXPECTED	A literal is not enclosed in inverted commas. ABORT? (y/n) /y/ SORT aborts /n/ Repeat the line

4. LITERAL LENGTH OUT OF RANGE

The literal length limits ($0 < L < 257$) have not been respected.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the line

5. ILLEGAL SYMBOL Both the action undertaken by SORT and the action suggested to the user, vary, depending on the command which has provoked the message.

6. ILLEGAL STATEMENT

A terminal symbol or a parameter attribute (both positional) are not in the correct position, and from the context it is not possible to extract any elements for a more precise interpretation.

SORT continues.

7. <-> EXPECTED

In a parameter with start-end positions (e.g. Key-Def and Field-Def in INPUT command) the '-' character is missing.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the line

8. ILLEGAL CHARACTER IN NUMERIC LITERAL

A literal defined as a numeric one (B, ZO, DG, FP, PD, UD) contains illegal characters.

ABORT? (y/n)

/y/ Sort aborts

/n/ Repeat the line

9. OUTPUT OVERRIDES PREVIOUS <ALL>

Following an INPUT command containing the ALL parameter, an OUTPUT command has been keyed in for the same Rec-Type/s (incompatible).

SORT continues

10. <(> OR NUMBER EXPECTED

A condition or a comparison length in INPUT command or a 'newl' in SUMMARY command is formally incorrect.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the line

11. TYPE NOT NUMERIC

The only types allowed in the SUMMARY command are: PD, UD, FP, B.

ABORT ? (y/n)

/y/ SORT aborts

/n/ Repeat the line

12. DOUBLE DEFINITION

A command has been keyed in more than once.

ABORT? (y/n)

/y/ SORT aborts

/n/ SORT accepts the second command.

13. TOO MANY FILES

Maximum number of input files to MERGE are 4, to SORT 16.

SORT aborts

14. FILE NAME MISSING

Undefined file name in the SORT/MERGE command.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the command

15. LENGTH OF ZO/DG TYPE NOT 1

The ZO, DG fields must be 1 character in length.

ABORT ? (y/n)

/y/ SORT aborts

/n/ Repeat the line

16. LENGTH OF BINARY TYPE NOT LESS THAN 9

Maximum length of B fields is 8 bytes.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the line

17. LENGTH OF FLOATING TYPE NOT 4 OR 8

Length of the FP fields can only be of 4 or 8 bytes

ABORT ? (y/n)

/y/ SORT aborts

/n/ Repeat the line

18. RELATIONAL OPERATOR EXPECTED

In a condition (INPUT command) a relational operator (> , < , = etc) is missing.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the command

19. NUMBER EXPECTED

A number is missing : for instance, a length definition in a totalisation field of SUMMARY command, or an 'end' position in start-end parameter of Field-Def or Key-Def (INPUT command).

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the command

20. UNBALANCED PARENTHESIS

Unbalanced parenthesis in a condition or in a length definition.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the line

21. OPERAND EXPECTED

Operand missing in a logical condition (INPUT command).

ABORT? (y/n)

/y/ SORT aborts.

/n/ Repeat the line

22. LOGICAL OPERATOR EXPECTED

Logical operator (AND,OR) missing in a condition (INPUT command).

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the line

23. LOGICAL EXPRESSION OPTIONAL FOR LAST DESCRIPTION ONLY

A Rec-Type defined in the INPUT command does not contain any condition and it is not the last one.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the whole INPUT command

24. NO INPUT COMMAND

The INPUT command does not immediately follow the SORT/MERGE command.

ABORT? (y/n)

/y/ SORT aborts

/n/ Insert INPUT command.

25. OVERLAP BETWEEN KEY AND/OR FIELD

In INPUT command key and/or field definitions overlap.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the entire record-type description

26. RECORD LENGTH INCONSISTENT WITH ALL STATEMENT

A Key-id or a field-id has been defined outside the limits established by the ALL(length) parameter, in INPUT command.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the line

27. NO KEY

At least one key must be defined for each Rec-Type.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the description if it is the last one keyed in; otherwise, repeat the whole INPUT command.

28. KEY NUMBER MISMATCH

In INPUT command the number of keys defined in a record-type description does not match that of the preceding description.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the record-type description, if it is the last one keyed in; otherwise, repeat the whole INPUT command.

29. TOTAL KEY LENGTH MISMATCH

The total length of the keys defined in a record-type is different from that of the previous record-type (INPUT command).

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the incorrect description if it is the last one keyed in; otherwise, repeat the whole INPUT command.

30. RECORD LENGTH OUT OF RANGE

Record length defined in INPUT command is not included between 1 and 4095.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the line

31. ILLEGAL LENGTH

A length is illegal due to logical reasons (e.g. a zero length) or because stated limits have not been respected (e.g. 'newl' limits in SUMMARY command, or limits for FP fields).

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the line

32. INVALID KEY OR FIELD

A Key-Def or a Field-Def in INPUT command is incorrect due to various reasons: e.g. the keyword for a key (K) or a field (F) is not followed by a number without embedded blanks; or start position = 0, etc.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the line

---

33. <(> EXPECTED

A summary field in SUMMARY command or ALL parameter in INPUT command is not followed by a left parenthesis.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the line

---

34. <)> EXPECTED

A length indication (e.g.in ALL parameter of INPUT command or 'newl' parameter in SUMMARY command) is followed by an indication of length, not ending with the right parenthesis.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the line

---

35. KEY MISMATCH

The Keys defined for various record-types in INPUT command do not match by either number, type, length or identifier (key-id).

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the description if the error is in the last one keyed in ; otherwise repeat the INPUT command.

---

36. NUMBER TYPE OUT OF RANGE

The record type identifier number (rec-id) does not stay between the stated limits (0 < N < 17) (INPUT or OUTPUT command).

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the command.

---

37. OUTPUT MUST PRECEDE SUMMARY

The SUMMARY command does not follow the prescribed order.

SORT continues.

---

38. FIELD OR KEY NOT FOUND IN INPUT STATEMENT

In the OUTPUT or SUMMARY command a field or a key is defined which has not been defined in INPUT.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the line

39. TYPE NOT DEFINED IN INPUT

In the OUTPUT command there is a record-type not defined in INPUT.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat last description

40. NUMBER LIST OPTIONAL FOR LAST DESCRIPTION ONLY

In the OUTPUT command, there is a Field-List not preceded by rec-id, which is not the last part of the command.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the whole command

41. SUMMARY LIST MISSING

Totalization fields missing in the SUMMARY command.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the command

42. TYPE MISMATCH

The totalization field type in SUMMARY command is not the same defined in INPUT for that field (which is a SORT Key).

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the incorrect command.

43. FIELD IN SUMMARY LIST NOT IN OUTPUT LIST

A SUMMARY field is not defined in OUTPUT command.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the incorrect command.

44. OUTPUT FILE EQUAL INPUT FILE

In MERGE or in SORT the output file is not different from the input file.

SORT aborts

45. ALTERNATE COLLATING SEQUENCE NOT SUPPLIED.

1) In INPUT, a C type Key has been defined, but the ALTSEQ command is missing.

SORT aborts.

2) The ALTSEQ command is not completed by the indication of 'filename'.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat the command.

46. ALTERNATE COLLATING SEQUENCE SIZE< >256.

The file containing the alternative collating sequence is not 256 bytes in length.

SORT aborts.

47. AMBIGUOUS COLLATING SEQUENCE

The characters which form the alternative collating sequence are not different one from the other.

SORT aborts.

48. NOT YET IMPLEMENTED

Command has not yet been implemented.

SORT aborts.

51. TOTAL KEY LENGTH > 256

The total length of the keys defined for a record type is > 256.

ABORT? (y/n)

/y/ SORT aborts

/n/ Repeat last description .

52. FILE SYSTEM ERROR

File System signals an error condition.

SORT aborts.

53. NO C TYPE

An ALTSEQ command has been provided, but no C type key has been defined in INPUT.

SORT continues.

54. TOO MANY CONDITIONS

The number of conditions forming a complex condition in INPUT command is greater than 128.

SORT aborts

55. TOO MANY LITERALS

The total length of all the literals defined in a command block is > 800 characters.

SORT aborts

56. TOO MANY FIELDS

The fields and/or keys defined in a record-type block are more than 128.

SORT aborts.

57. ILLEGAL PATH NAME

The file pathname does not follow the formal rules.

SORT aborts.

58. PARAMETER FILE ALREADY EXISTS

The name of the parameter file defined in REPEAT command (paramf or -implicit- SORTPARAM) is already existent.

SORT aborts.

APPENDIX B. SORT/MERGE MESSAGES

---

1. MEMORY REQUESTED NOT AVAILABLE, USED nn K

The storage requested for SORT by means of MEMORY command is not fully available, but a lesser extent is sufficient, and is used by SORT.

SORT continues.

---

2. INSUFFICIENT MEMORY, REQUIRED nn K

The storage requested for SORT by means of MEMORY command is partially available, but this amount is insufficient for SORT.

SORT aborts.

---

3. FILE NOT SORTED

A MERGE input file has not been previously sorted.

SORT/MERGE aborts.

---

4. SEQUENCE ERROR

An error has been detected in the sort operation.

SORT aborts.

5. ILLEGAL TYPE OF FILE

An input file is not of a positional type.

SORT aborts.

6. INVALID PATHNAME

The file name does not follow the File System rules.

SORT aborts.

7. FILE NAME NOT FOUND

One of the declared file names is unknown to the File System.

SORT aborts.

8. FILE ALREADY EXISTENT

The output file is already existent.

SORT aborts.

9. EQUAL FILENAMES

SORT : an input file is the same as the output file, even if they have different pathnames.

MERGE : two input files are the same.

SORT/MERGE aborts.

10. ARITHMETIC OVERFLOW

The 'newl' defined in the SUMMARY command is insufficient.

SORT continues.

11. PARAMETER FILE INCOMPATIBLE WITH CALL

The (SORT or MERGE) verb defined in the command block (interactive phase), is different from that defined in the SORTRUN command (paramf or SORTPARAM).

SORT aborts

12. ILLEGAL INPUT RECORD LENGTH

The record length defined in the INPUT command is different from that verified by the File System at run time.

SORT aborts

13. INPUT FILES WITH DIFFERENT RECORD LENGTH

The lengths controlled at run time by File System are not the same.

SORT aborts.

14. RECORD COUNT DISCREPANCY

An error has been detected in the sort operation.

SORT aborts.

15. SECURITY VIOLATION

Attempt to access a protected file.

SORT aborts.

16. OUT OF DISK SPACE

Insufficient space on disk for output file.

SORT aborts.

17. HW ERROR

Hardware malfunction.

SORT aborts.

18. DEVICE NOT READY

A device is not on.

SORT aborts.

19. OUT OF BOUNDS

This is a File-System message. See "Message Book", Part 10, error code 608.

SORT aborts.

20. SYSTEM ERROR

System Error.

SORT aborts.

21. TIMEOUT

This is a File System message, indicating an attempt to access a file with a timeout lock, where the timeout has expired.

SORT aborts.

22. SORT INTERRUPTED

1. The user has stopped the program using the SHELL command KILL.

2. An unknown software error has been detected.

SORT aborts.

23. PROGRAM CAN'T RUN

The overlays required by the SORT/MERGE utility have not been loaded into memory.

SORT aborts.