ccm-71-06

SORT
A File Utility Program for OS-3

February, 1972

OSU
COMPUTER CENTER
Oregon State University
Corvallis, Oregon 97331

SORT
A File Utility Program for OS-3

ccm-71-06

Computer Center
Oregon State University
Corvallis, Oregon 97331

February, 1972

TABLE OF CONTENTS

PREFACE

THE MANUAL

This manual is a technical description of how to use the OS-3 SORT program. This manual is complete and contains some sample problems.

THE PROGRAM

The OS-3 SORT program is a general-purpose sort-merge that allows the user to sort from one to thirty-two input files. It will handle fixed length, variable length, and blocked records. A number of options let the user choose how he wishes SORT to handle parity errors and block length errors. The program may be called as a subprogram and several special exits are provided in order to link it to user-supplied routines for special testing or modification of the records.

SORT FOR OS-3

1. USAGE

1.1 CALLING SEQUENCE

The calling sequence is the control statement SORT,I=a,O=b where a and b stand for logical unit numbers or file names. The I or input parameter specifies the logical unit or file that contains the parameter records for the SORT program. If the I is omitted, the SORT program will read parameter records from logical unit 60. The O or output parameter specifies the logical unit or file that the SORT program will use when writing error messages and record counts. If the O is omitted, the SORT program will write messages on logical unit 61. See section 7 for the calling sequence from a FORTRAN program.

One example of a typical sort job would be as follows:

```
7
8JOB,<number>,<user code>

7
8EQUIP,1=DATA

7
8EQUIP,2=FILE

7
8SAVE,2=SORTDATA

7
8SORT
 R20
 B50
 I1
 O2
 KEY2,2,2
 KEY10,1
 KEY4,6
 END
7
8LOGOFF
```

For other examples and formats, refer to section 8, Sample Problems.

1.2 SORT PARAMETER RECORDS

The parameter records are processed from left to right and have no format except that information in character positions beyond column 72 is ignored. The SORT program first searches the parameter input for a letter. Once a letter has been found, the SORT program will search the parameter input, ignoring all letters, for a positive integer. The SORT program will then process the parameter and start searching for a new letter. This process continues until the letter E is found, or until a file mark or end-of-data condition occurs. The comma, blank, and equals are always ignored.

EXAMPLE: The following parameters will process identically:

```
I5
I=5
INPUT=5
INPUT CAN BE FOUND ON LOGICAL UNIT 5
I 5
```

The parameter codes are:

R - Record size in words

B - Blocking factor (Records/Block)

I - Input file

O - Output file

K - Key field

T - Table for non-standard collating sequence

P - Parity error options

L - Length error options

E - End of SORT parameters

NOTE: The order of presentation of parameters is critical. The R, P, L, and B parameters apply only to the previous I or O parameter unless they occur before the first I or O parameter. If an R, P, L, or B parameter should occur before the first occurrence of an I or O parameter, the R, P, L, or

B parameter will be treated as if it is specifying a default option. It will then apply to all following I and O parameters which have no R, P, L, or B parameter.

EXAMPLE: The following parameter sequences will process identically:

```
I=5 R=20 B=0 I=6 R=20 B=5 O=7 R=20 B=5
R=20 I=5 B=0 I=6 B=5 O=7 B=5
R=20 B=5 I=5 B=0 I=6 O=7
```

The following paragraphs describe the various SORT parameters. The key letter is underlined. Only the key letter is necessary when making parameter cards. All letters between the key letter and the parameter value (positive integer) will be ignored. In other words, the user need not worry about whether he writes I=5, or I 5, INPUT5, or I5; the program will accept free form input.

1.2.1 END

END should be the last parameter. The END parameter will cause the SORT program to stop processing parameters and start reading from input units.

1.2.2 INPUT

A logical unit number should follow the INPUT parameter. Up to thirty-two input units may be specified by repeated use of this parameter. At least one input unit must be specified. The input units are examined, one at a time, from first to last. The input units are not rewound before or after reading. An end-of-file or end-of-data condition will stop the SORT program from reading from an input unit, and will cause the SORT program to start reading from the next input unit. The SORT program will stop reading when the last input unit is at end-of-data or end-of-file.

```
EXAMPLES:  INPUT=4
           I=65,I=66,I=67
           I3
           I 10
```

1.2.3 OUTPUT

Following the OUTPUT parameter should be the logical unit number on which the sorted output will be written. The output file is not positioned before writing, but following completion of writing the output, a file mark is written on the output unit. If the output unit is a file or magnetic tape, the unit is positioned to the location before the file mark. More than one output file may be specified; however, the output is normally written on the first output unit. For using more than one output file see section 7.1.2. One output unit is required.

```
EXAMPLES:  OUTPUT=61
           O=73
           OUT,3
           O 75
```

1.2.4 KEY

The sort key specifies the control field that will be used in sorting. This parameter may be used repeatedly to specify more than one SORT control field. The first key is the major key, or major control break. The last key is the least significant control field. Following the KEY parameter there are three numbers: the first column of the field, the number of columns in the field to be sorted, and the type of collation desired. (A maximum number of 10,000 columns may be specified in the keys.)

The first column number must be greater than zero and may be a column that is outside of the record. (If 80-column records are being sorted, a user may specify a key in column 100.) All of the input records have assumed blanks (if BCD) or assumed zeros (if binary) in columns greater than the last column of the record.

The number of columns must be greater than zero and is assumed to be one if omitted.

The collating sequence is specified by a number from zero to seven. If omitted, it is assumed to be zero, which is standard BCD collating sequence in ascending order.

If more than one record should contain identical keys, the SORT program will select the record that was read first as the "winner". That is, records with identical keys will maintain their order.

The meaning of the collating sequence numbers is the following:

0 - standard BCD collating sequence - ascending order
1 - standard BCD collating sequence - descending order
2 - binary collating sequence - ascending order
3 - binary collating sequence - descending order
4 - signed binary collating sequence - ascending order
5 - signed binary collating sequence - descending order
6 - supplied with the 1st Table card
7 - supplied with the 2nd Table card

Standard BCD collating sequence in Ascending Order

	60	<	32	$	53	(	74	4	04	C	23	K	42	S	62
:	12	.	33	*	54	→	75	5	05	D	24	L	43	T	63
=	13	)	34	↑	55	=	76	6	06	E	25	M	44	U	64
≠	14	≥	35	↓	56	∧	77	7	07	F	26	N	45	V	65
≤	15	¬	36	>	57	0	00	8	10	G	27	O	46	W	66
%	16	;	37	/	61	1	01	9	11	H	30	P	47	X	67
[	17	-	40	]	72	2	02	A	21	I	31	Q	50	Y	70
+	20	∨	52	,	73	3	03	B	22	J	41	R	51	Z	71

The binary and signed binary collating sequences use the CDC 3300 Internal BCD character codes to determine the collating sequence. However, the signed binary collating sequence uses the most significant bit of the field as a sign bit.

```
EXAMPLES:  KEY 3,3,5
           KEY 1,29
           KEY 40 20 6   KEY=1,10,0   KEY=72,8,2
           K 4 5 2    K1,2,6    K=10,2,7
```

1.2.5 TABLE

This parameter is used to define the special collating sequence tables to be used when parameter values 6 or 7 are used as the third integer in a key parameter.

The first time that the TABLE parameter is used, Table 6 is defined. Table 7 is defined the second time the TABLE parameter is used. The TABLE parameter may be used only twice. After the TABLE parameter there should be one delimiter (blank, comma, or equals) followed by a string of BCD characters. The string of characters is terminated when any character is repeated. The string of characters determines the collating sequence. All characters not in the string will have equal values, higher than any character in the string.

EXAMPLES:

1) A collating sequence to order playing cards could be defined by the following TABLE parameter:

   TABLE=AKQJT987654322

2) To order pinocle playing cards the following TABLE parameter could be used:

   TABLE,ATKQJ9A

3) To order playing cards by suits, the following TABLE may be used:

```
TABLE=SHDCC
```

The left parenthesis, right parenthesis, and V (internal BCD code of 52B) are given special treatment when they occur in a TABLE string. The parentheses may enclose a group of characters that are to be treated as having an equal value. In this manner sort tables may be made that handle zone or numeric collating sequences. For example, the following TABLE parameter defines a table that ignores the 11 punch that may occur in some types of numeric data.

```
T=(0VV)(1J)(2K)(3L)(4M)(5N)(6O)(7P)(8Q)(9R)
```

In the above table the 1 and the J will be treated exactly the same way. Both the 1 and the J will be treated as having a higher value than the 0 or V and a lower value than the 2 or K. Since the left and right parentheses have special meanings, if it is desired to include them in the table, it is necessary to precede them with the V symbol. The V symbol simply specifies that the symbol that follows it is not to be treated as a special symbol. Hence, to include the V symbol, duplicate it.

```
EXAMPLES:  TABLE=(0123456789)(ABCDEFGHI)(JKLMNOPQR)
                 (/STUVWXYZ)
           TABLE=(:=X,*+-V(V)VV/;)0123456789
```

NOTE: The following four parameters: (RECORD LENGTH, BLOCKING FACTOR, LENGTH ERROR, and PARITY ERROR) apply to the INPUT or OUTPUT parameter that precedes them.

Any of the following four parameters which occur before the first INPUT and OUTPUT parameter, is considered to be a default option and will apply to all INPUT and OUTPUT parameters which do not have specific parameters.

If default values are not provided for any of the following parameters, the SORT program will assume a default for the particular parameter. The assumed default values are:

RECORD LENGTH=100	(words)
BLOCKING FACTOR=0	(implies variable length records)
LENGTH ERROR=0	(terminate on length error)
PARITY ERROR=0	(terminate on parity error)

1.2.6 RECORD LENGTH

The record length (in words) should follow the RECORD parameter. For a detailed description of how this parameter is to be used with various kinds of data formats see section 3 on data formats.

```
EXAMPLES:  RECORD LENGTH=20
           R=100
           R,21
           R 10
```

1.2.7 BLOCKING FACTOR

The blocking factor or number of records in a block should follow the BLOCKING parameter. A blocking factor of zero implies variable length records. A blocking factor of one implies fixed length records. A blocking factor greater than one implies that blocks are to be read containing no more than B records where B is the specified blocking factor.

```
EXAMPLES:  BLOCKING FACTOR=10  (blocked ten)
           B=0  (variable length)
           B,1  (fixed length)
           B 0
```

1.2.8 LENGTH ERRORS

This parameter controls the way SORT handles block length errors (see section 4.2). Following the LENGTH ERROR parameter should be a zero, one, or two. Length errors cannot occur on the output unit or when using variable length records.

The meanings of the LENGTH ERROR parameter are:

0 - terminate on reading an incorrect length block after writing an error message

1 - write an error message and skip the block

2 - write an error message and use that portion of the block which contains complete records

```
EXAMPLES:  LENGTH ERROR=2
           L=1
           L 1
```

1.2.9 PARITY ERROR

Following the PARITY ERROR parameter should be a zero, one, or two. PARITY ERROR cannot occur on the output unit.

The meanings of the PARITY ERROR parameter are:

0 - terminate on a parity error

1 - skip the record with the error

2 - use the record with the error

In any case, an error message will be written.

```
EXAMPLES:  P=1
           P,0
           P IS 2
           P 2
```

2. CORE REQUIREMENTS

Rarely should a user of SORT have to concern himself with the memory requirements of SORT. This section is provided for users with unusual sorting requirements and for users planning to call SORT from a large running program.

The SORT program is about 2,000 decimal words long. It will use as a scratch work area all memory between the contents of an externally defined word called MEMLOW and the contents of an externally defined word called MEMHIGH. If SORT is not called as a subprogram, there are about 30,000 words of scratch work area. If there is not sufficient scratch work area for sorting or merging (internal) an M PARAMETER ERROR will result (see section 4.1).

To determine the minimum amount of scratch work area, the minimum memory requirement for sorting and the minimum memory required for merging must be calculated and the larger of the two quantities selected. Both sorting and merging will be faster if more than the minimum amount of memory is available.

The following formulas may be used to calculate the minimum scratch work area requirements for sorting and merging.

For SORT phase

$$M = 4K + 2R + MAXB + 600$$

For internal merging phase

$$M = 1.5K + 2R + B + 2000$$

M is the minimum scratch work area in words.
K is the number of characters in all keys.
R is the largest record length.
MAXB is the largest input block size or the sum of all output block sizes, whichever is larger. Zero if no blocking.
B is the sum of all output block sizes. Zero if no output blocking.

3. FILE SPECIFICATIONS

It is necessary to describe to SORT the structure of the input files so that SORT will interpret them correctly. Likewise it is necessary to tell SORT the desired output file structure. The file structure is specified after each INPUT parameter (one for each input unit) and after the OUTPUT parameter (for the output unit). SORT recognizes three basic file structures: variable length records, fixed length records, and simple blocked records. The B parameter (blocking factor) indicates the type of file structure to the SORT program.

3.1 VARIABLE-LENGTH RECORDS

Variable-length records are specified when the blocking factor is zero. The record length parameter determines the length of the longest whole record that may be read. Records longer than the record length will be truncated to the specified record length when being read or written.

3.2 FIXED-LENGTH RECORDS

Fixed-length records are specified when the blocking factor is one. On output, the record length parameter determines the length of all output records. Records may be truncated or filled with blanks (BCD) or zeros (binary) to produce the correct length records. On input, any record that does not have the length specified by the record length parameter will produce an error message and will be handled differently according to the length error parameter (section 1.2.8).

NOTE: The card reader under OS-3 is not a fixed-length input device. It may suppress trailing blanks. Usually fixed-length records should not be read from the card reader.

3.3 BLOCKED RECORDS

Blocked records are specified when the blocking factor is greater than one. On output, the logical records will be truncated or trailing blanks (BCD) or zeros (binary) will be added to produce the correct length record. These logical records will be blocked N to a block where N is the blocking factor. If there is not a multiple of N logical records (where N is the blocking factor), a partially filled block (physical record) will be written as the last block of the output file. This block will have a length equal to the number of records in the block times the length of a logical record.

On input, blocks with an integral number of logical records less than or equal to the blocking factor will be accepted. Long blocks (truncated) or blocks which do not contain a whole number of records will be treated as specified by the value of the length error parameter (section 1.2.8).

3.4 MULTIPLE TAPE CONVENTIONS

While reading from a tape unit, if a file mark (end-of-file) is read and end-of-data (end of tape reflector) status is present, the SORT program will assume that there is another input tape that is a continuation to the current tape. The current tape will be rewound and unequipped and a new tape (probably tape #1) will be equipped and reading will continue from the new tape.

If end-of-data (end of tape reflector) occurs on an output tape, the SORT program will write a tape mark (end-of-file), rewind, and unequip the tape. It will then ask for a new output tape to be mounted (probably tape #1 unless already in use).

4. ERROR MESSAGES

Various error messages may occur when using the SORT program. Section 4.1 describes errors that will keep SORT from processing any data. Sections 4.2, 4.3, and 4.4 describe errors that may occur while processing data.

4.1 PARAMETER ERRORS

The various error messages and conditions that may cause the message to occur are listed below. All of these errors will cause the SORT program to terminate before reading any information from the SORT input units.

PARAMETER ERROR

An illegal letter or special character was found as the first character on the parameter cards or the first character after an integer. The only legal characters are B, E, I, K, L, O, P, R, T, comma, and equals.

EXAMPLE: The Q is not legal. INPUT=5 Q O=7

K PARAMETER ERROR

In the key parameter the first parameter must be between 1 and 10,000. The sum of the first two parameters must be less than 10,000. The second parameter must not be zero. The third parameter (if present) must not be greater than 7.

M PARAMETER ERROR

The B, R, and K parameters require more than the available amount of core memory to perform the SORT. See section 2 on Core Requirements.

T PARAMETER ERROR

More than two sort tables are being specified.

R PARAMETER ERROR

A record length of zero is not legal.

I PARAMETER ERROR

Illegal logical unit number is being used.

No input units.

Too many input and output units are specified.

O PARAMETER ERROR

Illegal logical unit number is being used.

Too many input and output units are specified.

4.2 BLOCK LENGTH ERROR

This error message will always occur if a block (physical record) that does not contain an integral number of logical records is read from a SORT input unit. It will also occur if the block contains more logical records than specified by the BLOCKING parameter. This error cannot occur on output. The form of the error message is:

```
BLOCK LENGTH ERROR LUN XX       XXXXX RECORDS
```

4.3 PARITY ERRORS

This error message may occur on input only when an irrecoverable input error has occurred. When reading from a file it indicates that the file has become abnormal/unavailable. When reading from a tape drive this message indicates a permanent read error. The form of the message is:

```
PARITY ERROR ON LUN XX     XXXXX RECORDS
```

4.4 OTHER MESSAGES

The parameter cards, record, and record counts for each input and output device are listed at the end of a job. The format for the record count message is:

```
LUN XX      XXXXX RECORDS IN
       or
LUN XX      XXXXX RECORDS OUT
       or
XXXXX RECORDS DELETED
```

5. TIMING

No thorough study of timing has been made. However, it is possible to make some estimate of timing using the following formula:

$$T=(R/4+11)*N$$

where R is number of words in each record, N is the number of records to be sorted, T is CPU time in milliseconds.

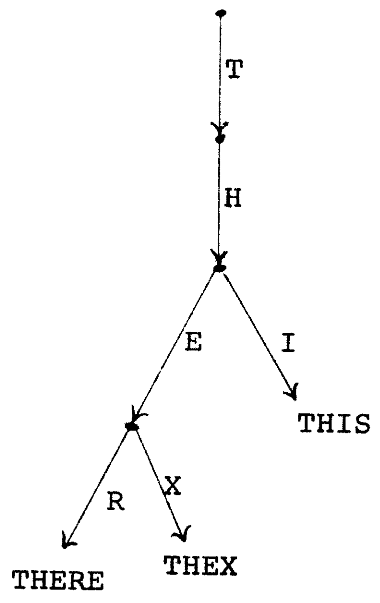
For sorts of less than about 1,000 records, no merge passes will be necessary and CPU times may be better than the formula indicates. For sorts of more than about 32,000 records, more than one merge pass will be necessary and times may be longer than the formula indicates. Using blocked input and output may reduce CPU time.

6. METHOD OR ALGORITHM

Sorting

During sorting, records are read into an ordered data structure. The data structure is such that the records can be written out in correct order. The data structure is a tree with ordered nodes for every significant character in the key of every record. The tree structure for

THEX, THIS, and THERE would be:



By "picking" the records off the tree from left to right, the records will be in order.

If memory becomes full while sorting, all records read are written into a merge unit. Up to 32 merge units may be used.

Merging

If memory does not become full, the output from SORT goes directly to the output unit. If merging is necessary, the merge phase reads from all the previous merge output units into as many as 32 more merge units. During internal merging, great care is taken to keep pre-ordered records from being shuffled when information in the sort keys is identical. Each time a merge pass is completed, the sorted strings on the merge units become 32 times longer. The final merge pass merges from up to 32 units onto the output unit. All merge files that the SORT program had equipped are now unequipped. A filemark is written on the output unit and the output unit is backspaced if it is a file or magnetic tape.

Tournament replacement sorting is done during internal merging.

7. CALLING SORT AS A SUBROUTINE

If it is desired to sort a data file as a step in a program the user may wish to call SORT from his program with a subroutine call. There are two ways this may be done. The SORT program may be loaded from the FORTRAN library and share the lower 32K memory with the user's program (see section 7.1), or the user may elect to use a version of SORT that runs in the upper 32K memory allowing the user's program to use all of the lower 32K memory (see section 7.2).

Source and binary decks of the SORT program may be obtained from the Computer Center program library. SORT may also be found on the FORTRAN library.

7.1 CALLING SORT FROM A PROGRAM

This section describes how to call SORT from a program so that the SORT subroutine is loaded into the lower 32K memory along with the user's program. The SORT subroutine requires about 2,000 (decimal) words of memory for the program and then requires additional scratch memory in order to run (see section 2). If the user's program is very large, if large arrays are needed, or if large blocked records are to be sorted, there may not be enough scratch memory available for the SORT subprogram to run efficiently. The amount of scratch memory available to

SORT may be found by computing the difference between the values of HIGHMEM and LOWMEM, symbols found near the end of a loader map made when the program is loaded.

The input and output logical unit numbers must be passed to the SORT program. The following is a FORTRAN calling sequence that may be used.

```
      CALL SORT(ILUN,IOLUN)
```

In the above subroutine call, the parameter ILUN may be an integer (or integer variable) which is the logical unit number that the SORT program will use when reading parameter records. However, if it is desired to pass the input parameters in memory, ILUN may be the name of a word array containing the SORT parameters as BCD characters. IOLUN is an integer that is the logical unit that will be used when writing error messages and record counts.

The SORT program may also be called as a function. It will return a floating value of one on normal termination and will return a value of zero on abnormal termination (caused by errors found in the parameter records, parity errors, or block size errors). The following is an example of the function calling sequence that may be used.

```
      IF (SORT(60,61)) 24, 992
```

Logical units 60 and 61 are used. Statement 24 will be executed when sorting is completed. Statement 992 will be executed if sorting could not be performed. If sorting could not be performed, an error message will be written on the output unit.

The COMPASS calling sequence is as follows:

```
              EXT      SORT
              RTJ      SORT
              77       ILUN
              77       OLUN
```

```
        AZJ,EQ    ABNORM    Abnormal termination
        AZJ,NE    NORM      Normal termination
ILUN    DEC       60        Input Logical Unit
OLUN    DEC       61        Output Logical Unit
```

When any of the methods of calling SORT described in this section (section 7.1) are used, the SORT program will call a number of subprograms which the user may define. The user may write subprograms, called exits, which may perform input and output file positioning, label checking, input data selection and modification, and output data formatting. If the user does not desire to use any or all of the exits, the standard exit program from the FORTRAN library will be loaded.

7.1.1 EXIT 1, INPUT RECORD EXAMINATION

EXIT 1 enables the user of SORT to examine, modify and/or delete records after they have been read (and unblocked if necessary) but before SORT examines the record. The following subroutine is a model EXIT 1 program containing only the necessary linkage.

```
      SUBROUTINE SEXIT1(IA,IC,LENG,IUNIT)
      CHARACTER IC
      DIMENSION IA(20),IC(80)
C     **BODY OF FORTRAN PROGRAM**
      RETURN
      END
```

In the above subroutine, IA is an integer array containing the last logical record that was read. The character array, IC, is also the record that was just read. The user may examine and modify the record using either or both the integer and character arrays. The length of the record in words is in LENG. Hence, the only valid values of the subscript of IA would be 1 through LENG. Likewise, the only valid values of the subscript of IC would be 1 through 4*LENG.

The integer IUNIT contains the SORT unit number. That is, if the record was read from the third input unit in the SORT parameter list, IUNIT will have the value of three. If IUNIT is set to zero, the record will be deleted by the program.

7.1.2 EXIT 2, OUTPUT RECORD EXAMINATION

EXIT 2 enables the user of SORT to modify and select an output unit for each logical record after the record has been sorted but before it has been written. The following subroutine is a model EXIT 2 program containing only necessary linkage.

```
      SUBROUTINE SEXIT2(IA,IC,LENG,IUNIT)
      CHARACTER IC
      DIMENSION IA(20),IC(80)
C     **BODY OF FORTRAN PROGRAM**
      RETURN
      END
```

In the above subroutine IA, IC, and LENG have the same functions as they did in the SEXIT1 program (EXIT 1).

The integer IUNIT contains the SORT output unit number on which the record would normally be written. SORT will always set IUNIT to one. If IUNIT is set to zero, the record is to be deleted. If IUNIT is one, the record is to be written on the first output unit; if two, the second output unit, and so on. The EXIT 2 program may change IUNIT causing the record to be deleted or written onto a different logical unit. If zero or an illegal output unit is specified, the record is deleted.

7.1.3 EXIT 1A, INPUT FILE OPEN

EXIT 1A enables the user of SORT to position and do label checking on SORT input units before SORT reads from the unit. Without EXIT 1A, SORT will expect the input units to be properly positioned and will not do any label checking. The following example is an EXIT 1A program which will rewind each SORT input unit before it is used. ILUN is the actual logical unit number.

```
SUBROUTINE SEXIT1A(ILUN)
REWIND ILUN
RETURN
END
```

7.1.4 EXIT 1B, INPUT FILE CLOSE

EXIT 1B should be used when special file positioning or label checking is desired after SORT has completed reading from a SORT input unit. SORT will stop reading from an input unit when either a filemark or end of data is found. Without EXIT 1B, SORT will leave the unit positioned after the filemark or at the end of data and will not do any trailer label checking. The following example is an EXIT 1B program which is designed to rewind and unequip input units which have been processed.

```
SUBROUTINE SEXIT1B(ILUN)
REWIND ILUN
CALL UNEQUIP(ILUN)
RETURN
END
```

7.1.5 EXIT 1C, INPUT END OF TAPE

EXIT 1C is called by SORT only when SORT has detected an end of file, and when the end of tape condition is also present. Without EXIT 1C, SORT will rewind and unequip an input unit which is at end of tape. SORT will then equip a new tape (probably tape #1) and continue to read from it. EXIT 1C has the same parameters as EXITS 1A and 1B.

7.1.6 EXIT 2A, OUTPUT FILE OPEN

EXIT 2A enables the user of SORT to position and do label checking or label writing on the SORT output unit before SORT writes on the unit. Without EXIT 2A, SORT will expect the output units to be properly positioned and will not do any label checking. The following example is an exit 2A program which writes a user label on the SORT output unit.

```
         SUBPROGRAM SEXIT2A(ILUN)
         REWIND ILUN
         WRITE(ILUN,99)
   99    FORMAT(≠ SORTED OUTPUT FOLLOWS≠)
         RETURN
         END
```

7.1.7 EXIT 2B, OUTPUT FILE CLOSE

EXIT 2B should be used when special positioning or label writing is desired after SORT has completed writing on a SORT output unit. Without EXIT 2B, SORT will write a filemark and, if the output unit is a tape or file, will backspace past the filemark. When EXIT 2B is used, SORT will always write a filemark and then call EXIT 2B. EXIT 2B has the same parameters as EXIT 2A.

7.1.8 EXIT 2C, OUTPUT END OF TAPE

EXIT 2C is called by SORT only when SORT has detected the end of tape condition. Without EXIT 2C, SORT will write a filemark, rewind, and unequip the tape. SORT will then equip a new tape (probably tape #1) and continue to write on it. When EXIT 2C is used, SORT will always write a filemark on the tape and then call EXIT 2C. EXIT 2C has the same parameters as EXIT 2A.

7.2 CALLING OVSORT FROM A PROGRAM

This section describes how to call SORT from a program so that the SORT subroutine is loaded into the upper 32K memory bank. Loading the SORT program into the upper 32K memory bank makes it possible for a user with a large program to include a file sort as a step in his program and yet only increase the length of his program about 50 words. However, since the SORT program is in the upper bank of memory, the SORT exits may not be used and the SORT parameters may not be passed in memory.

The following is a FORTRAN calling sequence that may be used.

```
CALL OVSORT(ILUN,IOLUN)
```

In the above subroutine call ILUN is an integer (or integer variable) that is the logical unit number that the SORT program will use when reading parameter records. IOLUN is an integer that is the logical unit number that will be used when writing error messages and record counts.

The FORTRAN function call and COMPASS calling sequence have the same form as described in section 7.1 except that OVSORT is used in place of SORT.

8. EXAMPLES

Example 1

A deck of cards containing names of people is to be sorted into alphabetical order to be listed on the printer. Columns 1-30 contain the person's name.

$^{7}_{8}$JOB,

$^{7}_{8}$EQUIP,1=FILE

```
7
8SORT
 I 60          Input on unit 60 (cards)
 O 1           Output on unit 1
 K 1,30        Sort on columns 1 for 30 columns
 END           End  of parameters
<cards to be sorted>
77
88             End of file

7
8COPY,I=1/R,O=61       List the sorted file on the printer

7
8LOGOFF
```

Example 2

A file containing fish observations at various observation stations is to be processed to make a report on the number of fish falling into various categories at each observation station. The report is to list for each station a daily summary of the fish in each category. Before the report may be made, the data (on the file called FISHDATA) must be grouped by observation station. Within each observation station, the data needs to be grouped by day, then by species and finally by length. The data has the following format:

Item	Columns
Observation Station ID	1-2
Julian Date	5-7
Year	8-9
Species ID	15-18
Length	11-13

Here is a sort that may be used to group the data for the report program.

```
7
8JOB,

7
8EQUIP,25=FISHDATA
```

```
7
8EQUIP,29=FILE

7
8SORT
 I=25                        Read input from logical unit 25
 O=29                        Write output onto logical unit 29
 KEY   1,2                   Sort first by Station ID within
 KEY   8,2                     each station, sort by year
 KEY   5,3                     within each year, sort by day
 KEY  15,3                     within each day, sort by species
 KEY  11,3                     within each species, sort by length

7
8REWIND,29
```

Example 3

Many sort applications involve records which may contain identical keys. Take, for instance, a computerized banking system that makes, for every transaction during a day, a card containing the bank account number, the type of transaction, and the amount of the transaction. In order to process the transactions for the day the transaction for each account must be grouped. Furthermore, deposits must be processed before withdrawals. However, when more than one deposit is received for the same account (an identical key) it is desirable that they stay in the order of arrival. Here is an example sort that the bank could use to order the transactions.

Item	Columns
Bank Number	2-3 (binary number)
Type of Account	10 (numerical)
Account Number	4-9 (numerical)
Type of Transaction	12 (alphabetic)

Transaction Code	Meaning
N	Opening new account
A	Correction
D	Deposit
W	Withdrawal
C	Closure

```
7
8JOB,

7
8EQUIP,10=RAWDATA

7
8EQUIP,11=FILE

7
8SORT
 I 10          Input unit
 O 11          Output unit
 KEY 2,2,2     Bank number
 KEY 10,1      Account type
 KEY 4,6       Account number

 TABLE=NADWC
 KEY 12,1,6    Transaction type
 END

7
8SAVE,11=SORTEDF

7
8LOGOFF
```

Example 4

For many applications of SORT it may be desirable to block either the input or output, or both. If magnetic tape is used the cost of reading the tape is generally much less if the records are blocked. Even with disk files, blocking will usually reduce the cost of the sort. A word of caution is in order, however. Blocked files may be awkward to use and may result in lost time due to additional debugging. Also, if the block size becomes very large the sort will become inefficient since the memory used by the large blocks is not available as scratch work area for SORT. A reasonable block size would be 1,000 words.

The following sort shows how unblocked records may be blocked to produce a blocked, sorted file. The output of the sort is a magnetic tape. The sort key is a signed floating point number in the second word of each record.

```
7
8JOB

7
8EQUIP,1=DATA

7
8EQUIP,2=MT 7032 AT 556 WITH RING

7
8SORT
 I=1                  Read variable length records from unit 1
 O=2,R=20,B=50        Write 20 word records blocked 50
 KEY 5,8,4            Signed binary, 2 words (floating point)
 ENDSORT              END or ENDSORT may be used to indicate
7                            sort terminations
8LOGOFF
```

Example 5

The following sort shows how unblocked records and blocked records may be sorted to produce blocked output.

```
7
8JOB,

7
8EQUIP,1=DATA5

7
8EQUIP,2=MT 7032 AT 556 NO RING

7
8EQUIP,3=MT 7047 AT 800 WITH RING

7
8SORT
 I=1
 I=2,R=20,B=50
 O=3,R=20,B=50
 KEY 5,8,4
 END

7
8LOGOFF
```

Example 6

If, in example 5, the data on the input tape (number 7032) was sorted it would usually be more economical to sort only the unsorted file (DATA5) and then merge it with the sorted file. In general, it is usually wise to replace large sorts with merges when possible since sorting almost always uses more computer time and memory than merging.

The following sort and merge will achieve the same result as the sort in example 5, but will probably cost less.

```
7
8JOB,

7
8EQUIP,1=DATA5

7
8EQUIP,10=FILE

7
8SORT
 I 1
 O 10,R20,B50
 K 5,8,4
 END

7
8REWIND,10

7
8EQUIP,2=MT 7032 AT 556 NO RING

7
8EQUIP,3=MT 7047 AT 800 WITH RING

7
8MERGE
 I 10  R20 B50
 I 2   R20 B50
 O 3   R20 B50
 K 5   8   4
 E

7
8LOGOFF
```

9. INDEX