Using DFSORT and ICETOOL

Using DFSORT and ICETOOL - Course Objectives

On successful completion of this class, the student, with the aid of the appropriate reference materials, should be able to:

1. Write JCL and DFSORT control statements to sort, copy, and merge records in the following kinds of files as input, and to produce these kinds of files as output:
   * Sequential data sets
   * VSAM data sets
   * Members of PDS or PDSE
   * Files in the z/OS File System (zFS - UNIX file support)

2. Create symbolic name files to use in DFSORT and ICETOOL control statements

3. Write JCL and DFSORT and ICETOOL control statements that enable you to perform these tasks with no need to write code in a programming language:
   * Work with subsets of files (filter / extract out records)
   * Build new records, reformat existing records, even recognizing and handling different record layouts in the same file; including:
     - adding, removing, and reordering fields
     - converting data type of fields, and editing content
     - doing arithmetic calculations, inserting sequence numbers
     - replace values using lookup pairs in specific locations
     - replace values in any location
   * Extract fixed length values from variable length fields (parse)
   * Output multiple files in a single pass, including XML, HTML, and reports with up to three levels of control breaks
   * Working with dates and times in a wide variety of formats, including SMF, TOD, and ETOD, and working with dates with two digit years
   * Convert betweeen fixed length record files and variable length record files
   * Join records from two files and process the resulting record set
   * Use locales for sorting and copying, and work with ASCII files
   * Combine fields from two or more files into records in a single file.

Using DFSORT and ICETOOL - Topical Outline

Day One

Introduction to DFSORT
- Background
- Computer Exercise: Setting Up for Labs 10

The DFSORT Program
- DFSORT Capabilities
- JCL and Control Statements for DFSORT
- Introduction to INCLUDE / OMIT Statements
- Introduction to the INREC Statement
- Introduction to the SORT Statement
- Introduction to the OUTREC Statement
- Using SORT to do a copy
- Computer Exercise: Running Sorts 29

Data Types and Symbolic Names
- Data Types
  - CH, AQ, ZD, ZDF, ZDC, PD, PDF, PDC, CSF, UFF, SFF, CSL, CST, CLO, CTO, FI, FL, BI, AC, ASL, AST
- Symbolic Names
- Literals
- Using Symbolic Names
- Converting values
- Additional symbolic name facilities
- Computer Exercise: Using Names 56

A Deeper Look at INCLUDE, OMIT, and SORT statements
- INCLUDE / OMIT: Additional COND tests
- The Complete SORT Statement
- Computer Exercise: Using Additional Tests and SORT Operands 69

The INREC and OUTREC Statements, round 2
- The Roles of INREC and OUTREC
- The PARSE Operand
- PARSE and symbolic names
- Computer Exercise: PARSE 88

Using DFSORT and ICETOOL - Topical Outline, p.2.

The INREC and OUTREC Statements, round 3

The BUILD operand

BUILD Values

Computer Exercise: Using BUILD 129

The INREC and OUTREC Statements, round 4

The OVERLAY operand

The FINDREP operand

Computer Exercise: OVERLAY and FINDREP 142

Day Two

The INREC and OUTREC Statements, round 5

The IFTHEN operand

Computer Exercise: IFTHEN 167

Working with Dates

Dates

Dates with four digit years

Dates with two digit years

Enhanced date processing

Date Field arithmetic

Computer Exercise: Sort and Format Dates 217

Working with Times

Times

OUTFIL - Multiple output files

Some Perspective

The OUTFIL statement

Computer Exercise: Using OUTFIL 242

OUTFIL, round 2 - Reports

Report terminology

Report related operands of OUTFIL

Headers, Trailers, Control Breaks

Computer Exercise: Generating Reports 290

Using DFSORT and ICETOOL - Topical Outline, p.3.

Day Three

OUTFIL, round 3 - Markup
- Markup Languages
- Introduction to XML
- DFSORT and XML
- HTML - An Introduction
- DFSORT and HTML
- Computer Exercise: Generating Markup 326

Working with zFS Files
- z/OS UNIX
- Introduction to the z/OS File System (zFS)
- zFS JCL Parameters
- JCL and zFS Files: DFSORT Usage
- Copying data to the zFS
- Computer Exercise: Using zFS Files with DFSORT 345

Alternative Orderings
- Collation sequence
- ALTSEQ - Specifying alternative collating sequences
- Locales - Ordering with an awareness of languages and formatting conventions
- Sorting ASCII files
- Computer Exercise: Sort an ASCII File 360

Additional DFSORT Control Statements
- DFSORT Statements
- Exits
- The SUM Statement
- The RECORD Statement
- Merge Operations
- The MERGE Statement
- The OPTION Statement
- JCL Statements Revisited
- Computer Exercise: Using Additional DFSORT facilities 382

Joining Files for a SORT or COPY operation
JOIN concepts
The JOINKEYS, JOIN, and REFORMAT statements
JOINKEYS Applications Notes
Computer Exercise: A JOINKEYS Application 404

Day Four

Introduction to ICETOOL
ICETOOL Overview
ICETOOL COPY operator
ICETOOL COUNT operator
Numeric editing in ICETOOL
ICETOOL DEFAULTS operator
ICETOOL MERGE operator
ICETOOL MODE operator
ICETOOL RANGE operator
ICETOOL SORT operator
ICETOOL STATS operator
ICETOOL UNIQUE operator
ICETOOL VERIFY operator
Computer Exercise: Introduction to ICETOOL 438

The ICETOOL DISPLAY operator
The DISPLAY Operator
DISPLAY examples
Computer Exercise: DISPLAYing Data 456

The ICETOOL OCCUR operator
The OCCUR Operator
OCCUR examples
Comparing ICETOOL Operators
Computer Exercise: Analyzing Data Patterns 470

Using DFSORT and ICETOOL - Topical Outline, p.5.

The ICETOOL RESIZE, DATASORT, SUBSET, and SELECT operators
- The RESIZE operator
- The DATASORT operator
- The SUBSET operator
- The SELECT operator
- Computer Exercise: Using SELECT 491

The ICETOOL SPLICE operator
- The SPLICE operator
- Computer Exercise: SPLICE-ing Files 512

Loose Ends
- But Wait! There's More!
- The ICEGENER utility
- VSAM support
- Work data sets
- Sorting Techniques
- Using JCL Symbolic Parameters and SET symbols in DFSORT and ICETOOL control statements
- Tape files
- Performance
- Miscellaneous Notes

Note: this course is a thorough introduction to the facilities of DFSORT and ICETOOL, omitting discussions of exits and the Extended Function Support.

While we cover all the control statements and operands in varying degree, with lots of examples, you should consider this a starting point.

The next stop on your journey would be to examine the publications on the DFSORT website, especially the "DFSORT Application Programming Guide", which is the definitive source for this material, and "Smart DFSORT Tricks", which provides additional examples of applications for DFSORT and ICETOOL based on requests from customers.

You can find these publications, and more, on the DFSORT website beginning at:

https://www.ibm.com/support/pages/dfsort then follow the links to details.
for the Smart Tricks: https://www.ibm.com/support/pages/node/665475

Section Preview

❒ **Introduction to DFSORT**

Background

Setting up for labs (Machine Exercise)

SORT - Background

❒ **When OS/360 first came out (this was the ancestor of today's z/OS), it included a free sort/merge utility as part of the operating system**

But it wasn't very good:

✗ Competitors came out and were charging for their sort products and were succeeding, even against the free sort included with OS/360!

❒ **At some point, IBM either got embarrassed or they saw a potential revenue stream and they started paying attention to their sort**

❒ **This competition was good, since all the major players competed against each other, each improving performance and functionality**

Often taking turns leapfrogging each other as the "best" available sort product

Today, the major players in the Sort market are IBM's DFSORT and SyncSort from Syncsort, Inc.

✗ At least one other sort package is in the fray, but these two have, by far, the largest market share

❒ **Another nice feature of the competition: often these two products have the same syntax for control statements: this makes it easier for a customer to switch from one to the other!**

❒ **So, while this course is about DFSORT, much of the content also applies to SyncSort**

SORT - Background, 2

❒ **Today, DFSORT has evolved to be a very sophisticated tool that can:**

Sort the records in a file into a desired sequence based on one or more sort keys

- ✗ The records can arrive from a sequential file, a VSAM file, a member of a PDS or PDSE, or a file in the zFS (z/OS File System - files used by z/OS UNIX System Services)
  - ➢ Or they can be inserted from any other source using a locally written exit routine
- ✗ You can sort all the records, or just some of the records
- ✗ You can reformat the input records before or after sorting
  - ➢ You can process records after sorting but before the actual output writing using a locally written exit routine
- ✗ The records can go to multiple output files
- ✗ The output file(s) can be formatted using a rich collection of field editing, page formatting, and report building features

Merge records from multiple files into a single file - as long as all the records have the same general layout and are in the same relative sort sequence by one or more collation keys

Copy the records from one file to another

❒ **Sort, Copy and Merge can all filter records as described above, and convert between FB and VB records**

❒ **DFSORT comes with a powerful utility, ICETOOL, and a high speed replacement for IEBGENER called ICEGENER**

These are discussed in this course also

Computer Exercise: Setting up for labs

This lab establishes the environment for running all the subsequent labs in this course. To do this, you will run a little dialog that:

1. Establishes the naming conventions to use for your data sets
   (default: <*tsoid*>**.TR.***name*)

2. Establish the zFS directories to use for holding zFS files for input and output for sort operations
   (default: **/<u/**<*tsoid*>**/SortData)**

   (Note: this is bypassed if you do not have access to the zFS, or if you blank out the directory name in the dialog when you run it)

The dialog will then create the following files for you:

<hlq>.TR.CONTACTS	(flat file test data)
<hlq>.TR.CNTL	(JCL library with some members already there)
<hlq>.TR.CUSTOMER	(flat file test data)
<hlq>.TR.DATA	(data library with some members already there)
<hlq>.TR.INPUTA	(flat file test data, character, packed, binary data)
<hlq>.TR.INPUTX	(flat file test data, character, packed, binary data)
<hlq>.TR.INPUTE	(flat file test data, variable length records)
<hlq>.TR.LOCS	(flat file test data)
<hlq>.TR.ORDERS	(flat file test data)
<hlq>.TR.ZINPUTA	(flat file test data, character data only)
<hlq>.TR.ZINPUTX	(flat file test data, character data only)
<hlq>.TR.ZASCIIX	(flat file test data, ASCII version of above file)

If you will be running the zFS-related labs, the dialog will also create the two directories mentioned above.

To run this dialog, from ISPF option 6 enter this command:

```
===> exec '____________.train.library(b625strt)' exec
```

Note that the record layouts for all files are in the Appendix to this book.

Section Preview

❐ The DFSORT Program

DFSORT Capabilities

JCL and Control Statements for DFSORT

Introduction to INCLUDE / OMIT Statements

Introduction to the INREC Statement

Introduction to the SORT Statement

Introduction to the OUTREC Statement

Using SORT to do a copy

Running Sorts (Machine Exercise)

This page intentionally left almost blank.

DFSORT Capabilities

- ❒ **Sort one file into a new sequence (SORT statement)**

- ❒ **Merge several files (already in the same relative sequence) into one file (MERGE statement)**

- ❒ **Copy a file (SORT, MERGE, or OPTION statement)**

- ❒ **Inputs: Sequential, member of PDS or PDSE, VSAM, or zFS files**

- ❒ **Outputs: Sequential, member of PDS or PDSE, VSAM, or zFS**

  VSAM files must be pre-defined unless SMS is installed and active in your system

- ❒ **Sort on subsets**

  Only records that meet some criteria (INCLUDE statement)

  All records except those that meet some criteria (OMIT statement)

- ❒ **Reformat records**

  Before sorting (INREC statement)

  After sorting (OUTREC statement)

- ❒ **Join records from two files based on one or more key fields prior to sorting or copying (JOINKEYS, JOIN, REFORMAT statements)**

- ❒ **In this section, we provide a basic introduction to various DFSORT control statements**

  More complete discussions appear later

DFSORT Control

JCL

//SORTIT	EXEC	PGM=SORT	
//SORTIN	DD	point to the file to be sorted	(may be concatenated inputs)
//SORTOUT	DD	point to the output location	
//SYSIN	DD	$\boxed{\text{for DFSORT control statements}}$	
//SYSOUT	DD	SYSOUT=*	for DFSORT messages

.
.
.

There are additional DD statements possible; they are discussed later

DFSORT control statements

OPTION	code content in columns 2-71, inclusive
INCLUDE	
OMIT	indicate continuation by comma at end of
INREC	last operand on a line;
SORT	continuation lines coded in 2-71 also
OUTREC	
	others, discussed later

Introduction to INCLUDE / OMIT Statements

- ❒ **INCLUDE: only sort, copy, or merge records that meet the condition test(s)**

- ❒ **OMIT: sort, copy, or merge all records except those that meet the condition test(s)**

  May only specify one INCLUDE or one OMIT; may not specify one of each

- ❒ **Condition test(s) specified as "COND=" and a series of one or more tests:**

  Input record field, Relationship, Input record field

  or

  Input record field, Relationship, Constant

Where

field is specified as *starting_location*, *length*, **and** *data_type* **(CH, PD, ZD, FI, BI,** *etc.***)**

Relationship is one of: EQ (equal), NE (not equal), GT (greater than), GE (greater than or equal), LT (less than), or LE (less than or equal to)

Constant is a numeric literal, a character string (C'...'), or a hex string (X'...')

- ❒ **Tests are separated by "AND", or "&" or "OR" or "|"**

Introduction to INCLUDE / OMIT Statements, Examples

```
OMIT   COND=(34,5,CH,NE,C'ENTRY',AND,6,4,FI,GE,20,4,FI,
                                 OR,1,1,CH, EQ,X'FF')
```

Note that ANDs are evaluated before ORs; you can use parentheses to direct the order of evaluation

```
INCLUDE   COND=((3,3,CH,EQ,C'AAB'),&,
                ((17,7,CH,LT,333,7,CH),|,
                 (220,2,CH,EQ,C'RR')))
```

Analyzes as…

```
INCLUDE   COND=((3,3,CH,EQ,C'AAB'),&,((17,7,CH,LT,333,7,CH),|,(220,2,CH,EQ,C'RR')))
                  field  op literal     field    op  field       field    op literal
```

Another example

```
OMIT   COND=(20,4,FI,GT,4096)
```

Introduction to the INREC Statement

INREC   BUILD=([*separator*,]*position*,*length*[,...])

- ❒ **Reformats the input records with only separator characters and the described fields**

  That is, describe how to take input records and build the actual record to be sorted, copied, or merged

  The result of reformatting an input record is a "sort record" that contains data from the original record and various filler data (blanks, binary zeros, or literals); note: in this course, the term "sort record" can also mean records to be copied or merged

  - ✗ So the BUILD operand describes how the sort record is constructed from a combination of separator characters and data from the input records

  - ✗ Note that FIELDS is a an older synonym for BUILD on both INREC and OUTREC statements

    - ➢ We will use BUILD in the lecture points, but you may find old jobs that use FIELDS; FIELDS should only be used in SORT, MERGE, and SUM statements

  The sort records are built from byte one; the order in which data items are specified in the BUILD operand is the order they appear in the sort record, immediately adjacent to one another

- ❒ **Note: the brackets ("[" and "]" indicate a value is optional: do not code brackets in your code**

- ❒ **Punctuation, as shown, counts …**

INREC Operands

separator characters

*n*X — insert *n* bytes of blanks (1 < *n* < 4095), for example

30X

*n*Z — insert *n* bytes of binary zeroes (1 < *n* < 4095), *e.g.*

26Z

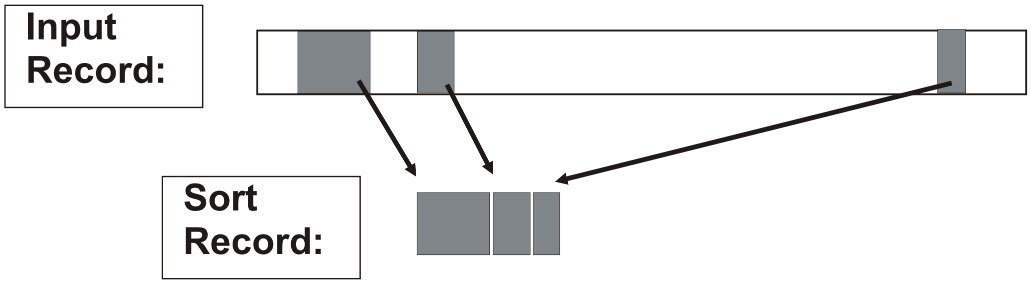
*n*C'...'— insert *n* repetitions of the string (1<*n*<4095);
or
*n*X'...'— string is hex or character string up to 256 characters

```
5C'*** HERE IT IS ***'
7X'80'
```

position,length

These two values are the actual starting position and length in the input record of the field to be extracted and put into the next available position in the "sort record"; you may build a sort record with no separators:

```
INREC   BUILD=(5,20,37,3,99,3)
```

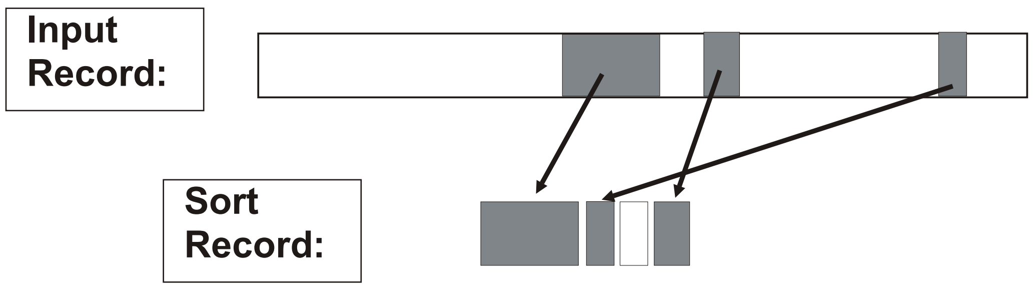


INREC Operands, 2

Another example

Extract fields from an input record to build a record to be sorted, inserting some binary zeros to extend a packed decimal field:

```
INREC        BUILD=(34,6,120,3,3Z,44,4)
```



❒ **The SORT statement FIELDS values (next page) describe positions in the "Sort Record", not the input record**

If your input record is variable length, be sure to allow for the 4-byte RDW - the first data byte is position 5

Introduction to the SORT Statement

```
SORT      FIELDS=(position,length,format,sequence[,...])
```

Identify the field(s) to sort on, in decreasing order of importance

```
SORT      FIELDS=(35,5,CH,A,3,2,PD,D,60,3,CH,A)
```

Notes on SORT FIELDS:

✗ All fields must be contained in the first 32752 bytes of the input records and the sum of the lengths of the control fields must be less than or equal to 4092 bytes

✗ If INREC is used to reformat input records, the *position* value is the position within the reformatted record

✗ Most common data formats are CH (character), BI (bit string), FI (fixed point signed), PD (packed decimal), and ZD (zoned decimal)

- ➢ After the data type, code A for ascending sequence or D for descending

✗ If you are sorting variable length records (including VSAM variable length records), the position values must allow for a 4 byte RDW at the front of the data

✗ Positions for BI data type can be specified on bit boundaries (*e.g.*: 5.3 is byte 5, bit 4); you may also specify lengths as some number of bytes and bits

Introduction to the SORT Statement, 2

- ❒ **If all the fields in a SORT statement are of the same data type, they may be specified with just three attributes and the whole list of fields must be preceded or followed by a FORMAT operand**

  For example

```
SORT       FORMAT=CH,FIELDS=(35,5,A,3,2,D,60,3,A)
```

  The attributes are *starting_position*, *length*, **and** *sequence*

- ❒ **If sorting will yield many records with the same sort key value (for example, zip code, mail code, area code, country code type fields), DFSORT does not guarantee records end up in the same relative order as originally found**

  Unless you code EQUALS on the SORT statement:

```
SORT      FIELDS=(35,5,CH,A,3,2,PD,D,60,3,CH,A),EQUALS
```

  Note: When DFSORT is installed, EQUALS may be set as the default; in this case you can specify NOEQUALS to suppress this behavior

  EQUALS or NOEQUALS may also be specified on the OPTION statement (discussed later) instead of on the SORT statement

Introduction to the OUTREC Statement

OUTREC   BUILD=([*separator*,]*position*,*length*[,...])

❒ **Reformats the sorted, copied, or merged records on output**

Operands have the same meaning and syntax as for INREC, but now formatting the final output records from the "sort record"s

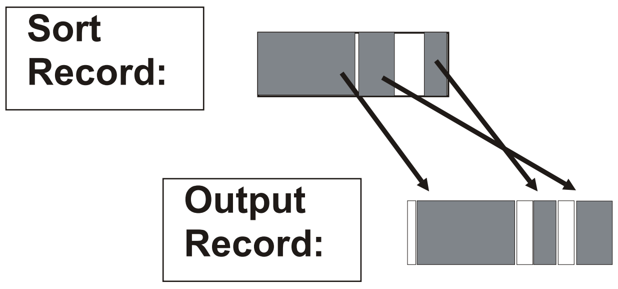
❒ **Nice for printed output especially**

Example

```
OUTREC      BUILD=(1X,1,30,2X,52,5,2X,31,10)
```

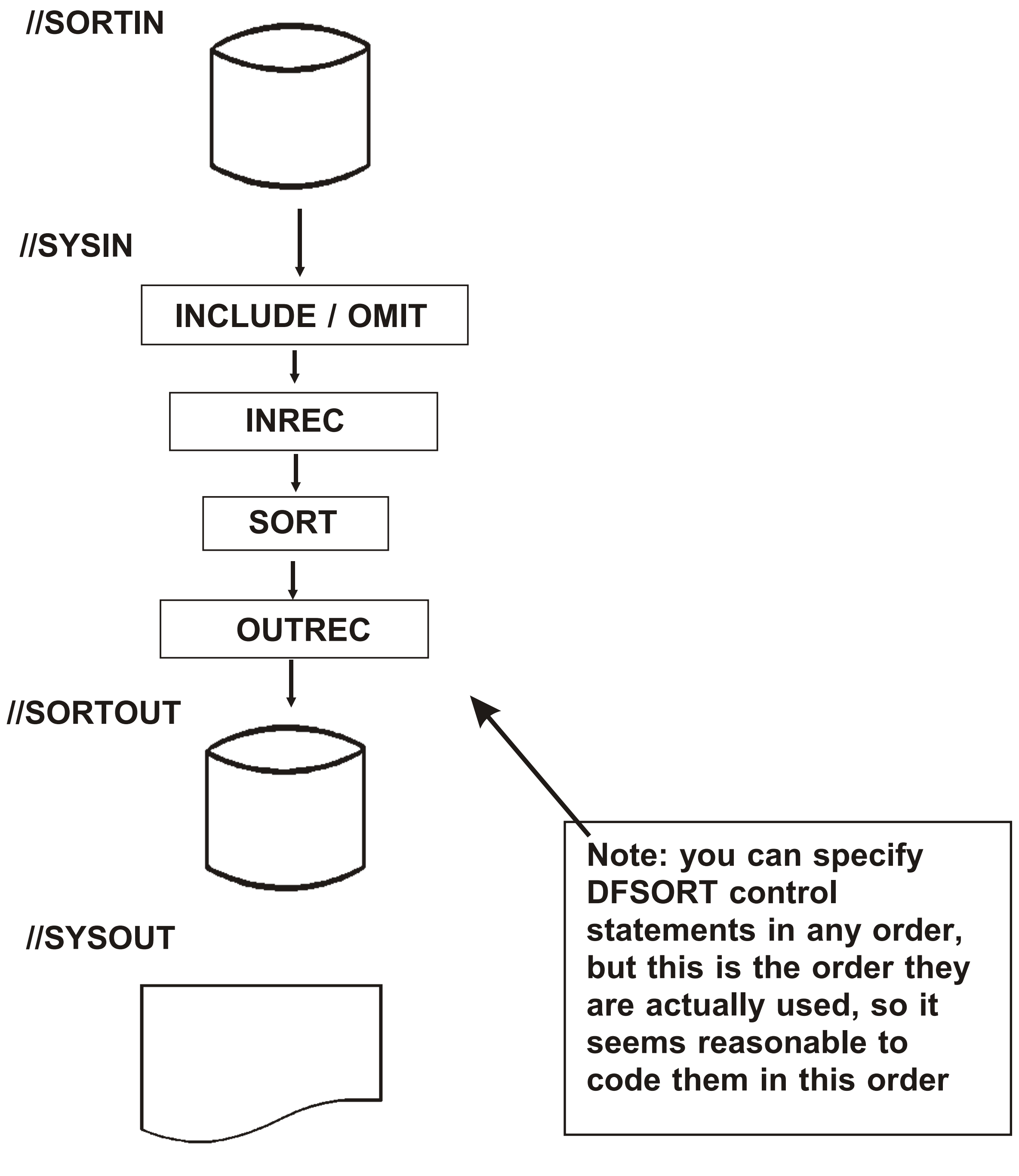
Picture: building output record from sort record

1 blank
1-30 from sort record
2 blanks
52-56 from sort record
2 blanks
31-40 from sort record



❒ **Another application: inserting delimiters such as commas when passing the data to a program that handles such data (for example downloading to a PC database product that can handle comma-delimited input files)**

Control Statement Actions



Note: you can specify DFSORT control statements in any order, but this is the order they are actually used, so it seems reasonable to code them in this order

Sample SORT Steps

```
.
.
.
//STEP13      EXEC    PGM=SORT
//SORTIN      DD      DISP=SHR,DSN=GR55.TESTER.DATA
//SORTOUT     DD      DSN=GR55.ADDRESS,DISP=(,CATLG),
//        SPACE=(260,(10,50)),AVGREC=K
//SYSIN       DD      *
          INCLUDE COND=...
          SORT BUILD=...
//SYSOUT      DD      SYSOUT=*
.
.
.
```

```
.
.
.
//STEP23      EXEC    PGM=ICEMAN
//SORTIN      DD      DISP=SHR,DSN=AR55.RESTER.DATA
//SORTOUT     DD      DSN=AR55.PHONES,DISP=(,CATLG),
//        LIKE=GR55.ADDRESS
//SYSIN       DD      DISP=SHR,DSN=AR55.PARMLIB(PHONES)
//SYSOUT      DD      SYSOUT=*
.
.
.
```

❐ **Note that the actual program name is ICEMAN; there are a number of aliases established for compatibility with earlier products and simplicity of use, including "SORT"**

Using SORT To Do a Copy Operation

❐ **A COPY request may be made by …**

Specifying it on the SORT statement:

```
SORT FIELDS=COPY
```

Or on the OPTION statement:

```
OPTION COPY
```

❐ **When doing a COPY:**

INREC, OUTREC, INCLUDE, OMIT may be used (!)

✗ This could be very useful!

❐ **You can eliminate records with duplicate control key values by including this DFSORT control statement:**

```
SUM   FIELDS=NONE
```

Note that SUM will work for SORT and MERGE operations but not COPY operations

DFSORT Control Statement Coding Rules

❒ **DFSORT control statements must follow these rules:**

Free form in columns 2-71

Operator must be completely specified on one line, as well as the first operand

At least one space before the operator and at least one space between an operator and its first operand

Commas separate operands, with no extra spaces

Continuation is indicated by comma-blank, semicolon-blank, or colon-blank; continuation begins on the next line with the first non-blank character

Operators and operands must be entered in upper case EBCDIC (note: non-numeric literals for operands may be upper case, lower case, or mixed case, as appropriate to the operand)

Operands of the form operand=value may also be coded operand=(value) or operand(value); so these are equivalent:

✗ SORT FIELDS=COPY

✗ SORT FIELDS=(COPY)

✗ SORT FIELDS(COPY)

➢ This is true for all operands for DFSORT for z/OS 1.10 and later, but only some operands for earlier releases

An asterisk in column 1 indicates a comment

Blank lines may appear anywhere

DFSORT Return Codes

❐ **Each DFSORT job step returns one of these values:**

0 - successful completion

4 - successful completion; an RC4 option was set to indicate one of several possible situations (*e.g.*: a summary field overflowed and OVFLO=RC4 was in effect)

16 - unsuccessful completion

20 - message data set missing

A Preview: Some Other DFSORT Capabilities

❐ **You can create multiple output files**

Reformatting records differently for different output files

Splitting the records across different output files

❐ **You can:**

Sort on two digit years (specify a century window through a run-time option and identify the date format from a list of options)

Transform two-digit year dates to four-digit year dates

Sort using locale processing

✗ Allowing DFSORT to be sensitive to collating sequences that differ between languages or cultural conventions

Use a symbol-specifications file pointed at by a SYMNAMES DD statement to sort on field names instead of offsets, lengths, and data type (discussed shortly)

And lots more, stay tuned

Computer Exercise: Running SORTs

In your TR.CNTL library is a member named SORTJOB. This contains the basic JCL for running a sort. For the lab, you will use this as a starting point to build other members that each run a single sort step.

For example, create members with names we'll assign (so we can reference them later) such as SORT01, SORT02, and so on (details on the following pages).

Reminder: the simplest way to create a new member based on an existing one is to start with the member list in edit mode (ISPF 3.4, enter an 'e' next to the library name). Then on the command line enter:

```
s sort01;copy sortjob
```

This will place you in edit of SORT01, ready to modify it for that sort, then to submit the job.

In SORTJOB we use the following DD statement:

```
//SORTIN   DD   DISP=SHR,DSN=&HL..TR.--------
```

For each sort job, simply change the dashes to the low level qualifier of the dsn you need to work with, for example:

```
//SORTIN   DD   DISP=SHR,DSN=&HL..TR.INPUTA
```

to reference your version of the TR.INPUTA data set; or:

```
//SORTIN   DD   DISP=SHR,DSN=&HL..TR.DATA(INPUTK)
```

to reference member INPUTK in your TR.DATA library.

NOTE: For all labs, unless otherwise specified, assume SORTOUT is to go to a print file (SYSOUT).

Computer Exercise: Running SORTs, 2

So here are the member names you should create and the task each separate job should accomplish:

1. SORT01

   Sort the TR.INPUTA inventory file by Description, ascending. Include only records that have a '3', '5', or '7' in the last position of Part Number.

   Format the output lines:

   (two blanks)**Description**(three blanks)**Part Number**(three blanks)**Category**

Take some time to look at the SYSOUT file (Sort messages), try to pick out the most useful information.

2. SORT02A, SORT02B

   Sort the TR.INPUTX inventory file by ascending values in Category. Do one run with EQUALS and one with NOEQUALS (see page 21) and compare the outputs.

   Format the output lines:

   (two blanks)**Category**(three blanks)**Part Number**(three blanks)**Description**

3. SORT03A, SORT03B

   Run SORT02A and SORT02B adding the statement

   ```
            SUM FIELDS=NONE
   ```

   after the SORT command in both cases (see page 28). Compare the results.

4. SORT04

Sort member INPUTG2 in your TR.DATA library into ascending sequence of first name within last name; the output records should contain last name, first name, employee number, and job title.

5. SORT05

Copy member INPUT1 in your TR.DATA library (see p. 27), unchanged.

6. SORT06

SORT member INPUT1 in ascending sequence by the sequence number field, and output just the text.

7. SORT07

Sort TR.ZINPUTA into ascending sequence by Description; only include records where quantity on hand is less than reorder level. Output records should display Description, Part number, quantity on hand, and quantity on order, with 2 spaces between the fields.

8. SORT08

Sort TR.ZINPUTX into ascending sequence by Old Part Number; output records should display Old Part Number, Description, and Unit Price, with spaces between the columns.

9. SORT09

Sort TR.INPUTE in ascending sequence by album title; the output records should include album title and artist name.

Reminder: for variable length records, you must allow for the four byte RDW at the front of the data records.

Note: if your SORTIN file is variable-length, then OUTREC, by default, will also be variable-length, and you must allow for a leading four byte RDW at the front of your OUTREC area.

In preparation for future work (note may not be possible if you are not on a current version of z/OS):

Browse TR.ZASCIIX. On the command line issue ==> **disp ascii**
then End (F3).

Edit TR.ZASCIIX. On the command line issue ==> **source ascii**
Change the first record's part number PART03105 -> PART04105
then End (F3).