JCL

JCL is a Job Control language to submit a job that is executed in the operating system.

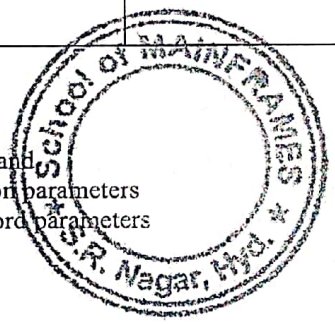
Purpose

JCL is used for the below activities

1. To compile COBOL, COBOL+DB2, COBOL+DB2+CICS programs
2. To execute both user defined and utility programs
3. To supply load libraries of user defined COBOL/COBOL DB2 programs
4. To supply input files and output files that is required for a step

Types of JCLs

Compile JCL	Run JCL
To compile COBOL COBOL + DB2 COBOL + CICS COBOL + DB2 + CICS BMS Maps	To run COBOL program COBOL + DB2 programs Utility programs Combination of all above



Different Fields in JCL

Name	Operation	Operand
Jobname	JOB	Position parameters
Stepname	EXEC	Keyword parameters
DDname	DD	

Coding Rules

Cols

1234----10 72

First two characters should be two forward slashes //

Third character should be name filed or a comment or space for a continuation

Name field	Comment	Continuation
//JOBNAME	//* This is a comment	//JOB1 JOB ACTG(8012T),'SAMPLE JOB',
//STEPNAME		// CLASS=A, PRTY=9,...
//DDNAME		Continuation should start between 4 and 16

Null statement

//

How to code name, operation and operand fields

```
//NAMEFLD OPERATION OPERAND1,OPERAND2,
//  OPERNAD3,OPERAND4...
```

JOB STATEMENT

```
//JobName  JOB  ACTG(xyz),'PROGRAMER NAME',CLASS=A,PRTY=9,
//              MSGCLASS=A,MSGLEVEL=(1,1),TIME=(MM,SS),
//              NOTIFY=USERID,REGION=0M,RESTART=STEPNAME,
//              COND=(0,NE),OUTLIM=8M,TYPERUN=SCAN
```

JobName

It is required to identify this job from other jobs in the SPOOL

1. 1 to 8 characters. Minimum is 1 character and maximum is 8 character
2. 1st character must be alphabet
3. Other characters can be alphabets or numerics or $, @, #

Common naming convention of jobs that are migrated to production

ABCDEFGH

A - Is environment. T/I/M/P

BCD - Is system name. To identify system name from other system jobs

E - Occurrence . D/W/M/Q/Y/S

Daily
Weekly
Monthly
Quarterly
Yearly
Special

FGH - Unique name of the job within the system

Sample jobnames: TACSD010 TACSDASR TL70[illegible]
IACSD010 IACSDASR IL70[illegible]
MACSD010 MACSDASR M[illegible]
PACSD010 PACSDASR P[illegible]

Job names for personal jobs or lab sessions

Userid+1/2 chars

OZA183C1 -- Userid is OZA183

OZA184R1 -- Userid is OZA184

Accounting info

It is a positional parameter and codes it after JOB statement.

It is used to capture the CPU time for the billing purpose.

It is installation defined. Use a ',' if it is not coded

Real time Examples: (8012T)
(8012I)
(8012M)
(8012P)

Lab: OZA

Programmer name

It is used to identify the programmers name. It can be maximum of 20 chars

Though it is a programmer name but it is used to give a brief description of the job.

Ex1: //TACSDASR JOB (99324T),'MADHU PADALA'

Ex2: //TACSDASR JOB (99324T),'AGENT SALES REPORT'

ASR Cannot give brief information.

Class

It is a keyword parameter. It has single character from A-Z
It says about the JOB queue that this job is routed.
It says about maximum time a job can be executed in the class.

Each class has priorities on resources over the other classes
Real time Examples: CLASS=P -- Production class
CLASS=M -- Model/UAT class
CLASS=I -- SIT Class
CLASS=A/B/C -- Test Class
A - Short Running Class
B - Long waiting class
C - Long Running Class
In test region more users submit their jobs, so more classes required.
In SIT/UAT/PROD most of jobs are submitted one after the other.

PRTY

It is a keyword parameter.
It increases the priority when the job is waiting in a Queue.
It has no effect on the jobs that are already executing.
It has a range of values from 1 to 15. 1 is least and 15 is highest.
In most of installations, permission are not given to the users to code it.
Because everyone submits their jobs with highest priority. So the purpose of this parameter is misused.
Session jobs execute with 15 and all other jobs execute with 9.

MSGCLASS

It is a keyword parameter.
It says about the device name where the job output messages are written.
It has valid values from A to Z.
Real time examples: MSGCLASS=X -- Spool
MSGCLASS=L -- SAR
Lab: MSGCLASS=A

MSGLEVEL

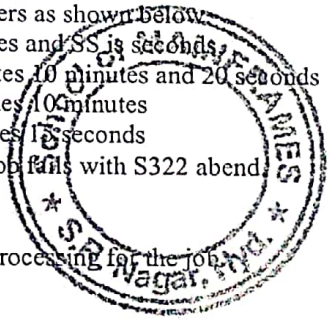
It is a keyword parameter.
It has two positional sub parameters. One is statements and another is Messages.
Ex: MSGLEVEL= (1,1)
Statements indicates which job control statements to be printed
0 The system prints the JOB statement and all comments and JECL statements up to the first EXEC statement.
1 The system prints all JCL statements, JES2 or JES3 control statements, the procedure statements, and IEF653I messages, which give the values assigned to symbolic parameters in the procedure statements.
2 The system prints only JCL statements and JES2 or JES3 control statements

Messages indicates which system messages is to be printed
0 The system prints only JCL messages. It prints JES and operator messages only if the job abnormally terminates, and prints SMS messages only if SMS fails the job.
1 The system prints JCL, JES, operator, and SMS messages.

NOTIFY
It is a key word parameter.
It is required to notify to the user when the job successfully executes or fails.
If it is not coded, then user has to check the status of the job from the spool.
It has two methods.
NOTIFY=USERID . When userid is given then it will notify to the given user.
NOTIFY=&SYSUID. When &SYSUID is given then it will notify to the user whoever submits it.

REGION
It is a keyword parameter.
It allocates address space required to execute a job.
If the address space is not sufficient then job fails with S804 abend.
REGION=8M -- It allocates 8 Million bytes address space.
REGION=0M -- It allocates maximum address space required for the job.

TIME
It is a key word parameter.
It is used to change the default time set by the CLASS
It has two positional sub parameters as shown below.
TIME=(MM,SS) -- MM is minutes and SS is seconds.
Ex: TIME=(10,20) -- Job executes 10 minutes and 20 seconds
TIME=10 -- Job executes 10 minutes
TIME=(,15) -- Job executes 15 seconds
If the time is not sufficient then job fails with S322 abend.



TYPRUN
This parameter requests special processing for the job.
General Syntax
TYPRUN={HOLD/SCAN}
HOLD - Job will held (and not executed temporarily) until the operator uses a command to release.
SCAN – Job will be scanned for all syntactical JCL errors but will not execute.
In many installations, the below commands are used to find the syntactical JCL errors. With these commands errors can be checked without submitting JCL.
!JCK
JSCAN
JEM

Restart
It is used to start the job from the required step of the job. I.e. Skip the initial steps.

Method1:
RESTART=JOBSTEP → Execute the job from the jobstep

```
//JOB1 JOB (8012T),'MADHU PADALA',RESTART=JS30
//JS10  EXEC PGM=PGM1
//JS20  EXEC PGM=PGM2
//JS30  EXEC PGM=PGM3
//JS40  EXEC PGM=PGM4
```

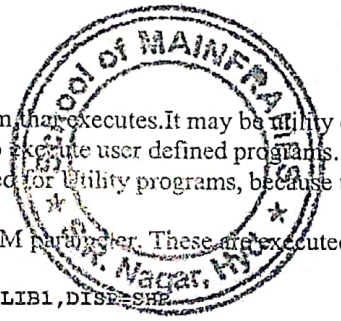
It will restart the job from JS30

Method2:
RESTART=JOBSTEP.PROCSTEP -- Execute the job from the jobstep

```
//JOB1 JOB (8012T),'MADHU PADALA',
//       RESTART=JS10.PS20
//JOBLIB DD DSN=USERID.LOADLIB,DISP=SHR
//*
//PROCCALL JCLLIB ORDER=USERID.PROCLIB1
//*
//JS10  EXEC PROC1
//    ENV=T
```

```
//PROC1 PROC ENV=T
//*
//PS10    EXEC PGM=PGM1
//INFILE  DD DSN=&ENV.ABC.PSEQ.FILE1,DISP=SHR
//OUTFILE DD DSN=&ENV.ABC.PSEQ.FILE1,
//           DISP=(NEW,CATLG,DELETE),
//                    ..
//PS20    EXEC PGM=PGM2
//INFILE DD DSN=&ENV.ABC.PSEQ.FILE2,DISP=SHR
//OUTFILE DD DSN=&ENV.ABC.PSEQ.FILE3,
//           DISP=(NEW,CATLG,DELETE),
//           ..
//PS30    EXEC PGM=PGM3
//INFILE DD DSN=&ENV.ABC.PSEQ.FILE3,DISP=SHR
//OUTFILE DD DSN=&ENV.ABC.PSEQ.FILE4,
//           DISP=(NEW,CATLG,DELETE),
//
```

EXEC STATEMENT



PGM

Syntax: PGM=programname
It is used to tell about the program that executes.It may be utility or user defined program.
Loadlib or Steplib are required to execute user defined programs.
Loadlib or Steplib are not required for utility programs, because these are executed from the system library.
Subprograms never coded on PGM parameter. These are executed as a part of main program.
Ex1:

```
//JOBLIB DD DSN=USERID.LOADLIB1,DISP=SHR
//STEP1  EXEC PGM=PGM1
```

Ex2:

```
//STEP1   EXEC PGM=PGM1
//STEPLIB DD DSN=USERID.LOADLIB1,DISP=SHR
```

Ex3:

```
//STEP1   EXEC PGM=SORT
```

PARM

Syntax: PARM='value' or PARM=(value)
It is a keyword parameter. It is used to pass control data from JCL to program. Maximum of 100 characters can be passed from JCL to cobol program.
Different parm values can be passed from JCL to program in different runs.
Without changing the program, the program can be controlled from JCL.

Example:
JCL to execute PGM1

```
//STEP1   EXEC  PGM=PGM1,PARM='IND'
```

Code in PGM1

```
Linkage-section.
01 ls-var.
     05 ls-len         pic s9(4) comp.
      05 ls-country    pic x(3).
procedure division using ls-var.
            .
            .
         Read emp-file.
         if emp-country = ls-country
            perfrom process-data.
```

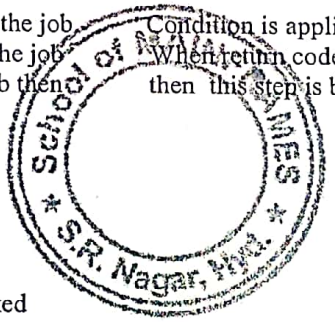
If the PARM='USA' is used, then it will process file for USA country code.

Time, Cond, Region parameters

The above parameters can be coded both in Job and Step Level

Cond Parameter

Job Level	Step Level
Condition applied for all steps of the job. When return code of any step in the job is true with condition code on Job then all other steps are bypassed	Condition is applied only on single step. When return code of prior step is true then this step is bypassed



Syntax:

Method1

COND=(CC,RO,STEPNAME)
CC -- Condition code to be checked
RO -- Relation operation
Stepname -- Return code of Prior step Name

```
//STEP1  EXEC PGM=PGM1
//STEP2  EXEC PGM=PGM2,COND=(4,LT,STEP1)
```

Step2 executes only when Step1 is returning <= 4

Method2

COND=(CC,RO)
Stepname is optional. When omitted, it takes maximum return code of the prior steps.

```
//STEP1  EXEC PGM=PGM1
//STEP2  EXEC PGM=PGM2
//STEP3  EXEC PGM=PGM3
//STEP4  EXEC PGM=PGM4,COND=(0,NE)
```

Step4 executes when maximum return code of the prior steps is zero.

Method3

COND=EVEN/ONLY

It is used to execute steps based on abends on prior steps. It don't depend on Return codes.

COND=EVEN It executes even if the prior step abends

COND=ONLY It executes only of the prior step abends

These are used at step level but not job level.

System Abends S122 and S222 has no effect on COND=ONLY or COND=EVEN

Important points:

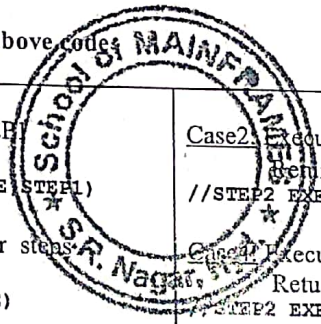
JOBLIB with COND=ONLY

If the job contains a JOBLIB DD statement and ONLY is specified in a job step, the JOBLIB unit and volume information are not passed to the next step; when the next step is executed, the system searches the catalog for the JOBLIB data set.

Few examples on condition code

```
//JOB1 JOB .....
//..
//STEP1 EXEC PGM=PGM1
//STEP2 EXEC PGM=PGM2
//STEP3 EXEC PGM=PGM3
```

Condition at step level for the above codes



Case1: Execute STEP2 only when STEP1 Returns zero.

```
//STEP2 EXEC PGM=PGM2,COND=(0,NE,STEP1)
```

Case2: Execute STEP2 only when STEP1 returns <= 4

```
//STEP2 EXEC PGM=PGM2,COND=(4,LT,STEP1)
```

Case3: Execute STEP3 only when prior steps Return zero.

```
//STEP3 EXEC PGM=PGM3,COND=(0,NE)
```

Case4: Execute STEP2 only when STEP1 Returns >= 4

```
//STEP2 EXEC PGM=PGM2,COND=(4,GT,STEP1)
```

Case5: Never execute STEP3

```
//STEP3 EXEC PGM=PGM3,COND=(0,LE)
      OR
//STEP3 EXEC PGM=PGM3,COND=(4095,GE)
```

Case6: Execute STEP3 only when prior steps abend

```
//STEP3 EXEC PGM=PGM3,COND=ONLY
```

Case7: Execute STEP3 when prior steps abends or not

```
//STEP3 EXEC PGM=PGM3,COND=EVEN
```

Condition at job level for the above code

Case1: Execute job when prior step returns <= 4

```
//JOB1  JOB .....,COND=(4,LT)
```

Case2: Execute only STEP2

```
//JOB1  JOB .....,RESTART=STEP2,COND=(0,LE)
```

Condition at both job level and step level

```
//JOB1 JOB ...,COND=(4,LT)
//..
//STEP1 EXEC PGM=PGM1
//STEP2 EXEC PGM=PGM2,COND=(0,NE)
//STEP3 EXEC PGM=PGM3
```

Case1: If step1 returns 4
Condition on step2 overrides condition on JOB so STEP2 bypassed

Case2: If step1 returns 8
Condition on step2 overrides condition on JOB but job is stopped before condition on step2 is checked.

Conclusion
Condition on step overrides condition on Job. But condition on JOB is checked before condition on step is checked. If a job level condition is true then all successive steps are bypassed.

Q: There are 6 steps in a JCL. Execute 4, 5, 6 only when first 3 steps executes with 0.

```
//JOB1 JOB .....
//STEP1 EXEC PGM=PGM1
//STEP2 EXEC PGM=PGM2
//STEP3 EXEC PGM=PGM3
//STEP4 EXEC PGM=PGM4,COND=(0,NE)
//STEP5 EXEC PGM=PGM5,COND=((0,NE,STEP1),(0,NE,STEP2),(0,NE,STEP3))
//STEP6 EXEC PGM=PGM6,COND=((0,NE,STEP1),(0,NE,STEP2),(0,NE,STEP3))
```


Time Parameter

```
//JOB1  JOB .........,TIME=3
//        .
//        .
//STEP1 EXEC PGM=PGM1,TIME=2
//STEP2 EXEC PGM=PGM2,TIME=2
```

Case1: If step1 executes more than 2 min

Job abends at STEP1 with S322

Case2: If step1 executes 2 min and step3 exceeding 1 min

Job abends at STEP2 with S322

Conclusion: Time at step overrides Job but overall time is considered from JOB

Region Parameter

```
//JOB1  JOB .........,REGION=8M
//        .
//        .
//STEP1 EXEC PGM=PGM1,REGION=6M
//STEP2 EXEC PGM=PGM2
```

Both STEP1 and STEP2 takes 8M address space.

DD STATEMENT

It is important operation in JCL.
It is used to provide
1. Input and output files to the program given in EXEC statement.
2. System defined ddnames which prints output messages to the spool.
   SYSOUT, SYSPRINT, SYSDUMP, SYSABEND
3. loadlibraries to execute user defined programs with Joblib/Steplib.
4. Instream data to the user defined programs with SYSIN.
5. Control information to the utility programs with SYSIN.

Syntax:
//ddname DD operand1,operand2....

Ex1: For input files
```
//INFILE  DD DSN=FILE1,DISP=SHR
```

Ex2: For output files
```
//OUTFILE  DD DSN=FILE2,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=SYSDA,
//            SPACE=(CYL,(5,2),RLSE),
//            DCB=(RECFM=FB,LRECL=50,BLKSIZE=0,DSORG=PS)
```

Different operands in DD statement:
DSN -- To provide the dataset name
DISP -- To provide the status of the dataset
UNIT -- To provide device name where the dataset is created
VOL=SER -- To provide volume and serial for the tape datasets
SPACE -- To provide the space required for creating disk datasets
DCB -- To provide the attributes of dataset

DSN
It is a keyword parameter. It is used to provide the dataset name
Syntax: DSN=AAA.BBB.CCC -- It is dataset name. maximum Len of name is 40

E.g.1 `//DDDNAME DD DSN=OZA183.SMF.EMPFILE`
Identifies file `OZA183.SMF.EMPFILE`

E.g.2 `//DDNAME DSN=OZA183.SMF.PDS(MEM1)`
Identifies a member of a PDS

E.g.3 `//DDNAME  DD DSN=&&TEMP`
It is a temporary dataset that is created in a step and passed to the other steps of the JCL. It is deleted on normal or abnormal completion of JCL
When a physical existence of the file is not required to pass the data from one step of JCL to another step of the JCL then temporary dataset is used.

E.g.4 `//DDNAME  DD`
If the DSN name is omitted from a DD statement (except DD * , SYSOUT and DUMMY) also indicates a temporary dataset.

E.g.5 `//DDNAME DD DSN=NULLFILE`
`//DDNAME DD DUMMY`

If you don't want to supply a file to the program then use DUMMY or NULLFILE. If it is coded then all the I/O statements against this file are ignored.

DISP

It is a keyword parameter. It has 3 sub positional parameters

Syntax: DISP=(status,normal-termination,abnormal-termination)

Status -- It says about the initial status of the dataset
Normal-termination -- Action on normal completion
abnormal-termination -- Action on abnormal completion

Possible values on these three sub positional parameters.

Status	Normal-Termination	Abnormal-Termination
NEW	CATLG	CATLG
SHR	UNCATLG	UNCATLG
OLD	DELETE	DELETE
MOD	KEEP	KEEP
	PASS	

NEW -- File is not existing, create it.
SHR -- File is existing, use a dataset in a share mode. Others can use this dataset in SHR mode.
OLD -- Use a dataset in exclusive mode. Others cannot use this dataset.
MOD -- File may or may not be existing.
CATLG -- Catalog dataset in system cataloged entries.
UNCATLG -- Uncatalog dataset from the system cataloged entries. But the file is not deleted. If it is to be used then use volume and serial information of the file.
DELETE -- Delete system cataloged entry and also the file.
KEEP -- Keep the dataset in the same cataloged entry along with volume and serial.
PASS -- It is used to pass the dataset to the subsequent steps of the JCL. It is used on datasets which are created but not cataloged. i.e. for the temporary datasets.

Default disp parameters

If DISP is not coded -- It takes DISP=(NEW,DELETE,DELETE)
DISP=(NEW) -- It takes DISP=(NEW,DELETE,DELETE)
DISP=(NEW,CATLG) -- It takes DISP=(NEW,CATLG,CATLG)
If the third sub parameter is not coded then it takes second parameter as third parameter.
DISP=SHR/OLD -- It takes DISP=(SHR/OLD,KEEP,KEEP)
It keeps the dataset on the same volume and serial
DISP=(NEW,PASS) -- It takes DISP=(NEW,PASS,DELETE)
It is used on temporary datasets. Pass is not allowed in the abnormal- termination with that reason it takes Delete.

Disp for input files

DISP=SHR -- File should be existing and other users can also use it in SHR
DISP=OLD -- File should be existing and use it in exclusive mode.
Others cannot use it in SHR/OLD mode.
DISP=(OLD,DELETE,KEEP) -- File should be existing and use it in exclusive mode.
-- Delete it on normal completion and keep it on abnormal completion.

Disp for Output files

DISP=(NEW,CATLG,DELETE) -- Create a new file, catalog it on normal termination
delete it on abnormal termination.
-- It is a common DISP parameters used for output

DISP=(MOD,CATLG,DELETE) -- If the file is existing then appends data
-- If the file is not existing then create it
-- On normal completion catalog it
-- On abnormal completion delete it

DISP=SHR/OLD -- It deletes existing data and writes new data

Disp on files to update

DISP=SHR/OLD -- Update is done only on KSDS VSAM files
-- Existing data can be read, write, update, delete

Disp on Temporary output files

DISP=(NEW,PASS) -- It takes DISP=(NEW,PASS,DELETE)
-- Temporary files are not cataloged.
-- These are created and passed to the next steps in the same JCL.

Disp on Temporary input files

DISP=(OLD,DELETE) -- Use temporary datasets created by the previous steps of the JCL
-- Delete it after using

DISP=(OLD,PASS) -- Use temporary datasets created by the previous steps of the JCL
-- Pass it to next steps of the JCL

Possible Disp parameters for VSAM files

DISP=SHR/OLD -- Existing VSAM datasets can be used in any step
-- These cannot be created with disp parameter
-- These files should be created with IDCAMS utility

Disp to delete output files before these are created

DISP=(MOD,DELETE,DELETE) -- If the file is existing then opens in append mode
otherwise it creates it.
-- File is deleted after use.

Ex: Delete a file before it is created.

```
//step1   EXEC PGM=IEFBR14          → It is a NULL program
//dd1     DD  DSN=FILE2,DISP=(MOD,DELETE,DELETE)
//step2   EXEC PGM=PGM1
//infile  DD  DSN=FILE1,DISP=SHR
//outfile DD  DSN=FILE2,DISP=(NEW,CATLG,DELETE),
//            UNIT=SYSDA,
//            SPACE=(CYL,(10,5),RLSE),
//            DCB=(RECFM=FB,LRECL=50,BLKSIZE=0)
```

UNIT

It is a keyword parameter. It says about where the dataset is created.

Syntax: UNIT=SYSDA -- Disk
UNIT=TAPE/CART -- Tape or cartridge

Space is required for the Disk file.

Ex:
```
UNIT=SYSDA,SPACE=(CYL,(10,5),RLSE)
UNIT=SYSALLDA,SPACE=(CYL,(10,5),RLSE)
UNIT=3390,SPACE=(CYL,(10,5),RLSE)
```

VOL=SER

It is used for the tape files.

Volume and serial are required for TAPE/CART files. Huge files file are written on the tape. Tape files cannot be viewed or edited in ISPF 3.4.These can be processed only in JCL. If you want to browse data of tape files, then copy into disk file and then browse it.

Ex:
```
UNIT=TAPE,VOL=SER=(,,,9)
UNIT=CART,VOL=SER=(,,,9)
```

SPACE

It is a keyword parameter. It has sub positional parameters. Space parameter is required on output files that are created on DISK.

Syntax: SPACE=(Unit-size,(primary-space,secondary-space,directory-blocks),rlse,contig)
Unit-size -- CYL/TRK
Primary-size -- numeric value -- Initially System allocates primary space.
Secondary-size -- numeric value -- If the primary space is not sufficient, then system will create 15 extents of secondary space.
Directory-blocks -- Numeric value -- It is required only for PDS. It tells about the maximum number of members required.
maximum number of members = (num of directory blocks * 6) - 1
Rlse -- Release excess memory that is allocated but not utilized.
Contig -- Allocate continuous memory in the tracks.

Ex1:
```
SPACE=(CYL,(10),RLSE)
```
It allocates maximum of 10 cyls. If it is not sufficient then it abends with SD37.
Code secondary space if it abends.

Ex2:
```
SPACE=(CYL,(10,5),RLSE)
```
It allocates maximum of 85 cyls. If it is not sufficient then it abends with SB37.
Increase primary and secondary space.
It allocates 10 cylinders of primary space. If it is not sufficient then it allocates 5 cylinders of secondary space and it allocates 15 times.Maximum space allocated = 10 + 15 * 5 = 85 cylinders.
`SPACE=(CYL,(10,5),RLSE)` -- It allocates maximum of 85 cyls

Ex3:
```
SPACE=(CYL,(10,5,10),RLSE)
```
It allocates maximum of 85 cyls.Maximum 59 members can be created.
If the above space is not sufficient then it abends with SE37.If it abends then first compress dataset by using Z infront of the PDS.
In general, PDS is not created with BATCH JCL. It is created with ISPF 3.2.

DCB
It is a keyword parameter. It has sub positional key word parameters.
Syntax: DCB=(RECFM=Record-format,LRECL=number,BLKSIZE=number/0,DSORG=PS/PO)

RECFM sub parameter
Record-format -- F/FB/V/VB/U
F -- Fixed length.
FB -- Fixed length blocked. I-O device reads block full of records at a time and keeps it into main memory. It improves the performance of I-O operation.
In both F and FB, every record of the file has same length.
On F, I-O device reads one record and returns to program.
On FB, it reads block full of records and are kept in main memory. It improves the performance of the I-O operations.LRECL in JCL = record length in COBOL

V -- Variable length. I-O device reads one record at a time.
VB -- Variable length blocked-O device reads block full of records at a time and keeps it into main memory. It improves the performance of I-O operation
In both V and VB, each record may have different length.
On V, I-O device reads one record and returns to program.
On FB, it reads block full of records and are kept in main memory. It improves the performance of the I-O operations. LRECL in JCL = Length of longest record in COBOL + 4 .
U -- Undefined length.

LRECL sub parameter
It is logical record length.

BLKSIZE sub parameter
BLKSIZE=Lrecl * n -- If it is coded then code it multiples of LRECL
BLKSIZE=0 -- If zero is coded then system determine the best blksize. Usually it is 32760.
It is always recommended to code BLKSIZE=0

DSORG sub parameter
It says about the dataset type.
DSORG=PS/PO -- PS is used for physical sequential files
-- PO is used for PDS
-- If it is omitted it takes PS.

How to supply a dummy dataset
//ddname dd DUMMY
If DUMMY is used, then other operands should not be coded. All I-O operations on this DDNAME are ignored.

Job stream input
To include input data as part of a job stream, code the following

```
//DDNAME   DD  *
 .      (actual data/source data set)
 .
 .
/*
//    →  The NULL statement is the default end of input data
```

• **DATA parameter**

DATA parameter should be specified if you want to include // in the input data stream

```
//DDNAME   DD DATA
   .
   .
//
/*
```

The last statement specifies the end of input data and should be coded without fail.

• **DLIM parameter**

When you want to enter /* in the input stream,then code a delimiter statement to indicate the end of a input stream.

```
//DDNAME   DD *,DLM=XX   or //DDNAME   DD   DATA,DLM=XX
   .
   .
//XX
```

Printer statements

```
//DDNAME   DD    SYSOUT=CLASS,OUTLIM=5000,COPIES=2
```

The sysout parameter is used to direct output to a printer

When * is coded for sysout class(backward reference),the dataset is automatically sent to the same output class as in the MSGCLASS parameter of the JOB statement

OUTLIM parameter controls the number of lines printed in the output.

Concatenation of datasets

Data set concatenation allows several physical datasets to be treated as a single logical dataset for input.

Concatenated datasets are requested by coding a DD statement for each dataset to be concatenated,and placing the DD statements in the order in which the datasets are to be processed

Only the first DD statement has a ddname.The datasets that are to be concatenated should have same DCB parameters

Ex:

```
//JOBLIB DD DSN=ENDVR.TEST.ACS.LOADLIB,DISP=SHR
//       DD DSN=ENDVR.INTG.ACS.LOADLIB,DISP=SHR
//       DD DSN=ENDVR.MODL.ACS.LOADLIB,DISP=SHR
//       DD DSN=ENDVR.PROD.ACS.LOADLIB,DISP=SHR
```

In the above example, Load libraries of Test, Integration, UAT/Model and Production environments are concatenated.

1.16 PDS OR 255 sequential datasets can be concatenated.

2 .Lrecl and record format should be same.

3. If the block size is different, then largest block size dataset should be first.

4. Datasets may reside on different devices and device types.

Special ddnames

SYSOUT
Syntax:
```
//SYSOUT    DD   SYSOUT=*
```
This ddname is required to print the output messages of the program. Display statements given in the program are routed SYSOUT. Brief information of the user abends are routed to SYSOUT

SYSIN
Syntax:
```
//SYSIN  DD *
 Data
/*
```
It is used to pass the instream data to the cobol program. One line is accepted to one accept statement. Maximum of 80 characters can be passed in a single line. It is also used to pass the control information to utility program.

How to pass data from SYSIN to COBOL
Code in JCL
```
//STEP1   EXEC PGM=PGM1
          .
          .
 //SYSIN  DD *
 20120129
/*
```
In program
```
01 ws-date    pic x(8).
procedure division.
     .
     .
          ACCEPT WS-DATE.
```

SYSPRINT
Syntax: `//SYSPRINT    DD   SYSOUT=*`
It is used for utility programs to route utility execution or error messages.

SYSABEND:
Syntax: `//SYSABEND    DD   SYSOUT=*`
This system dd name identifies the dataset for formatted dump of control blocks and program, used if the program fails.

SYSDUMP
Syntax: `//SYSDUMP    DD   SYSOUT=*`
It is used to route dump messages on abends.

JOBLIB: It follows the job statement.loadmodules of any steps(EXEC) that don't have respective steplib will be looked into this PDS. If not found, it will be checked against system libraries. if it is not found there also, then the job would abend with s806. It is only DDNAME used without a EXEC statement.

STEPLIB: It follows exec statement.loadmodules will be checked first in this library and then in the system libraries. if it is not found in both places, then the job would abend with S806 code.

Referback

Referback permits to obtain information from previous JCL statements in the job stream. It can be used for DSN, DCB, PGM parameters. '*' is a referback operator.

Referback on PGM.

```
//STEP2     EXEC     PGM=*.STEP1
```

Referback on DD statement

It has three formats

1. Referback dataset name of a DDNAME from the prior step.

*.stepname.ddname

Ex :`DSN=*.STEP1.DD1`

2. Referback dataset name of another DDNAME from the same step.

*.ddname

Ex :`DSN=*.DD1`

3. Referback dataset name of a DDNAME from the prior step of prior proc.

*.procstep-in-JCL.stepname-in-PROC.ddname

Ex :`DSN=*.js10.step1.dd1`

Same as DSN, DCB can also be coded in three different formats.

Note: Referback is not recommended because it reduces readability.

System abends

SOC7

What are the situations to soc7?

1. Uninitialized numeric items (mostly comp-3 items) in the working storage.
2. Non numeric data in the file is mapped to a numeric data item and it is used in either MOVE or COMPUTE statements.

It is most common abend in the production support

How to find statement and record?

1. How to find statement?
   1. Take the offset from SYSOUT of RUNJCL in spool
   2. Find the offset in the SYSPRINT of compile job in the spool. Program should be compiled With compile parameters LIST or OFFSET
   3. See the line number and verb that is having this offset number
   4. Find the line number in the same compile listing from the top. That line will have statement that is causing the SOC7.

2. How to find the record?
   1. Change the program to put a display statement after reading the input file, compile and run it. This time it will abend again with SOC7 at same location and record.

Now find the last record that is displayed in the spool. That is the record causing the SOC7.

Solution

1. correcting the program
   a) If SOC7 is with un initialized comp-3 variables, then initialize it in the working storage or initialize it in the procedure division before using it in the computation or move statement
   b) If it is from file ,then use the business rule whether to write the record into a error file or move zeros to non-numeric data by validating 'IS NUMERIC' class condition.

2. Skip the bad record from input file and restart the program.
   a) Find the current record from the AbendAid and skip the record from the input file. i.e by copying all other records except the abend causing record into a TSO file then override TSO file and then restart the job.
   b) Send the skipped record to the onsite team.

If this program is abending frequently then it is suggested to for a permanent solution by fixing the program. But it is a user's decision on how to fix it.

Note: If the job abends with S0C7 in the production then oncall programmer (person who is working in the production support) will go the with the second solution.

SOC1

1. Misspelled DD name in RUN JCL
   DD name not defined in RUN JCL
2. When a correct processor group is not supplied while adding it to the endevor.
3. Subprogram called dynamically and it is not found in the supplied load libraries.

Solution:

1. Correct the DD name
2. Add it with correct processor group
3. Supply the correct load libraries. Most of the cases we forget to give production load libraries after the personal load libraries.

SOC4

1. Trying to refer an element of occurs beyond max limitation and the program is compiled with SSRANGE.
2. Trying to open a VSAM file which is not available.
3. Trying to open a VSAM file in I-O mode which is created and no records are written into it.

Solution

1. Correct the program and make sure that it is not going beyond the maximum occurrence.
2. Make sure that the VSAM file is available in the JCL.
3. Write at least one record just before use it.
   When a VSAM file is created in the step1 and it is used in the step2 in I-O mode, then insert one more step in between step1 and step2 to load a sequential file that has two records one with low values and one with high values. Most of the installations use this solution.

S322

Time out abend

Solution:

1. Check the class parameter, if it is a short running class then change it to long running class
2. If it is already a long running class, then check whether it is in infinite loop or not.
   a) If it is in infinite loop then fix the program and rerun it.
   b) If it is not in infinite loop then give the time parameter to 1440.

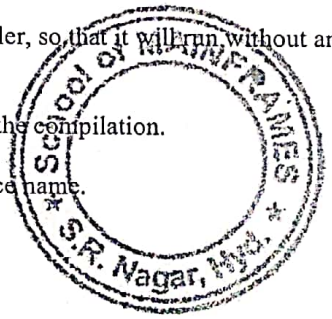
How to find the job is in infinite loop or not.
While job is running, Go to DA in the spool and the see the CPU % and Exception Count
If a job is running in the infinite loop then it takes more CPU and Exception count is not going to be changed.

For the confirmation debug the program either with XPEDITER or with display statements.

S806
Load module not found in the load library
Possible reasons:
1. Load library is not supplied in JOBLIB or STEPLIB
2. Main program is not compiled
3. STEPLIB is given and the load module is not found in the STEPLIB, but it is found in the JOBLIB. when a STEPLIB is given, system will not check the load module in the JOBLIB.

Solution
Supply load module (or) compile the program to correct load library

S913
Authorization failure to read or update a dataset.
Solution:
Run the job by the ZEKE scheduler, so that it will run without any problem.



S013
Source program is not found for the compilation.
Solution
Give the correct library and source name.

Space Abends

SE37
Space not available for new member in PDS
Solution
1. Compress the PDS with line command Z. If it is still abending go for the second solution.
2. Create a new PDS more cylinders and more directory blocks, move all elements from existing to new PDS,delete old PDS and rename the new PDS to old PDS.

SB37
primary, secondary are declared but not sufficient
Solution
Increase primary and secondary appropriately.

SD37
No secondary space is given
Solution
Give secondary cylinders.

Procedures

A procedure is a set of standard JOB step definitions that can be used repeatedly to perform a FUNCTION. Procedures are invoked by EXEC statements

Rules:
Procedures begin with proc or EXEC statements.
Only statements allowed in a procedure are EXEC, DD & comment.
Cannot have instream data statements in a procedure.
A procedure cannot invoke another procedure.
Cannot contain joblib&jobcatDDnames

Types of PROCS

Instream Proc	Cataloged Proc
1. Proc is coded within the JCL	1. Proc is coded in a separate member
2. It can be executed from the same jcl where it is coded	2. It can be executed from any JCL
3. It has PROC and PEND	3. It has PROC but PEND is not required
4. It is rarely used in personal JCLs	4. It is almost used in every JCL that is executed from different environment
5. First two chars of every statement in proc is expanded as ++ in JESJCL	5. It is expanded as XX
6. Override on any statement expands as +/ in JESJCL	6. On overrides, It is expanded as X/
7. It can be executed 15 times from the same JCL	7. There is no such a limitation

Ex: Instream proc and its execution
Note: It can be executed 15 times from the same JCL but it cannot be executed from different JCL.

```
//JCL1  JOB ....
//..
//PROC1 PROC
//PS10 EXEC PGM=PGM1
//INFILE  DD DSN=&ENV.ACS.PSEQ.FILE1,
//           DISP=SHR
//OUTFILE DD DSN=&ENV.ACS.PSEQ.FILE2,
//           DISP=(NEW,CATLG,DELETE),
//..
//..
//       PEND
//*
//JS10 EXEC PROC1,
//          ENV=T
```

Ex: Cataloged proc and its JCLs from different environments

Note: When a new JCL is required in production environment, then create 4 JCLs.

1. Test JCL 2. Integration JCL 3. Model JCL 4. Prod JCL

In general, all steps are coded into a cataloged proc with symbolic parameter for High level qualifier of the file name. Then execute this proc from above 4 JCLs.

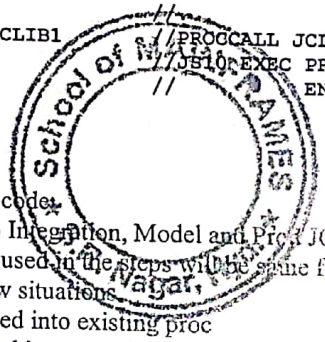
```
//PROC1 PROC ENV=
//PS10 EXEC PGM=PGM1
//INFILE  DD DSN=&ENV.ACS.PSEQ.FILE1,
//           DISP=SHR
//OUTFILE DD DSN=&ENV.ACS.PSEQ.FILE2,
//           DISP=(NEW,CATLG,DELETE),
//..
//..
```

```
//TJCL
//...
//PROCCALL JCLLIB ORDER=PROCLIB1
//JS10 EXEC PROC1,
//          ENV=T
```

```
//IJCL
//...
//PROCCALL JCLLIB ORDER=PROCLIB1
//JS10 EXEC PROC1,
//          ENV=I
```

```
//MJCL
//...
//PROCCALL JCLLIB ORDER=PROCLIB1
//JS10 EXEC PROC1,
//          ENV=M
```

```
//PJCL
//...
//PROCCALL JCLLIB ORDER=PROCLIB1
//JS10 EXEC PROC1,
//          ENV=P
```



Advantage of PROCs

1. Reduced coding with reusable code.
2. JCL errors are not promoted to Integration, Model and Prod JCLs
3. Number of steps and ddnames used in the steps will be same for all JCLs
4. Reduced maintenance on below situations
   a. When a new step is introduced into existing proc
   b. When a new file is introduced into an existing step
   c. When an existing file attributes are changed

   In all the above cases PROC is impacted but not the JCLS.

PROC is used to execute steps from the different JCLs from different environments.

Proc Overrides

EXEC operands	DD operands
PGM	DSN
PARM	DISP
TIME	UNIT
REGION	SPACE
COND	DCB

Cond Override

1. Make a step not to execute
2. Change the existing cond

DD override

1. Supply prod file as input instead of test file
2. Change space/DCB parameters of an existing file
3. Override 2nd file of 3 concatenated file
4. Make a file dummy
5. Supply a new DDNAME to an existing step

Test JCL

```
//JOB1   JOB ......
//..
//PROCCALL JCLLIB ORDER=(PROCLIB1,
//                       PROCLIB2)
//
//JS10 EXEC PROC1,
//        ENV=T
```

PROC

```
//PROC1 PROC ENV=
//*
//PS10     EXEC PGM=PGM1,PARM=(IND)
//INFILE    DD DSN=&ENV.ACS.PSEQ.FILE1,DISP=SHR
//OUTFILE   DD DSN=&ENV.ACS.PSEQ.FILE2,
//             UNIT=SYSALLDA,
//             DCB=(RECFM=FB,LRECL=22,BLKSIZE=0)
//             SPACE=(CYL,(10,5),RLSE)
//PS20     EXEC PGM=PGM2
//INFILE    DD DSN=&ENV.ACS.PSEQ.FILE2,DISP=SHR
//          DD DSN=&ENV.ACS.PSEQ.FILE2A,DISP=SHR
//          DD DSN=&ENV.ACS.PSEQ.FILE2B,DISP=SHR
//OUTFILE   DD DSN=&ENV.ACS.PSEQ.FILE3,
//             DISP=(NEW,CATLG,DELETE),
//..
//*
//PS30     EXEC PGM=PGM3
//INFILE    DD DSN=&ENV.ACS.PSEQ.FILE3,DISP=SHR
//OUTFILE   DD DSN=&ENV.ACS.PSEQ.FILE4,
//             DISP=(NEW,CATLG,DELETE),
//..
//*
//PS40     EXEC PGM=PGM4
//INFILE    DD DSN=&ENV.ACS.PSEQ.FILE4,DISP=SHR
//OUTFILE   DD DSN=&ENV.ACS.PSEQ.FILE5,
//             DISP=(NEW,CATLG,DELETE),
//..
//*
//PS50     EXEC PGM=PGM5
//INFILE    DD DSN=&ENV.ACS.PSEQ.FILE5,DISP=SHR
//OUTFILE   DD DSN=&ENV.ACS.PSEQ.FILE6,
//             DISP=(NEW,CATLG,DELETE),
//..
```

Override operands on EXEC statement

Case1: Make PS40 never execute

```
//JS10    EXEC PROC1,
//   COND.PS40=(0,LE),
//             ENV=T
```

Case2: Change PARM in PS10 to USA

```
//JS10    EXEC PROC1,
//   PARM.PS10=(USA),
//             ENV=T
```

Case3: Add TIME=10 in PS10

```
//JS10    EXEC PROC1,
//   TIME.PS10=10,
//             ENV=T
```

Case4: Restart job from PS30 of PROC1

```
//JOB1   JOB ....RESTART=JS10.PS30
//..
//JS10    EXEC PROC1,
//             ENV=T
```

Case5: Execute PS30 only

```
//JOB1   JOB ....RESTART=JS10.PS30,
//               COND=(0,LE)
//..
//JS10    EXEC PROC1,
//             ENV=T
```

Case6: Execute PS30 and PS50 only

```
//JOB1   JOB ....RESTART=JS10.PS30
//..
//JS10    EXEC PROC1,
//   COND.PS40=(0,LE),
//             ENV=T
```

Override operands on DD statement

Case1: Override INFILE of PS10 with production file

```
//JS10    EXEC PROC1,
//             ENV=T
//PS10.INFILE DD DSN=PACS.PSEQ.FILE1,
//               DISP=SHR
```

Case2: Override second concatenated file in INFILE of PS20 with production file

```
//JS10    EXEC PROC1,
//             ENV=T
//PS20.INFILE DD
//            DD DSN=PACS.PSEQ.FILE2A,DISP=SHR
//            DD
```

Case3: Override OUTFILE of PS50 with DUMMY

```
//JS10    EXEC PROC1,
//             ENV=T
//PS50.OUTFILE DD DUMMY
```

Case4: Supply a new DDNAME INFILE1 in PS10

```
//JS10    EXEC PROC1,
//             ENV=T
//PS10.INFILE1 DD DSN=&ENV.ACS.PSEQ.FILE1A,
//               DISP=SHR
```

Case5: Concatenate one more file to INFILE of PS30

```
//JS10    EXEC PROC1,
//             ENV=T
//PS30.INFILE DD
//            DD DSN=&ENV.ACS.FILE3A,
//               DISP=SHR
```

Case6: Override space parameter to increase space of OUTFILE in PS10

```
//JS10    EXEC PROC1,
//             ENV=T
//PS10.OUTFILE DD  SPACE=(CYL,(50,25),RLSE)
```

GDG (Generation Data Group)

Q: Why is GDG required?

A: When a back dated files are to be retained then use GDG

Q: What are different real situations where GDG is required?

A: Daily Transaction files
Daily backup files
Monthly backup files
Interface files

Q: Steps to use GDG?

A: 1. Create GDG Base with IDCAMS in Batch or in a foreground with 3.2 option in File Aid
2. Create/use the generations in any JCL

Q: How to create GDG?

A:
```
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSIN     DD *
  DEFINE GDG (NAME(USERID.FILE1.BKP)
              LIMIT(N)
              NOEMPTY/EMPTY
              SCRATCH/NOSCRATCH)
/*
```

N is number from 1 to 255
NOEMPTY - Uncataloged oldest generation when a new generation is created and limit is already reached
EMPTY - Uncataloged all generations for the above situation
SCRATCH - Delete that uncataloged generation
NOSCRATCH - Do not delete that uncataloged generation
Common used options are NOEMPTY and SCRATCH

Q: How to create a new generation of a GDG?

A:
```
//STEP1 EXEC PGM=IEBGENER
//SYSPRINT  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSUT1    DD DSN=USERID.FILE1,DISP=SHR
//SYSUT2    DD DSN=USERID.FILE1.BKP(+1),
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,
//             SPACE=(CYL,(10,5),RLSE),
//             DCB=(RECFM=FB,LRECL=100,BLKSIZE=0)
//SYSIN     DD DUMMY
```

Q: How to use current generation of a GDG as input?

A:
```
//INFILE    DD DSN=USERID.FILE1.BKP(0),DISP=SHR
```

Q: How to use previous generation of a GDG as input?

A:
```
//INFILE    DD DSN=USERID.FILE1.BKP(-1),DISP=SHR
```

Q: How to use all generations of a GDG as input?

A:
```
//INFILE    DD DSN=USERID.FILE1.BKP,DISP=SHR
```

Q: How to delete all generations of a GDG?

A:
```
//STEP1  EXEC PGM=IEFBR14
//DD1    DD DSN=USERID.FILE1.BKP,DISP=(MOD,DELETE,DELETE)
```

Q: How to delete all generations and also GDG base?

A:
```
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSIN     DD *
   DELETE USERID.FILE1.BKP FORCE
/*
```

Q: How to use a new generation created from step1 into step2?

A:
```
//STEP1 EXEC PGM=IEBGENER
//SYSPRINT  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSUT1    DD DSN=USERID.FILE1,DISP=SHR
//SYSUT2    DD DSN=USERID.FILE1.BKP(+1),
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,
//             SPACE=(CYL,(10,5),RLSE),
//             DCB=(RECFM=FB,LRECL=100,BLKSIZE=0)
//SYSIN     DD DUMMY
//*
//STEP2     PGM=PGM1
//INFILE    DD DSN=USERID.FILE1.BKP(+1),DISP=SHR
```

While the job is executing the generations are not upgraded, so use (+1) as input
If STEP2 abends and if you want to restart from STEP2 then make file in step2 as below

```
//STEP2     PGM=PGM1
//INFILE    DD DSN=USERID.FILE1.BKP(0),DISP=SHR
```

When the job is abended in step2 the generations are upgraded and (+1) becomes (0)

Q: Current generation on the file is FILE1.G9999.V00. If I create a new generation with (+1) what happens?

A: `FILE1.G0001.V00`

Q: On what situation you cannot give GDG base as input?

A: When different generations are created with different lengths

Q: Is it possible to create different generations with different lengths?

A: It is possible but it is not recommended in normal situations.
It is required when the existing file length is increase for future maintenance
From the last 3 months, (+1) generations are created on FILE1 with length 100.
For a new requirement, it is requested to increase from 100 to 110.
In this case old generations will have 100 length and for the new generations it will create 110.

Q:
```
//JOB1 JOB ......
//..
//STEP1   EXEC PGM=PGM1
//INFILE  DD DSN=FILEX,DISP=SHR
//OUTFILE DD DSN=FILEY(+1),DISP=(NEW,CATLG,DELETE),..
//*
//STEP2   EXEC PGM=PGM2
//INFILE  DD DSN=FILEZ,DISP=SHR
//OUTFILE DD DSN=FILEY(+2),DISP=(NEW,CATLG,DELETE),..
//*
//STEP3   EXEC PGM=PGM3
//INFILE  DD DSN=FILEY(+1),DISP=SHR
//*
//STEP4   EXEC PGM=PGM4
//INFILE  DD DSN=FILEY(+2),DISP=SHR
```

In the above example if Job fails at Step3 then how do you restart?

Ans:
```
//JOB1 JOB ......RESTART=STEP3
//..
//STEP1   EXEC PGM=PGM1
//INFILE  DD DSN=FILEX,DISP=SHR
//OUTFILE DD DSN=FILEY(+1),
//         DISP=(NEW,CATLG,DELETE)
//*
//STEP2   EXEC PGM=PGM2
//INFILE  DD DSN=FILEZ,DISP=SHR
//OUTFILE DD DSN=FILEY(+2),
//         DISP=(NEW,CATLG,DELETE),..
//*
//STEP3   EXEC PGM=PGM3
//INFILE  DD DSN=FILEY(-1),DISP=SHR
//*
//STEP4   EXEC PGM=PGM4
//INFILE  DD DSN=FILEY(0),DISP=SHR
```

Change generation on step3 as (-1) and step4 as (0)

In the above example if Job fails at Step2 then how do you restart?

Ans:
```
//JOB1 JOB ......RESTART=STEP2
//..
//STEP1   EXEC PGM=PGM1
//INFILE  DD DSN=FILEX,DISP=SHR
//OUTFILE DD DSN=FILEY(+1),
//         DISP=(NEW,CATLG,DELETE),..
//STEP2   EXEC PGM=PGM2
//INFILE  DD DSN=FILEZ,DISP=SHR
//OUTFILE DD DSN=FILEY(+1),
//         DISP=(NEW,CATLG,DELETE),..
//*
//STEP3   EXEC PGM=PGM3
//INFILE  DD DSN=FILEY(0),DISP=SHR
//*
//STEP4   EXEC PGM=PGM4
//INFILE  DD DSN=FILEY(+1),DISP=SHR
```

Change generation on step2 as (+1), step3 as (0) and step4 as (+1)

Spool (Simultaneous peripherals objects online)

When do you go to spool?

To check the below items

1. Job execution status
2. Steps execution status
3. Displayed statements of a program
4. JCL errors on submitted jobs
5. Return code of each step
6. Information about the abended steps
7. Infinite loop
8. Reports written into spool

How to go to the SPOOL?

9;S;ST From ISPF primary option menu

What are different options in SPOOL?

1. ST -- To check the status of the jobs
2. I -- To check the jobs in Input queue
3. O -- To check the jobs in Output queue
4. H -- To check the held output queue
5. DA -- To check the status of the jobs that are running now.

What are different messages of a job in a SPOOL?

Spool Messages

Job Level Messages	Step Level Messages
1. JESMSGLG It is used to find Start time and end time Return code or abend code of each step CPU and elapsed time of job and step File name on which job is waiting Authorization failures	**1. SYSOUT** It is used to find Display statements of COBOL Abend code and offset of variable used in abend step Complete information about U4038
2. JESJCL It is used to find All JCL statements Proc Expansion statements Expansion of symbolic parameters Overrides given from JCL to PROC	**2. SYSPRINT** It is used to find Statistics of Utility program Number of records read and write Error messages on utility programs
3. JESYSMSG It is used to find JCL errors on line numbers in JESJCL Allocation failure messages	**3. SYSUDUMP** It is used find Dump information on abends

How to correct JCL errors when a job is failed with JCL errors?

1. Go to JESYSMSG and get the statement number that is causing error
2. Go to JESJCL and find the statement and find the mistake
3. Go to JCL member in PDS and correct it

How to check whether the job is in infinite loop or not?

1. Go to DA in the Spool while the job is running
2. Check the EXCP-CNT and CPU% of the job
   If the EXCP-CNT is constant and CPU% is high then a step of job is in the infinite loop
3. Correct until condition of the program to fix infinite loop

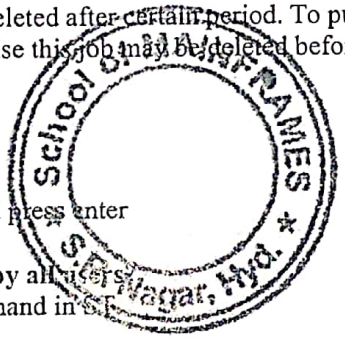
How to check JCL errors without submitting for execution?
1. SUBMIT it to check JCL errors with TYPRUN=SCAN in JOB card.
2. Use below commands in view or edit mode. No need to SUBMIT the job.
   !JCK
   JSCAN
   JEM

Here errors statistics are shown on the top of the member and errors are shown on error line of the JCL.

How to Submit a JCL which is logged in the SPOOL?
Use Command SJ infront of the Jobname in the spool. It will show JCL in the spool.
Then submit it for execution.

If a Job is waiting for a file which is used by another user, then how to find the user?
1. Go to JESMSGLG and see the file name that is waiting for
2. Press F1 twice on the file name in 3.4. It gives user id with whom this file is used

Why to capture spool job into a file? How to capture it?
Why to Capture?
Job written in the spool will be deleted after certain period. To put this job for review, capture this job into a sequential file. Otherwise this job may be deleted before the job is reviewed by peer reviewer



How to Capture it?
1. XDC Jobname
2. Give file name, disp, Lrecl and press enter

How to see the jobs submitted by all users?
Use below Command level command in ST.
   Owner *; Pre *

How to see all jobs submitted by user id?
Use below Command level command in ST.
   Owner Userid; Pre *

How to see all jobs submitted by users with job name starts with TAS?
Use below Command level command in ST.
   Owner Userid; Pre TAS*

How to release a job submitted with Typrun=hold?
Go to SPOOL and use a line command A in front of the jobname.
Developer may not have this facility but operator can have such a facility.

	Jobname	status
A	job1	hold

JCL Utilities

IEFBR14

It is a Null program used to create, catalog, delete a file. A DD statement cannot be used without a EXEC statement, with this reason this Null program is used.
Even though the files can be created with this utility, it is highly recommended to delete an output file before it is created.

Create a physical sequential file (PS)

```
//STEP1     EXEC PGM=IEFBR14
//DD1       DD DSN=OZA183.SMF.FILE1,
//             DISP=(NEW,CATLG,DELETE),
//             SPACE=(CYL,(1,1),RLSE),
//             UNIT=SYSDA,
//             DCB=(RECFM=FB,LRECL=100,BLKSIZE=1000)
```

Create a Partition dataset file (PDS)

```
//STEP1     EXEC PGM=IEFBR14
//DD1       DD DSN=OZA183.SMF.FILE1,
//             DISP=(NEW,CATLG,DELETE),
//             SPACE=(CYL,(1,1,10),RLSE),
//             UNIT=SYSDA,
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=800,DSORG=PO)
```

→ Directory blocks 10 are used.
→ Maximum number of members =(10 * 6) -1 =59
→ Mostly, PDS is created with Lrecl=80

Delete a file before it is created in the subsequent steps of the same JCL

```
//STEP1     EXEC PGM=IEFBR14
//DD1       DD DSN=OZA183.SMF.FILE2,
//             DISP=,DISP=(MOD,DELETE,DELETE),
//             SPACE=(TRK,(1,1),RLSE)
//*
//STEP2     EXEC PGM=PGM1
//INFILE    DD DSN=OZA183.SMF.FILE1,DISP=SHR
//OUTFILE   DD DSN=OZA183.SMF.FILE2,
//             DISP=(NEW,CATLG,DELETE),
//             SPACE=(CYL,(1,1),RLSE),
//             UNIT=SYSDA,
//             DCB=(RECFM=FB,LRECL=50,BLKSIZE=0)
```

→ If the file is existing then it opens in append mode it creates. In both the cases it deletes after execution.

Note: If STEP1 is not coded then OUTFILE has to be deleted manually before you submit JOB. Otherwise it fails with JCL error 'OZA183.SMF.FILE2 already exists'.
Note: One IEFBR14 step is enough to delete all output files before these are created in the other steps.

How to delete multiple files in a single IEFBR14 step

```
//STEP1     EXEC PGM=IEFBR14
//DD1       DD DSN=OZA183.SMF.OUTFILE1,
//             DISP=(MOD,DELETE,DELETE),
//             SPACE=(TRK,(1,1),RLSE)
//DD2       DD DSN=OZA183.SMF.OUTFILE2,
//             DISP=(MOD,DELETE,DELETE),
//             SPACE=(TRK,(1,1),RLSE)
```

→ Note: Same DDname cannot be used more than once

How to delete all generations of a GDG base

```
//STEP1    EXEC PGM=IEFBR14
//DD1      DD DSN=OZA183.SMF.GDGBASE,
//            DISP=(MOD,DELETE,DELETE)
```

→ It deletes all generations of a GDG but not GDG base

IEBGENER

It is mainly to used to copy a file into another file. i.e. to take a backup.
It is also used to send a mail from Batch JCL

Copy one file into another file

```
//STEP2   EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=OZA183.SMF.FILE1,DISP=SHR
//SYSUT2   DD DSN=OZA183.SMF.FILE2,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=SYSDA,
//            SPACE=(CYL,(10,5),RLSE),
//            DCB=(RECFM=F,LRECL=100,BLKSIZE=0)
//SYSIN    DD DUMMY
```

→ Input and output should have same length

→ DUMMy is used because a control info is not required for a simple copy

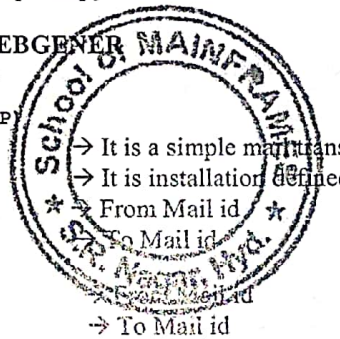
General syntax of Email with IEBGENER

```
//STEP1    EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT2   DD SYSOUT=(B,SMTP)
//SYSUT1   DD *
HELO <SYSTEMID>
MAIL FROM:<mailid1>
RCPT TO:<mailid2>
DATA
FROM: mailid1
TO: mailid2
SUBJECT: TEST MAIL

 This is my first mail sent from mainframes

QUIT
/*
```

→ It is a simple mail transfer protocol
→ It is installation defined
→ From Mail id
→ To Mail id
→ From Mail id
→ To Mail id
→ Subject in the mail

→ Message part of the mail

→ End of message



A Sample situation to generate an Email

```
//PABCDACT JOB (8012P),'ACCOUNT BALANCE JOB',CLASS=P
//*
//STEP1    EXEC PGM=BALANCE1
//SYSPRINT DD SYSOUT=*
//*
//FILE1    DD DSN=PABC.PSEQ.ACCT1,DISP=SHR
//FILE2    DD DSN=PABC.PSEQ.ACCT2,DISP=SHR
//REPORT1  DD SYSOUT=(*,$001)
//*
//STEP2    EXEC PGM=IEBGENER,COND=(4000,NE,STEP1)
//SYSPRINT DD SYSOUT=*
//SYSUT2   DD SYSOUT=(B,SMTP)
```

→ It is a file balancing program to check both the files are balanced or not This program generates return code 4000 if the files are not balanced

→ It is a report. Report name is PABCDACT001

→ It executes only when step1 returns 4000

```
//SYSUT1   DD *
HELO ABCCOMPANY
MAIL FROM:<TECH_SUPPORT@ABC.COM>
RCPT TO:<ACCOUNT_DEPT@ABC.COM>
DATA
FROM: TECH_SUPPORT@ABC.COM
TO: ACCOUNT_DEPT@ABC.COM
SUBJECT: ACCOUNTING FILES ARE OUT OF BALANCE

 Accounting files are out of balance

 Check the balancing report :PABCDACT001

QUIT
/*
```

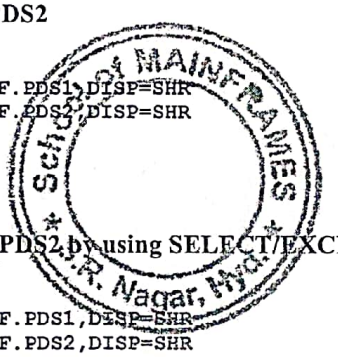
→ This step generates mail from TECH_SUPPORT@ABC.COM to ACCOUNT_DEPT@ABC.COM

IEBCOPY

It is used create a backup of a PDS.
It is used to copy few members of a PDS1 to PDS2.
It is used to compress a PDS to reclaim the unused space from deleted members..

Copy all members of PDS1 to PDS2

```
//STEP1    EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//DD1      DD DSN=OZA183.SMF.PDS1,DISP=SHR
//DD2      DD DSN=OZA183.SMF.PDS2,DISP=SHR
//SYSIN    DD *
    COPY INDD=DD1,OUTDD=DD2
/*
```



Copy few members of PDS1 to PDS2 by using SELECT/EXCLUDE

```
//STEP1    EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//DD1      DD DSN=OZA183.SMF.PDS1,DISP=SHR
//DD2      DD DSN=OZA183.SMF.PDS2,DISP=SHR
//SYSIN    DD *
    COPY INDD=DD1,OUTDD=DD2
       SELECT MEMBER=(mem1,mem2,mem3)/ EXCLUDE MEMBER=(mem5,mem6)
/*
```

Select list is used to include Exclude list is used to omit

Compress PDS1

```
//STEP1    EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//DD1      DD DSN=OZA183.SMF.PDS1,DISP=SHR
//DD2      DD DSN=OZA183.SMF.PDS1,DISP=SHR
//SYSIN    DD *
    COPY INDD=DD1,OUTDD=DD2
/*
```

→ Input and output are same

IEBCOMPR

It is used to compare two sequential files or Members of two PDS.
It halts after 10 differences.
Column comparison is not possible with this.
It does not say about where differences.
ISRSUPC Utility is used in place of IEBCOMPR.

Compare two PS files

```
//STEP1 EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=*                          → Output is written into SYSPRINT
//SYSUT1 DD DSN=OZA183.SMF.FILE1,DISP=SHR       → PS File
//SYSUT2 DD DSN=OZA183.SMF.FILE2,DISP=SHR       → PS File
//SYSIN DD *
  COMPARE TYPORG=PS
/*
```

Compare two PDS and all its members

```
//STEP1 EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=*                          → Output is written into SYSPRINT
//SYSUT1 DD DSN=OZA183.SMF.PDS1,DISP=SHR        → PDS
//SYSUT2 DD DSN=OZA183.SMF.PDS2,DISP=SHR        → PDS
//SYSIN DD *
  COMPARE TYPORG=PO
/*
```


ISRSUPC

It is used to compare two sequential files
It is used to compare all members of PDS1 to PDS2
It is used to compare selected members of PDS1 with the same names in PDS2
In all above cases, column comparison is possible

Compare two sequential files

```
//SUPERC  EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','')
//NEWDD   DD DSN=OZA183.SMF.FILE1,DISP=SHR
//OLDDD   DD DSN=OZA183.SMF.FILE2,DISP=SHR
//OUTDD   DD SYSOUT=*
```

Compare two sequential files on specified columns

```
//SUPERC  EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','')
//NEWDD   DD DSN=OZA183.SMF.FILE1,DISP=SHR
//OLDDD   DD DSN=OZA183.SMF.FILE2,DISP=SHR
//OUTDD   DD SYSOUT=*
//SYSIN   DD *
CMPCOLM 1:3 5:6      → Compare first 3 bytes and then 5th and 6th bytes
/*
```

Compare all members of PDS1 with same names in PDS2

```
//SUPERC  EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','')
//NEWDD   DD DSN=OZA183.SMF.PDS1,DISP=SHR
//OLDDD   DD DSN=OZA183.SMF.PDS2,DISP=SHR
//OUTDD   DD SYSOUT=*
```

Compare selected members of PDS1 with same names in PDS2

```
//SUPERC  EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','')
//NEWDD   DD DSN=OZA183.SMF.PDS1,DISP=SHR
//OLDDD   DD DSN=OZA183.SMF.PDS2,DISP=SHR
//OUTDD   DD SYSOUT=*
//SYSIN   DD *
SELECT MEM1,MEM2,MEM3
/*
```

IEBEDIT

It is used to execute required steps, omit steps not required and restart from a job step.

To execute the required steps

```
//STEP1     EXEC PGM=IEBEDIT
//SYSPRINT DD SYSOUT=*
//SYSUT1    DD DSN=PDS1(JCL1),DISP=SHR → Keep the JCL in a member of a PDS/PS
//SYSUT2    DD SYSOUT=(*,INTRDR)
//SYSIN     DD *
    EDIT TYPE=INCLUDE,STEPNAME=(STEP10,STEP5,STEP15)→ Use Include to execute
/*                                                    required steps
```

To omit the steps that are not required

```
//STEP1     EXEC PGM=IEBEDIT
//SYSPRINT DD SYSOUT=*
//SYSUT1    DD DSN=PDS1(JCL1),DISP=SHR → Keep the JCL in a member of a PDS/PS
//SYSUT2    DD SYSOUT=(*,INTRDR)
//SYSIN     DD *
    EDIT TYPE=EXCLUDE,STEPNAME=(STEP1,STEP6,STEP12)→ Use exclude to omit
/*                                                    the steps
```

To restart from the required steps

```
//STEP1     EXEC PGM=IEBEDIT
//SYSPRINT DD SYSOUT=*
//SYSUT1    DD DSN=PDS1(JCL1),DISP=SHR → Keep the JCL in a member of a PDS/PS
//SYSUT2    DD SYSOUT=(*,INTRDR)
//SYSIN     DD *
    EDIT TYPE=POSITION,STEPNAME=(STEP6)→ Use Position to restart job from
/*                                       required steps
```


IDCAMS

Create GDG base

```
//STEP1  EXEC  PGM=IDCAMS
//SYSIN  DD    *
  DEFINE GDG(NAME(OZA183.SMF.GDGBASE)    -
  LIMIT(5)                               -
  NOEMPTY                                -
  SCRATCH)
/*
```

Delete GDG base

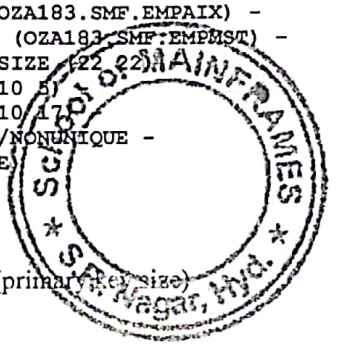
```
//STEP1  EXEC  PGM=IDCAMS
//SYSIN  DD    *
  DELETE OZA183.SMF.GDGBASE FORCE        -
  LIMIT(5)                               -
  NOEMPTY                                -
  SCRATCH)
/*
```

Define KSDS VSAM file

```
//STEP1     EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD  *
 DEFINE CLUSTER (NAME(OZA183.SMF.EMPMST) -
                 CYL(5 5)      -
                 FSPC(10 10)   -
                 KEYS(7 0)     -
                 RECSZ(31 31)  -
                 SHR(2 3)      -
                 RESUE         -
                 INDEXED       -
                 CISZ(2048))   -
        DATA (NAME(OZA183.SMF.EMPMST.DATA)) -
        INDEX (NAME(OZA183.SMF.EMPMST.INDEX) -
               CISZ(512))

/*
```

Define AIX

```
//DEFALTIX EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
 DEFINE    AIX      (NAME (OZA183.SMF.EMPAIX) -
                     RELATE (OZA183.SMF.EMPMST) -
                     RECORDSIZE (22 22)
                     CYL  (10 5)
                     KEYS (10 17)
                     UNIQUE/NONUNIQUE -
                     UPGRADE)
/*
```



Default is unique
Record size = 5+alt key size + n (primary key size)

Define PATH

```
//DEFPATH  EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
 DEFINE    PATH     (NAME (OZA183.SMF.EMPMST.PATH) -
                     PATHENTRY (OZA183.SMF.EMPAIX) -
                     UPDATE)
/*
```

Build Alternate Index

```
//BLDINDEX EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
//BASEDD   DD   DSN=OZA183.SMF.EMPMST,DISP=SHR
//AIXDD    DD   DSN=OZA183.SMF.EMPAIX,DISP=SHR
//IDCUT1   DD   UNIT=SYSDA,SPACE=(CYL,55)
//IDCUT2   DD   UNIT=SYSDA,SPACE=(CYL,55)
//SYSIN    DD   *
 BLDINDEX  INFILE (BASEDD) -
           OUTFILE (AIXDD)
/*
```

Unload a VSAM file into PS in Method1

```
//STEP1     EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//INDD1     DD DSN=OZA183.SMF.EMPMST,DISP=SHR
//OUTDD1    DD DSN=OZA183.SMF.EMPMST.BKUP,DISP=SHR
//*
//SYSIN     DD  *
 REPRO -
     INFILE (INDD1)   -
     OUTFILE (OUTDD1)
/*
```

Unload a VSAM file into PS in Method2

```
//STEP1     EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//*
//SYSIN     DD  *
 REPRO -
     INDATASET (OZA183.SMF.EMPMST)   -
     OUTDATASET (OZA183.SMF.EMPMST.BKUP)
/*
```

Delete a VSAM file

```
//STEP1     EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//*
//SYSIN     DD  *
 DELETE OZA183.SMF.EMPMST PURGE   → if dataset not found it returns maxcc=8
 SET MAXCC=0                      → here maxcc sets to zero
/*
```

Sort

Different uses of SORT

1. To take a back up of a file
2. To skip first few records and copy other few records
3. To Sort a file before generate a report
   To sort a trx file that reduces number of hits to master file
4. To sort a file on key in ascending order and eliminate duplicates before load a VSAM KSDS file
5. To copy records conditionally
6. To split a file into multiple files based on different conditions to input to different programs or jobs or different interface systems
7. To summarize records before generating summary reports
8. To eliminate duplicates on entire record or single field or multiple fields
9. To restructure input into output without writing a program
10. To join two files to generate matched records or unmatched records
11. To convert VB to FB and FB to VB
12. To include a new field in an existing file with a default value
13. To change the value of a field with another value on a condition

Different SORT options

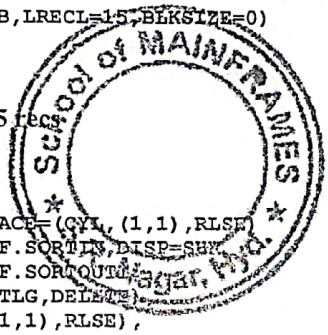
Simple Backup

Data and layout of the file OZA183.SMF.SORTIN are shown below

```
A003TNP10500000
A002KAP10700000
A002APP10500000
A001APP10500000
A001TNP20600000
A002TNP10500000
A003KAP20600000
A001KAP10500000
A003APP10700000
```

```
01 sales-rec.
   05 agent-id     pic x(4).
   05 state-code   pic x(2).
   05 product-code pic x(2).
   05 sales-amt    pic 9(5)v99.
```

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.SORTIN,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.SORTIN.BKUP1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=15,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=COPY
/*
```



Skip first 4 recs and copy next 5 recs

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.SORTIN,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=15,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=COPY,
 SKIPREC=4,
 STOPAFT=5
/*
```

Output

```
A001TNP20600000
A002TNP10500000
A003KAP20600000
A001KAP10500000
A003APP10700000
```

Copy last 5 recs first and first 4 recs last

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.SORTIN,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=13,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=COPY,
 SKIPREC=4,
 STOPAFT=5
/*
```

Intermediate Output

```
A001TNP20600000
A002TNP10500000
A003KAP20600000
A001KAP10500000
A003APP10700000
```

```
//STEP2    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.SORTIN,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.SOROUT1,
//            DISP=(MOD,CATLG,CATLG), --> append
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=13,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=COPY,
 SKIPREC=0,
 STOPAFT=4
/*
```

Output

```
A001TNP20600000
A002TNP10500000
A003KAP20600000
A001KAP10500000
A003APP10700000
A003TNP10500000
A002KAP10700000
A002APP10500000
A001APP10500000
```

Sort on agent-id in asc order

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.SORTIN,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=13,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=(1,4,CH,A)
/*
```

Output

```
A001APP10500000
A001TNP20600000
A001KAP10500000
A002KAP10700000
A002APP10500000
A002TNP10500000
A003TNP10500000
A003KAP20600000
A003APP10700000
```

Sort on agent-id in asc order and then on State-code in asc order

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.SORTIN,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=13,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=(1,4,CH,A,5,2,CH,A)
/*
```

Output

```
A001APP10500000
A001KAP10500000
A001TNP20600000
A002APP10500000
A002KAP10700000
A002TNP10500000
A003APP10700000
A003KAP20600000
A003TNP10500000
```

Sort on agent-id in asc order and on sales-amount in Desc order

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.SORTIN,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=13,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=(1,4,CH,A,9,5,ZD,D)
/*
```

Output

```
A001TNP20600000
A001APP10500000
A001KAP10500000
A002KAP10700000
A002APP10500000
A002TNP10500000
A003APP10700000
A003KAP20600000
A003TNP10500000
```

Copy records only when state code is AP

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.SORTIN,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=13,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=COPY
 INCLUDE COND=(5,2,CH,EQ,C'AP')
/*
```

Output

```
A002APP10500000
A001APP10500000
A003APP10700000
```

Copy records only when state code is not AP

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.SORTIN,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=13,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=COPY
 OMIT COND=(5,2,CH,EQ,C'AP')
/*
```

Output

```
A003TNP10500000
A002KAP10700000
A001TNP20600000
A002TNP10500000
A003KAP20600000
A001KAP10500000
```

Split AP records into 1st file and non AP records into 2nd file

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.SORTIN,DISP=SHR
//SORTOF1  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=13,BLKSIZE=0)
//SORTOF2  DD DSN=OZA183.SMF.SORTOUT2,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=13,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=COPY
 OUTFIL FILES=1,INCLUDE=(5,2,CH,EQ,C'AP')
 OUTFIL FILES=2,OMIT=(5,2,CH,EQ,C'AP')
/*
```

Sortof1

```
A002APP10500000
A001APP10500000
A003APP10700000
```

Sortof2

```
A003TNP10500000
A002KAP10700000
A001TNP20600000
A002TNP10500000
A003KAP20600000
A001KAP10500000
```

Copy few fields of input to output with OUTREC FIELDS Agent-id, product-code sales-amount only

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.SORTIN,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=11,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=COPY
 OUTREC FIELDS=(1:1,4,5:7,2,7:9,5)
/*
```

Output

```
A003P10500000
A002P10700000
A002P10500000
A001P10500000
A001P20600000
A002P10500000
A003P20600000
A001P10500000
A003P10700000
```

Find total number of records on each product-code

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.SORTIN,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=06,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=COPY
 OUTREC FIELDS=(1:7,2,3:C'0001')   → Include 0001 in every record
/*
//STEP4    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.SORTOUT1,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT2,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=06,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=(1,2,CH,A)          → Summary records on product
 SUM FIELDS=(3,4,ZD)
/*
```

Intermediate output

```
P10001
P10001
P10001
P10001
P20001
P10001
P20001
P10001
P10001
```

Final Output

```
P10007
P20002
```

Eliminate duplicates on entire rec

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD *
CCCCC
AAAAA
BBBBB
AAAAA
CCCCC
BBBBB
CCC
```

Output

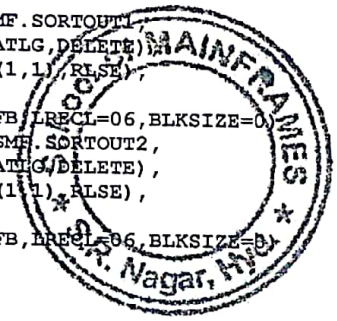
```
AAAAA
BBBBB
CCC
CCCCC
```

```
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=06,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=(1,5,CH,A)
 SUM FIELDS=NONE
/*
```

Eliminate duplicates on first 3 characters and capture eliminated duplicates in another file

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD *
CCCCC
AAAAA
BBBBB
AAAAA
CCCCC
BBBBB
CCC
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=06,BLKSIZE=0)
//SORTXSUM DD DSN=OZA183.SMF.SORTOUT2,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=06,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=(1,3,CH,A)
 SUM FIELDS=NONE,XSUM
/*
```

SORTOUT	SORTXSUM
AAAAA	AAAAA
BBBBB	BBBBB
CCC	CCCCC
	CCCCC



Summarize records on number field (4th to 7th position), if the first 3 chars are same

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD *
CCC10000
AAA20000
BBB10000
AAA10000
CCC20000
BBB20000
DDD10000
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=06,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=(1,3,CH,A)
 SUM FIELDS=(4,5,ZD)
/*
```

SORTOUT
AAA30000
BBB30000
CCC30000
DDD10000

Change the values at 4th and 5th col from '11' to '33'

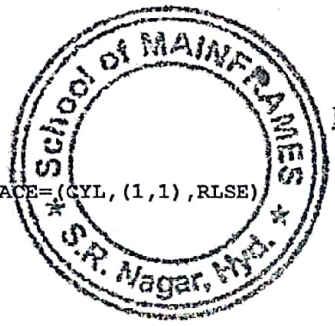
```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD *
AAA11
BBB22
DDD11
CCC22
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=05,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=COPY
 OUTREC FIELDS=(1:1,3,4:4,2,CHANGE=(2,C'11',C'33'),NOMATCH=(4,2))
/*
```

Output

```
AAA33
BBB22
DDD33
CCC22
```

Reverse records based on its position

Input	Output
AAA	CCC
BBB	DDD
DDD	BBB
CCC	AAA




```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD *
AAA
BBB
DDD
CCC
//SORTOUT  DD DSN=&&TEMP1,
//            DISP=(NEW,PASS),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=17,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=COPY
 OUTREC FIELDS=(1:1,3,4:SEQNUM,4,ZD,START=1,INCR=1)
/*
//STEP2    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=&&TEMP1,DISP=(OLD,DELETE)
//SORTOUT  DD DSN=OZA183.SMF.SORTOUT1,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=13,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=(4,4,ZD,D)
 OUTREC FIELDS=(1:1,3)
/*
```

Intermediate output

```
AAA0001
BBB0002
DDD0003
CCC0004
```

→ It shows how to create temporary dataset and how to pass to the next steps of jcl

→ Include a record number

→ Sort rec num in reverse order

→ do not include record number

Final Output

```
CCC
DDD
BBB
AAA
```

Merge records of two files alternative

File1	File2	Output
AAA	XXX	AAA
BBB	YYY	XXX
CCC	ZZZ	BBB
		YYY
		CCC
		ZZZ

```
//STEP1     EXEC PGM=SORT
//SYSOUT    DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN    DD *
AAA
BBB
CCC
/*
//SORTOUT   DD DSN=&&TEMP1,
//             DISP=(NEW,PASS),
//             SPACE=(CYL,(1,1),RLSE),
//             UNIT=SYSDA,
//             DCB=(RECFM=FB,LRECL=7,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=COPY
 OUTREC FIELDS=(1:1,3,4:SEQNUM,4,ZD,START=1,INCR=2)
/*
//STEP2     EXEC PGM=SORT
//SYSOUT    DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN    DD *
XXX
YYY
ZZZ
/*
//SORTOUT   DD DSN=&&TEMP2,
//             DISP=(NEW,PASS),
//             SPACE=(CYL,(1,1),RLSE),
//             UNIT=SYSDA,
//             DCB=(RECFM=FB,LRECL=7,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=COPY
 OUTREC FIELDS=(1:1,3,4:SEQNUM,4,ZD,START=2,INCR=2)
/*
//STEP3     EXEC PGM=SORT
//SYSOUT    DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN    DD DSN=&&TEMP1,DISP=(OLD,DELETE)
//          DD DSN=&&TEMP2,DISP=(OLD,DELETE)
//SORTOUT   DD DSN=OZA183.SMF.SORTOUT1,
//             DISP=(NEW,CATLG,DELETE),
//             SPACE=(CYL,(1,1),RLSE),
//             UNIT=SYSDA,
//             DCB=(RECFM=FB,LRECL=3,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=(4,4,ZD,A)
 OUTREC FIELDS=(1:1,3)
/*
```

Intermediate output

```
AAA0001
BBB0003
CCC0005
```

→ Shows about how to create temporary dataset

Intermediate Output

```
XXX0002
YYY0004
ZZZ0006
```

→ Shows about how to concatenate two files

Final output

```
AAA
XXX
BBB
YYY
CCC
ZZZ
```

Match two files to generate mached recs and unmatched from 1st file

Product file	Forex file	Matched recs	Unmatched recs from 1st file
P1IND01000	AUD000900	P1IND01000000020	P7EUR05000
P2JPY02000	CHD000100	P2JPY02000000300	P8RUR05000
P3GBP02000	GBP001500	P3GBP02000001500	P9EUR05000
P4JPY05000	IND000020	P4JPY05000000300	
P5IND06000	JPY000300	P5IND06000000020	
P6USD05000	USD001000	P6USD05000001000	
P7EUR05000			
P8RUR05000			
P9EUR05000			

Layout of product File

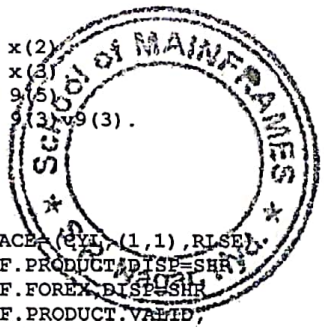
```
01 Product-Rec.
   05 product-code  pic x(2).
   05 curr-code     pic x(3).
   05 product-price pic 9(5).
```

Layout of Forex File

```
01 Forec-Rec.
   05 curr-code         pic x(3).
   05 conv-rate-in-usd  pic 9(3)v9(3).
```

Layout of matched file (Layout of unmatched file is same as Product file)

```
01 Product-Rec.
   05 product-code      pic x(2).
   05 curr-code         pic x(3).
   05 product-price     pic 9(5).
   05 conv-rate-in-usd  pic 9(3)v9(3).
```



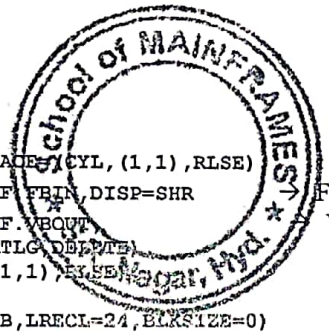

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTJNF1 DD DSN=OZA183.SMF.PRODUCT,DISP=SHR
//SORTJNF2 DD DSN=OZA183.SMF.FOREX,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.PRODUCT.VALID,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=16,BLKSIZE=0)
//SYSIN DD *
  JOINKEYS FILES=F1,FIELDS=(3,3,A)
  JOINKEYS FILES=F2,FIELDS=(1,3,A)
  JOIN PAIRED
  REFORMAT FIELDS=(F1:1,10,F2:4,6)     -> Matched records
  SORT FIELDS=COPY
/*
//STEP2    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTJNF1 DD DSN=OZA183.SMF.PRODUCT,DISP=SHR
//SORTJNF2 DD DSN=OZA183.SMF.FOREX,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.PRODUCT.ERROR,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=10,BLKSIZE=0)
```

```
//SYSIN DD *
  JOINKEYS FILES=F1,FIELDS=(3,3,A)
  JOINKEYS FILES=F2,FIELDS=(1,3,A)
  JOIN UNPAIRED,F1,ONLY
  REFORMAT FIELDS=(F1:1,10)          → UnMatched records from file1
  SORT FIELDS=COPY
/*
```

Sort a variable length file on its first 4 bytes

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.FILE1,DISP=SHR
//SORTOUT  DD DSN=OZA183.SMF.FILE2,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=FB,LRECL=06,BLKSIZE=0)
//SYSIN DD *
 SORT FIELDS=(5,4,CH,A)      → First 4 bytes of Variable length files
/*                                So take the offset from 5 instead of 1.
```

Convert a FB file into VB file




```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.FBIN,DISP=SHR      → FB with lrecl=20
//VBOUT    DD DSN=OZA183.SMF.VBOUT,             → VB with lrecl=24
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(1,1),RLSE),
//            UNIT=SYSDA,
//            DCB=(RECFM=VB,LRECL=24,BLKSIZE=0)
//SYSIN  DD    *
  SORT FIELDS=COPY
  OUTFIL FNAMES=VBOUT,FTOV,VLTRIM=C' '  → Remove trailing spaces and create VB
/*
```

FBIN	Length of record	VBOUT	Length of record
AAAAAAAA	20	AAAAAAAA	4+8 = 12
BBBBBBBBBBB	20	BBBBBBBBBBB	4+11 = 15
CC	20	CC	4+2 = 06
Total bytes	60	Total bytes	33

Convert a VB file into FB file

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.VBOUT,DISP=SHR → Output from above step
//FBOUT    DD DSN=OZA183.SMF.FBOUT1,
//             DISP=(NEW,CATLG,DELETE),
//             SPACE=(CYL,(1,1),RLSE),
//             UNIT=SYSDA,
//             DCB=(RECFM=FB,LRECL=20,BLKSIZE=0)
//SYSIN  DD    *
  SORT FIELDS=COPY
  OUTFIL FNAMES=FBOUT,VTOF,OUTREC=(1:5,20)  → It pads smaller records with trialing
/*                                             spaces
```

```
VBOUT        Length of record     FBOUT                    Length of record

AAAAAAAA        4+8  = 12         AAAAAAAA                        20
BBBBBBBBBBB     4+11 = 15         BBBBBBBBBBB                     20
CC              4+2  = 06         CC                              20
                      ----                                       ----
          Total bytes  33                          Total bytes  60
                      ----                                       ----
```

Convert a VB file into FB file and pad trailing chars in smaller records with '*'

```
//STEP1    EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)
//SORTIN   DD DSN=OZA183.SMF.VBOUT,DISP=SHR
//FBOUT    DD DSN=OZA183.SMF.FBOUT2,
//             DISP=(NEW,CATLG,DELETE),
//             SPACE=(CYL,(1,1),RLSE),
//             UNIT=SYSDA,
//             DCB=(RECFM=FB,LRECL=20,BLKSIZE=0)
//SYSIN  DD    *
  SORT FIELDS=COPY
  OUTFIL FNAMES=FBOUT,VTOF,OUTREC=(1:5,20),VLFILL=C'*'  → It pads smaller
/*                                                        records with trialing '*'
```

```
VBOUT        Length of record     FBOUT                    Length of record

AAAAAAAA        4+8  = 12         AAAAAAAA**********              20
BBBBBBBBBBB     4+11 = 15         BBBBBBBBBBB*********            20
CC              4+2  = 06         CC******************            20
                      ----                                       ----
          Total bytes  33                          Total bytes  60
                      ----                                       ----
```