Lesson 2:

JCL Syntax

JCL Statements

- JCL consists of eight basic statements which serve specific functions.
- The following JCL statements will be used:

JOB
The job statement defines a job and provides information about the job as a whole.

EXEC
The execute statement defines a job and provides information about the program to be executed.

JCL Statements

DD
The data definition statement describes a dataset needed by the program.

//*
The comment statement permits descriptive comments to be included with the JCL.

/*
The delimiter statement ends or delimits an in-stream dataset.

//
The null statement ends a job.

JCL Statements

PROC
The procedure statement begins a group of predefined JCL.

PEND
The procedure end statement ends a procedure.

JCL Syntax

- JCL statements are made up of one or more of these fields:

```
//NAME   OPERATOR   PARAMETERS   COMMENTS
```

  - The identification field or slash-slash identifies a statement as belonging to the job control language.
  - All JCL statements, except the delimiter, must contain two slashes (//) in columns 1 and 2 of the statement.
  - The first two columns of the delimiter must contain a slash asterisk (/*).

JCL Syntax

- The name field identifies the control statement in order that other statements and system control blocks can refer to it.
  - If a name is used, the name field must begin in column 3.
- The operation field specifies the type of control statement, such as JOB and DD.
  - The operation field follows the name field and must be preceded and followed by at least one blank.

JCL Syntax

- The operand field contains parameters separated by commas.
  - These parameters, which specify the action to be taken by the statement, constitute the bulk of JCL.
  - The operand field must follow the operation field and must be preceded and followed by at least one blank.
- The comments field contains additional information which may be helpful to others who look at the JCL.
  - It must follow the operand field and must be preceded by at least one blank.

Naming Rules

- All names used in JCL, such as the name field, must follow these rules:
- Contain from 1 through 8 characters.
  - Begin with an:
    - alphabetic character A - Z
    - national character @ # $
  - Contain only:
    - alphabetic character A - Z
    - national character @ # $
    - numeric character 0 - 9

Valid Names

- Examples of valid names which can be used include:

```
JOB398
FRED
$1984
Z
```


JOB Statement

- The JOB statement is the first JCL statement of a job.
- A job consists of one or more programs that z/OS schedules for execution and executes in sequence.
- The JOB statement designates the start of a job.
- The null or // statement usually designates the end of a job.
  - If the null statement is not present, the next JOB statement marks the end of the previous job and the beginning of the next job.

JOB Statement

- The JOB statement provides z/OS with a way to identify a job.
- It will be necessary to designate a name for a job which is used by z/OS.
- It will also be necessary to specify additional information which describes how a job will be classified and handled by z/OS.
  - accounting information, scheduling directions, printing, etc.
- A job consists of one or more steps.
  - Each job step is defined by an EXEC or execute statement which identifies the program to be run.

JOB Statement Format

- The JOB statement has the following format:

```
//jobname  JOB  positional,keyword  comments
```

  - Only use one JOB statement per job.
  - Each JOB statement must have a jobname which follows the naming rules stated previously.
- Although it is not necessary to make each jobname unique, doing so helps in keeping track of jobs.
  - In most z/OS installations, the jobname will contain the TSO userid.

JCL Syntax
Running a Simple Job

JOB Statement Format

- The operand field can have either positional or keyword parameters.
  - Positional parameters must follow in a specific sequence or position on the job statement.
  - Keyword parameters, designated by a keyword and an equals sign, may follow in any sequence.
- All JOB statement parameters are optional; unless a particular installation requires them.
  - If no parameters are provided, the comments field can not be included.

One Job - One Step

// JOB

Identifies the start of the job.

// EXEC

Defines step and identifies the program.

//

Null statement defines the end of job.

One Job - Multiple Steps

// JOB

Identifies the start of the job.

// EXEC

Defines the first step.

// EXEC

Defines the second step.

.

.

//

Null statement defines the end of job.

Accounting information

- The first positional parameter of the JOB statement allows accounting information to be supplied.
  - z/OS does not need this parameter.
  - However, an installation may require accounting information to charge specific users for their jobs and to keep track of the overall use of system resources.
- Each installation determines the content of this parameter.

Accounting information

- If the job accounting information includes only an account number and the number does not contain special characters, code the account number.

```
//SYSEDJOB  JOB  12A75
```

- The account number and each additional item of accounting information must be separated by commas.
- The job accounting information can be contained either in parentheses or apostrophes.

```
//SYSEDJOB  JOB  (12A75,DEPTD58,706)
//SYSEDJOB  JOB  `12A75,DEPTD58,706'
```


Programmer Name

- The second positional parameter on the JOB statement indicates the name or group responsible for a job.
  - The parameter is positional and must follow the accounting information.
  - z/OS does not use the programmer name parameter.
  - Instead, operators or production control staff use it to deliver listings to the appropriate person.
  - The programmer name parameter cannot exceed 20 characters.

Programmer Name

- If special characters other than hyphens, or leading or imbedded periods are included, enclose the name in apostrophes.

```
//SYSEDJOB  JOB  12A75,`M. HARRIS`
```

- If accounting information is not coded, a leading comma is necessary.
  - This informs z/OS that the first positional parameter is missing.

```
//SYSEDJOB  JOB  ,`M. HARRIS'
```


CLASS Parameter

- The CLASS parameter is an optional keyword parameter.
  - It assigns a job to an input job class.
  - JES uses this job class to determine the order in which jobs are scheduled for execution.
- The CLASS parameter is coded in the following format:

```
CLASS=job-class
```

- Any character A-Z or 1-9 defined by an installation can be used.

```
//SYSEDJOB  JOB  ,`M. HARRIS',CLASS=T
```

- If a class is not specified, the system uses a default based on where the job was submitted from: workstation, time-sharing user, etc.

CLASS Parameter

- Each installation defines input classes based on the scheduling requirements.
- Example:
  - Installation input job class definitions:

A	Regular (default)
N	Overnight turnaround
P	Production
Q	Special setup requirements
T	Tests

- The installation standards manual can be used in order to select the job class which fits the characteristics of the job.

MSGCLASS Parameter

- The MSGCLASS parameter specifies the output class to which the job listing is written.
- Code the MSGCLASS parameter in the following format:

```
MSGCLASS=output-class
```

- Output classes may be A-Z and 0-9, as in this example:

```
//SYSEDJOB  JOB  ,GEORGE,MSGCLASS=A
```


MSGCLASS Parameter

- As with input classes, each installation defines output classes which provide for a variety of output requirements.
- **Example:**

A	Standard forms (default)
B	Punch
L	Laser printer
M	Microfiche
P	Programmer test
T	TSO
Z	Discard

Continuing JCL Statements

- Keyword parameters can be coded in any order.
- JCL statements can be included on a single line as long as they do not extend past column 72.
- JCL statements coded on separate lines.

```
//NIGHT318   JOB   ACCT12,MIKE,
//           CLASS=N,
//           MSGCLASS=P
```

Continuing JCL Statements

- Although it doesn't make any difference to z/OS, coding parameters on separate lines will make JCL much easier to read and understand.
- When continuing a statement, ensure that each line, except the last, ends in a comma.
- Parameters on the continuation lines must begin between columns 4 and 16.

EXEC Statement - Execute

- Each step in a job is described by an EXEC (execute) statement.
  - It instructs the system what program is to execute.
  - It may also contain other information associated with that job step.
- An EXEC statement must be coded for each job step.
- A single job can have a maximum of 255 steps.

EXEC Statement Format

- The EXEC statement format is:

```
//stepname EXEC positional,keyword comments
```

- The stepname is a name given to an EXEC statement which identifies the step within a job.
- Although a stepname is optional, if a job has more than one step, then it a good practice to have it included.
- Including a stepname will help in interpreting output listings for a job.
- When coding a stepname, follow the naming rules stated previously.
- Stepnames should be unique within a job.

PGM Parameter

- The PGM parameter is a positional parameter that specifies the program to be executed.
  - The PGM parameter informs the z/OS system which program to execute for this step.
  - Even though it contains a keyword, the PGM parameter is positional and must be the first parameter on an EXEC statement.
- The format of the PGM parameter is:

```
PGM=parameter name
```

- The program name is the name of the load module or program which is to be executed.

```
//STEP4   EXEC   PGM=UPDATE
```


ACCT Parameter

- Some installations keep track of accounting information for job steps instead of the entire job.
  - The ACCT parameter allows accounting information to be specified for the job step.
- The ACCT parameter on the EXEC statement is used for specifying job step accounting information:

  ACCT=accounting information
- Accounting information is specific to a z/OS installation.

```
//STEP2 EXEC PGM=TESTPAY,
//  ACCT=(3862,`T/24',#AV)
```


ACCT Parameter

- Accounting information can be specified for the step either instead of or in addition to the job.
- When used in addition to job accounting information, the value specified for ACCT overrides the job information for that step only.
- The value specified for the job applies to all other steps in the job.

DD Statement

- All programs use files.
  - Files are read, created or updated as a program executes.
- The DD or Data Definition statements are used for describing each file that a program uses.
- The DD statement is used by the operating system to locate the file.
- "Where is the input located?" and "Where should it put the output?" are questions which DD statements answer.

DD Statement Format

- DD statements are associated with a specific program or job step.
  - Since an EXEC statement begins a job step, the DD statements for one step must be included before the next EXEC statement or the end of the job.
- The format of a DD statement is:

```
//ddname DD positional,keyword  comments
```


ddname Field

- The ddname identifies a DD statement.
  - It follows the standard JCL naming rules.
  - The ddname should be unique within a step; however, it can be repeated in other steps.
- The ddname is the link between a program and the operating system.
  - Each programming language has its own method of identifying the files a program uses.
  - All languages specify an "external name" or ddname.

ddname Field

- z/OS uses this "external name" to associate the file used by a program with a dataset described by a DD statement.
  - This two-step approach provides a great deal of flexibility in assigning files.
    - By changing the DD parameter, an output file can be directed to the printer, punch, tape, or disk.
    - It will not be necessary to change and recompile the program in order to redirect its output.

ddname Field

- Different languages can specify the ddname on a z/OS operating system with the following JCL:

```
//PRTSTEP   EXEC   PGM=PRNTCARD
//CARDIN    DD     ...
//PRINTOUT  DD     ...
```


ddname Field

COBOL programming language

- The files need to be defined in the Environment Division:

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT CARD-FILE       ASSIGN TO CARDIN
   SELECT PRINTER-OUTPUT  ASSIGN TO PRINTOUT
```

- A COBOL program uses two files.
  - The ASSIGN clause specifies the ddnames, CARDIN and PRINTOUT.
  - z/OS uses the ddnames to locate the datasets for the program.

ddname Field

Assembler programming language

- In Assembler language, the ddname is an operand of the DCB macro.
- The equivalent Assembler statements are:

```
CARDFILE  DCB   DDNAME=CARDIN ...
PRINTER   DCB   DDNAME=PRINTOUT ...
```

CETi/COMPUTER EDUCATION TECHNIQUES, INC.

1-2: 37