Structured COBOL Workshop for Enterprise COBOL

Structured COBOL Workshop for Enterprise COBOL - Course Objectives

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

1. Code and test programs using the "IBM Enterprise COBOL for z/OS" compiler to process sequential files
2. Describe fields, records, and files to COBOL
3. Correctly use the most common COBOL verbs in their various forms
4. Use the following techniques in designing or coding COBOL programs:

   Data editing, including use of multiple currency symbols and the Euro
   Loop control and switch setting and testing
   Move mode and locate mode processing
   Pseudocoding as a design tool
   Reference modification
   Some intrinsic functions
   The COBOL COPY statement
5. Code COBOL programs using installation standards, with an awareness of the ANSI standard
6. Define numeric data items to COBOL that are packed decimal or binary integer in format
7. Use the COBOL arithmetic verbs ADD, SUBTRACT, MULTIPLY, DIVIDE, and COMPUTE
8. Code and test COBOL programs to create reports, including page break processing and control breaks.
9. Code and test COBOL programs to perform batch transaction processing using match-merge logic (sequential processing of transaction and master files), including update in place for sequential disk files
10. Use the following techniques in designing or coding COBOL programs:

    Top down development
    Structured programming
    Pseudocoding as a design tool
    Modular design
11. Code COBOL programs that read from and write to zFS files on systems using z/OS UNIX.

Structured COBOL Workshop for Enterprise COBOL - Topical Outline

Day One

Day Two

Day Three

Day Three, continued

Data Alignment
- Slack Bytes and Sync
- Numeric Data Transmission Considerations
- Computer Exercise: Ensuring Proper Alignment 253

Arithmetic Instructions
- ADD, SUBTRACT, MULTIPLY, DIVIDE
- Rounding
- Arithmetic expressions
- COMPUTE
- Planning calculation results
- SIZE ERROR Condition
- Computer Exercise: Using Arithmetic Verbs 284

EVALUATE
- Syntax
- EVALUATE and conditions
- EVALUATE with ANY and ALSO
- EVALUATE and truth tables
- Points and Tips
- Computer Exercise: Using EVALUATE 301

Day Four

Basic String Manipulation
- INITIALIZE, ACCEPT / DISPLAY
- Conceptual Data Items (DATE [YYYYMMDD], DAY [YYYYDDD], DAY-OF-WEEK, TIME)
- Reference Modification
- Hex Notation
- Computer Exercise: DATE, TIME, and DISPLAY 317

A Summary of COBOL Releases

Compiler(s)	ISO/ANSI standards	End of Support
OS/VS COBOL	68 / 74	June 1994
VS COBOL II - R1, R2	74	June 1996
VS COBOL II - R3, R4	74 / 85	March 2001
COBOL/370 (V1R1) COBOL for VSE/ESA	74 / 85+89	September 1997
COBOL for MVS & VM (V1R2)	74 / 85+89 +OO	December 2001
COBOL for OS/390 & VM (V2R1)	74 / 85+89 +OO'	December 2004
COBOL for OS/390 & VM (V2R2)	74 / 85+89 +OO'	December 2004
IBM Enterprise COBOL for z/OS and OS/390 (V3R1) V2R2 + Java, Unicode, XML parse		April 2004
IBM Enterprise COBOL for z/OS and OS/390 (V3R2) V3R1 + enhanced OO capabilities		October 2005
IBM Enterprise COBOL for z/OS (V3R3) V3R2 + XML generate		April 2007
IBM Enterprise COBOL for z/OS (V3R4) V3R3 + Larger element size + Unicode support stage 2		April 2015
IBM Enterprise COBOL for z/OS (V4R1) V3R4 + enhanced XML support + compiler parms in file		April 2014
IBM Enterprise COBOL for z/OS (V4R2) V4R1 + underscore in names + XML PARSE with validation		April 2022
IBM Enterprise COBOL for z/OS (V5.1) 02 improved Java & C interoperability XML and Web services		April 2020
IBM Enterprise COBOL for z/OS (V6.1) support 64-bit programming		September 2022
IBM Enterprise COBOL for z/OS (V6.2) improvements in Java, XML, Web		N/A
IBM Enterprise COBOL for z/OS (V6.3) support for new LE features		N/A
IBM Enterprise COBOL for z/OS (V6.4) interoperability with 31-bit & 64-bit		N/A

Section Preview

❒ **Fundamentals**

Hardware and Software

Instructions and Programs

Compiling and Binding

COBOL Basics

✗ A Sample Program

✗ Character Set, Words, Punctuation

✗ Program Structure

✗ Identification Division

✗ Environment Division

✗ Data Division

Starting a COBOL Program (Machine Exercise)

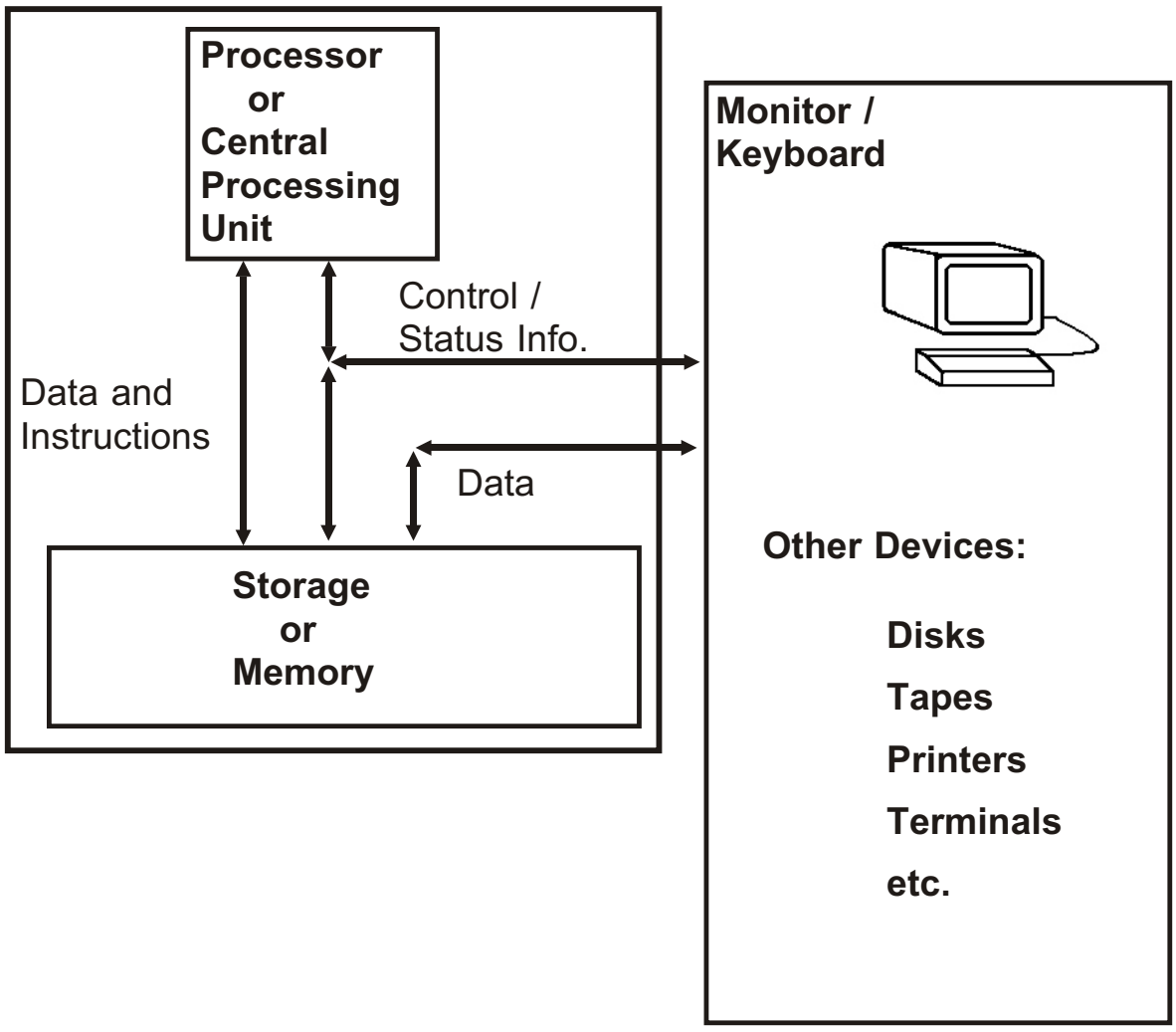
Computer Hardware and Software

❒ **Computers have two broad categories of components**

Hardware (actual machines)

Software (programs and data)

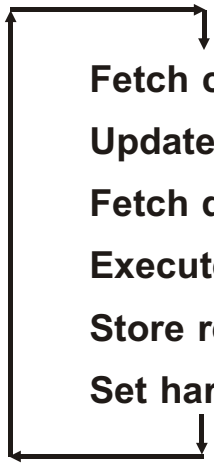
❒ **Although details vary from machine to machine, the essential hardware components are:**



The Processor

❒ **The Central Processor (or CP, or Processor, or Central Processing Unit, or CPU) works as follows (conceptually):**

Instruction Execution



Fetch current instruction from memory into CPU

Update instruction pointer to next instruction in memory

Fetch data item(s) referenced in instruction, from memory

Execute instruction

Store result(s) in memory

Set hardware condition code

External Device Control

Send commands to device (for example, input or output request)

- ✗ data transferred from memory to device (output request)
- ✗ or data transferred from device to memory (input request)

Device signals processor of completion of I/O operation (successful or failure)

Processor may also query device regarding status, to check if powered on, connected, working properly, etc.

❒ **Notice that data flows between devices and memory and between the CPU and memory; control flows between the CPU and devices**

Instructions

❒ **The CPU can only process the machine instructions it was designed to recognize**

Electronic patterns that indicate the operation to be performed (arithmetic, moving data in memory, comparing data, branching, and so on)

Instructions also indicate where the data item(s) to be operated on are found in memory: the memory address of the item(s)

Instructions need to be in memory before they can be executed

The data that instructions operate on must already be in memory at the time the instruction is executed

✗ Except for input instructions, of course, since these are requests to read data into memory from some external device

✗ However, these kinds of instructions must identify where in memory to put the data (what memory address to use)

Programs

- **A program is a list of machine instructions the CPU is to execute to accomplish some task**

  Along with any necessary data embedded in the program

- **Some programs are designed to help run the computer itself**

  These belong to the operating system

- **Some programs are designed to accomplish useful work, such as updating records on external files, producing reports, or displaying data on a terminal**

  These are called application programs

- **Some programs are designed to convert instructions keyed as character strings into actual machine instructions**

  These programs are called compilers

Programming Languages

❒ **An unknown number of programming languages have been devised over the years**

Each one trying to provide a natural style for programmers to describe what they want the computer to do

The most commonly used programming languages today (in no particular order):

- ✗ COBOL
- ✗ PL/I
- ✗ Assembler
- ✗ JAVA
- ✗ C
- ✗ C++
- ✗ FORTRAN
- ✗ RPG
- ✗ Pascal
- ✗ Python
- ✗ php

Here, of course, we focus on COBOL

Compiling and Binding

- A compiler is a program that examines a program written in a particular language (COBOL in our case) and converts this program into machine instructions the CPU can understand

  A program typically also contains support data, such as report headers, areas for doing calculations, and so on

- A program written in a programming language is called a source module (or also a "source program" or just "source code")

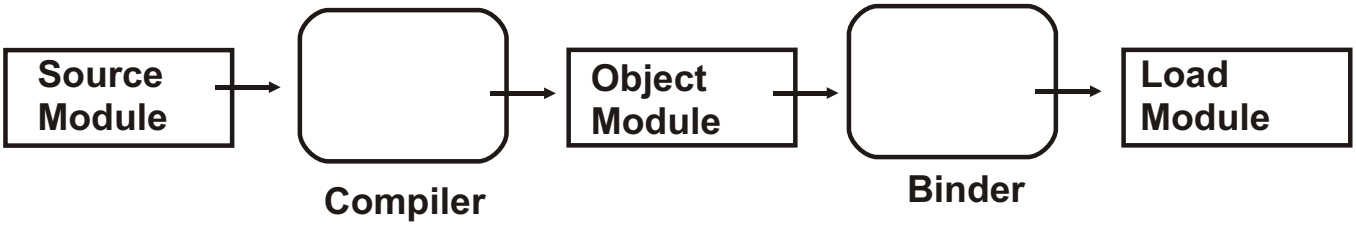
- A compiler reads source code and produces an object module (or “object code”, or “object program”)

  This is the resulting machine instructions and support data, based on the rules of the language and the source code written by the programmer

- An object module must be bound before it can be run

  Binding (or linkage editing) produces a load module (or “executable program”, or “program object”)

  A load module is stored in a special format on a disk library for fast loading into memory when the program is to be run



COBOL

COmmon Business Oriented Language

- The world's most successful, pervasive programming language

- Standards of the language are set and modified by ISO - the International Standards Organization

  U.S. member of this group: American National Standards Institute (ANSI)

- Most recent versions:

  COBOL 68

  COBOL 74

  COBOL 85

  COBOL 85 with 1989 amendment: Intrinsic Functions

  COBOL 2002

  COBOL 2014

- Despite efforts to create a truly "portable" language, each vendor complies with various levels of the official standards and then adds its own enhancements

Standards

- **There are various sets of standards established by different organizations to define what they consider to be "official", or "standard" COBOL; among them:**

  The ISO/ANSI Standard

    Compilers from different vendors must be certified independently by ISO or ANSI

  FIPS (Federal Information Processing Standard)

    For U.S. government installations

  Installation-specified coding standards

- **These course materials are based on the IBM compiler called IBM Enterprise COBOL for z/OS**

    And any later COBOL compilers for the z/OS platform

- **Discussion of variations from the official standards is marginal here**

    We focus on the compiler as it is, not how it compares to the standards

A Sample COBOL Program

```
Identification division.
Program-id. ISDF2F.

Environment division.
Input-output section.
File-control.
    Select INFILE assign to INDD.
    Select OUTFILE assign to OUTDD.

Data division.
File section.

FD  INFILE
    Block contains 0 records.
01  INREC                        PICTURE   X(128).

FD  OUTFILE
    Block contains 0 records.
01  OUTREC                       PIC       X(128).

Working-storage section.
01  Record-work                  pic       x(128).

Procedure division.
Initialization section.
    Open input INFILE
         output OUTFILE.

Copyfile section.
    Read INFILE into record-work
                 at end go to termination.
    Write outrec from record-work
    Go to copyfile.

Termination section.
    Close INFILE OUTFILE.
    Stop Run.
```

Components of the COBOL Language

Basic Character Set

- **52 Alphabetic characters, (A-Z, a-z), 10 Numeric digits (0-9), and 17 Special characters:**

	Space
.	Decimal point (period)
<	Less than
(	Left parenthesis
+	Plus sign
$	Dollar sign
*	Asterisk
)	Right parenthesis
;	Semicolon
:	Colon
-	Minus sign (hyphen)
_	Underscore (in Enterprise COBOL 4.2 or later)
/	Slash or stroke
,	Comma
>	Greater than
=	Equals sign
"	Double quote
'	Single quote / apostrophe

- **Older COBOL programs are mostly written in upper case letters (all capitals)**

- **Newer programs tend to be written in lower case or mixed case letters**

- **This is mostly a matter of style and personal preference, since the compiler treats upper case and lower case letters the same (except when enclosed in quotes)**

Elementary Uses of the Character Set

Punctuation characters: Space . ( ) ; , : '

A separator is a contiguous string of one or more punctuation characters

✗ In particular, the following are designated as COBOL separators:

b	**Space**
,b	**Comma**
.b	**Period**
;b	**Semicolon**
(	**Left parentheses**
)	**Right Parentheses**
:	**Colon**
"b	**Quote or**
'b	**Apostrophe**
==	**Pseudo-text delimiter**
x' or x"	**Start hexadecimal literal**
z' or z"	**Start null-terminated literal**
n' or n"	**Start DBCS or national literal**
g' or g"	**Start DBCS literal**
nx' or nx"	**Start hexadecimal national literal**

Note that the x, z, n, and g characters above can be upper case or lower case

Elementary Uses of the Character Set, 2

Separators, continued

Quotes, apostrophes, and pseudo-text delimiters must always occur in pairs, and the first one must be preceded by at least one blank while the second must be followed by at least one blank

Hexadecimal literals, null-terminated literals, DBCS literals. national literals, and national hexadecimal literals must only contain the allowed character types and must be terminated with a quote or apostrophe (whichever is used for the opening)

❒ **A character string is a single character or sequence of contiguous characters that forms a word, literal, picture character string, or comment**

✗ A character string is delimited by a separator

❒ **Note that literals and run-time data can include characters other than the basic character set**

The basic character set is what is used to compose COBOL recognized words and names

Elementary Uses of the Character Set, 3

COBOL words (30 characters maximum):

- **User-defined words**

  Supplied by (made up by) programmer

  Contains alphanumeric characters, hyphens, and underscores

  Hyphen may not be first or last character; underscore may not be the first, but it may be the last

  Must contain at least one alpha character (except paragraph names and section names)

  Must not be COBOL reserved word

- **System names (*e.g.*: IBM-370, SYSIN)**

- **Function names (*e.g.*: CURRENT-DATE)**

- **Reserved words**

  Key Words (ADD, READ, WRITE, ...)

  Optional Words (IS, ARE, ...)

  Special Registers (LINAGE-COUNTER, TALLY, ...)

  Special Character Words (+ - / * ** < > = <= >=)

  Figurative Constants (ZERO, SPACES, ...)

  Special Object Identifiers (SELF, SUPER)

Elementary Uses of the Character Set, 4

Literals

- **Numeric: 0-9,+,-, decimal point (no commas); max 18 digits**
- **Non-numeric: in quotes or apostrophes; max 160 characters**
- **Hexadecimal: inside x'...' or x"..."; only hexadecimal characters (0-9, A-F, or a-f); max 320 characters (160 bytes)**
- **Null-terminated: inside z'...' or z"..." ; max 159 characters; COBOL appends a null character (x'00')**
- **DBCS: inside g'...' or g"..." or n'...' or n"..."; max 28 characters**
- **National: inside n'...' or n"..."; max 80 characters**

Picture Character Strings

- **Used to describe data items, for example: PIC x(20)**
- **Used to describe desired editing, for example: PIC ZZ,ZZ9.99**

Comments

- **A line with an asterisk (*) or slash (/) in column seven (everything else in the line is considered to be the comment)**

  Using a slash in column seven will also start a new page of your COBOL compile listing output

  The string *> in a line indicates all following text is a comment (an "inline comment" [IBM COBOL V5 or later])

Other Characters Supported

❒ **In addition to the basic character set, the IBM COBOL compiler supports the following characters / character sets**

<u>DBCS</u> - Double Byte Character Set; strings of characters with each character consisting of two bytes

- ✗ Delimited by shift-in and shift-out characters (X'0E' and X'0F', respectively)
- ✗ In the range X'41' to X'FE' for each byte
- ✗ Can be used to create COBOL words (for example, data item names or paragraph labels) (maximum of 14 characters [28 bytes] plust the shift-in and shift-out delimiters)
  - ➢ Some restrictions: cannot be used for program names, object oriented class names, and a few other places
- ✗ Can also be used in literals, comments, and picture strings
  - ➢ Some restrictions for literals, discussed as encountered

Other Characters Supported, continued

❒ Other character sets supported

Unicode **- single standard to encode characters from all human languages, plus many specialized symbols**

- ✗ Although there are several variations, this compiler supports the Unicode version called UTF-16 in which most Unicode characters are two bytes but some may be composed of two two-byte pairs (called surrogate pairs)
  - ➢ In some cases a character is composed of one or more Unicode characters and one or more combining units (so a single character may take four bytes or more, in increments of 2 bytes)
  - ➢ Thus, for Unicode, the term "character" encompasses 2 bytes (or more, in increments of 2 bytes)
  - ➢ It is the programmer's responsibility to ensure Unicode characters are not split as a result of MOVEs or other COBOL instructions
- ✗ Called "National" characters in this compiler
- ✗ Cannot be used for COBOL words, but may be used in literals and run-time data

Compiler Options

❒ **In addition to COBOL statements, your COBOL source program may include directives to the compiler itself**

The first we will mention are compiler options - telling the compiler various ways to look at the source, produce object code, format the compile listing, and so on

Default compiler options are specified when the compiler is installed

Although we won't go into details in this course, most default options can be overridden

- ✗ In the JCL that requests the compile and / or
- ✗ In PROCESS statements in your source program

❒ **From time to time in this course, we will mention when a compiler option, or other directive, impacts how the compiler interprets what you code in your COBOL source statements**

Our first example is the compiler option NSYMBOL; this option has one of two values to choose from

- ✗ NSYMBOL(DBCS) - the IBM-supplied default; says literals coded using n'...' or N'...', n"...", or N"..." represent DBCS values
- ✗ NSYMBOL(NATIONAL) - says literals coded using n'...' or N'...', n"...", or N"..." represent NATIONAL values (UTF-16)

Quotes and Apostrophes

❒ **A matched set of (single or double) quotes is used to delimit non-numeric literals**

E.g.:

```
'Inventory Report'

"Customer name"
```

You should use only one of these for delimiting literals throughout any given program

✗ You may mix them, as long as each starting delimiter has a corresponding closing delimiter of the same kind (both ' or both " for any given literal)

Although the double quote is the ISO/ANSI standard, most IBM installations use the single quote (apostrophe)

✗ For this reason, we'll use the apostrophe for delimiting non-numeric literals in this course

Quotes and Apostrophes, 2

- **If a quoted string is to contain a quote, the contained quote is represented by two consecutive quotes:**

```
'Larry''s News Shop'
```

- **In the latest standard, you may mix and match quotes and apostrophes in a single program:**

  Include some literals bounded by apostrophes

  Include some literals bounded by quotes

  As long as the opening character is the same as the closing character for any one non-numeric literal string

- **Comments and non-numeric literals may contain any character; COBOL does not try to process these items**

Overall Structure of a COBOL Program

IDENTIFICATION DIVISION
- **Paragraphs**
  - **Entries**
    - **Clauses**

ENVIRONMENT DIVISION
- **Sections**
  - **Paragraphs**
    - **Entries**
      - **Clauses**
        - **Phrases**

DATA DIVISION
- **Sections**
  - **Entries**
    - **Clauses**
      - **Phrases**

PROCEDURE DIVISION
- **Sections**
  - **Paragraphs**
    - **Sentences**
      - **Statements**
        - **Phrases**

❒ **That is, each program is made up of divisions, which are further sub-divided as shown**

❒ **Comments may be placed anywhere in the program, as may blank lines**

Area-A and Area-B

❒ **Lines in a COBOL program have various areas the compiler is sensitive to:**

Columns	Use
1 — 6	Sequence numbers (optional)
7 — 7	Continuation column (for continuation indications and comment indications)
8 — 11	Area-A
12 — 72	Area-B
73 — 80	Program name (optional)

❒ **Area-A is used for: division, section, and paragraph headers (the first line in each of these entities) as well as for the first level in a data structure (or level 77 items)**

That is, each of these objects must begin in Area-A

In addition, "end program", "end class", and "end method" headers, DECLARATIVES, and END DECLARATIVES statements must all begin in Area-A (none of these are discussed in this course)

❒ **Area-B is used for all other components**

Identification Division

```
Identification division.
Program-id. INVENTORY.
```

Notes

The PROGRAM-ID ('INVENTORY' in this case) is the name the program will be known by externally (that is, when you want to run this program, or if another program calls this program)

✗ This is an example of an external name

Only the first eight characters of the program name ('INVENTOR' in our example) will be used, so it's really best to use only names that are eight characters or fewer (then no surprises later on)

Also, external names should not contain a hyphen or start with a numeric digit, and lower case letters will be folded to upper case when used by the system; cannot use DBCS

Example

```
 Identification division.
 Program-id. Inpupda.
* You may use comment lines ('*' in column 7)
* to augment these entries according to
* installation standards
```

Note that earlier versions of COBOL supported a variety of additional paragraphs in this division (AUTHOR, DATE-COMPILED, etc.); now we generally just use comments

Programs and Files

❒ **Most COBOL programs process data stored in external files, files that are usually stored on tape or disk (although a report is also a file)**

In this course we focus on sequential files, but COBOL can work with a variety of file types, including indexed files and relational data bases

❒ **COBOL processes external files as input, output, or i-o; the type of processing to be done is specified at OPEN time**

If a file is OPENed as an input file, the COBOL program issues READ **statements to retrieve records from tape or disk and transfer the data into memory for processing**

If a file is OPENed as an output file, the COBOL program builds a record in memory and then issues WRITE **statements to send the data to tape, disk, or a report**

Files OPENed as i-o may have both input and output operations:

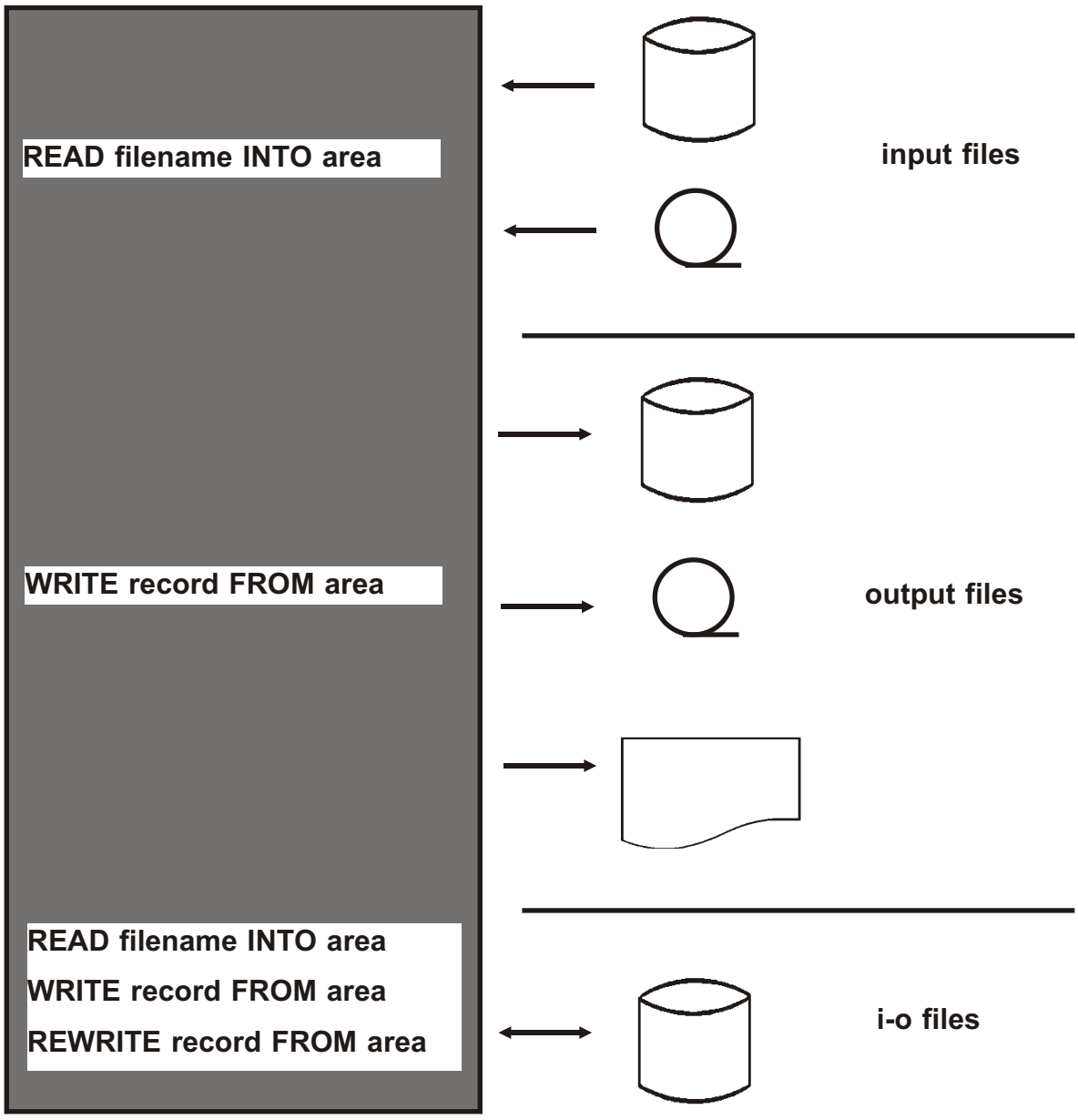
✗ A READ statement retrieves an existing data record from tape or disk into memory

✗ A WRITE statement adds a record to the external file from memory

✗ A REWRITE statement updates a record on a disk file with new information from memory

Programs and Files, continued

❒ **Pictorially, we have something like this, when the program is run:**



❒ **In your COBOL program, you identify the files you will be working with in the Environment Division ...**

Environment Division

```
Environment division.
Input-Output section.
File-control.
    Select Cardin assign to CARDS.
    Select Listing assign to REPORT1.
    Select Transact assign to TRANSACT.
    Select Master assign to master
                        file status is master-stat.
.
.
```

- ❒ **Notice the division header, the section header, and the paragraph header**

  This will help you get the feel for the structure and organization of this part of the program

- ❒ **The word immediately following 'Select' is the 'file name', or the name you use inside your program to reference the file (for example in 'open' and 'read' statements)**

- ❒ **The word immediately following 'assign to' is the name you will use at run time to relate each file name to a particular external file**

- ❒ **The 'file status' name is a data item you define in working-storage as a tool for checking the outcome of I/O requests; more on this later**

Identifying External Files

❒ **The object of the ASSIGN TO clause must follow the rules for an external name, and is called the "ddname"**

At run time, JCL is used to connect each ddname to an actual, external file using a JCL DD statement:

```
//ddname    DD  DSNAME=actual-file-name,. . .
```

❒ **COBOL can also access files residing in a zFS (z/OS File System) - files supported as part of z/OS UNIX System Services**

In the SELECT statement, specify ORGANIZATION IS LINE SEQUENTIAL

The object of the ASSIGN TO clause is a ddname if you supply a DD statement at runtime

✗ This statement must contain at least PATH= and FILEDATA=TEXT

✗ It may also contain PATHOPTS, PATHMODE, PATHDISP

If no DD statement is supplied at run time, OPEN will treat the ddname as an environment variable name

✗ In this case, you must export a path name into this variable before you OPEN the file (details not discussed in this course)

Data Division

❒ **The Data division contains one to four sections**

The File Section.

✗ Used for file / record descriptions

The Working-storage Section.

✗ Describes constants, counters, tables, other data elements and structures

The Local-storage Section.

✗ Contains data elements that are dynamically created each time the program gets control and deleted when the program returns

The Linkage Section.

✗ Describes items passed from other programs by 'CALL'

❒ **We'll only use the first two of these sections in this course**

File Section Examples

```
Data division.
File section.
FD  CARDIN.
   -----Record description(s).


FD  LISTING.
   -----Record description(s).


FD  TRANSACT
    block contains 0 records.
   -----Record description(s).


FD  MASTER
    block contains 0 records.
   -----Record description(s).
```

Notes

- **Watch punctuation and Area-A, Area-B margins**

- **Names after 'FD' must match file names in Select statements ('FD' stands for File Definition)**

- **'Block contains' clause is optional, but if not coded implies 'block contains 1 records' (The phrase must be 'block contains 1 records')**

  Generally code 'block contains 0 records' for output files, since the operating system calculates block size

  Omit 'block contains' for input files (system will get from the label) and for print files (operating system blocks them in a special way automatically)

File Section Notes

- **To describe a record, you'll need to supply a record name, a picture of the data, and something called a level number**

- **The compiler also accepts a RECORDING MODE clause**

  It is optional, but without it you get a warning message at compile time

- **For this course, you may omit the RECORDING MODE clause, or you may code "recording mode is F" to eliminate the warning messages**

  ('F' says your records are fixed length records)

```
FD  CARDIN
    recording mode is F.
   -----Record description(s).

FD  LISTING
    recording F.
   -----Record description(s).

FD  TRANSACT
    recording mode is f.
   -----Record description(s).

FD  MASTER
    recording mode f.
   -----Record description(s).
```

Note that "mode" and "is" are optional words

Record Descriptions

- We'll be looking at all the alternatives for describing data later in the course

- For right now, it's sufficient to describe a record by giving it a name, using a level number of 01, and indicating how large the record is

- In our example we might code:

```
Data division.
File section.
FD  CARDIN
    recording mode is F.
01  Card-rec            pic  x(80).

FD  LISTING
    recording F.
01  Report-rec          pic  x(100).

FD  TRANSACT
    recording mode is f.
01  Trans-rec           pic  x(60).

FD  MASTER
    recording mode f.
01  Master-rec          pic  x(320).
```

- So in the Data division, File section, we have an FD entry for every file

  And each FD entry must be followed with an entry that gives a name to records in the file and that indicates how large records in the file are

Recap So Far

- ❒ **We have looked at the fundamentals of COBOL**
  - **Character set, punctuation, words**
  - **Area-A and Area-B**
  - **Structure of a program (divisions, sections, paragraphs, etc.)**
- ❒ **We looked at the first two divisions as thoroughly as needed for the content in this course**
- ❒ **We have examined the File section of the Data division**
  - **We have not yet examined how to define data structures or individual data items**

Recap, continued

- **The roles of the various divisions are:**

  Identification Division

  Specify program-id

  May contain comments to describe function and history of program

  Environment Division

  Identify file names and associate internal file names to external file names

  Data Division

  File Section: Specify file and record characteristics (data outside the program: external files)

  Working-storage Section: Specify data items in the program

  Procedure Division

  Specify the instructions to use to operate on the data

- **We still need to learn how to specify data items, both external and internal; but first ...**

Computer Exercise: Starting a COBOL Program

If you need a logon id, this is the time to get one from the instructor. If you need any help getting going, this is a good time to get it.

Run the rexx exec called D715STRT; this creates two libraries for you:

<userid>.TR.CNTL - contains JCL you will need to compile, link, and test the labs

<userid>.TR.COBOL - contains some starter code for later; this is where you will code all your programs

To run the exec, use ISPF 6 (command); and key in the following:

```
===> ex   '__________.train.library(d715strt)'   exec
```

and press <Enter>

This will run the rexx exec; D715STRT prompts you for a high level qualifier to use for the data set names mentioned above, defaulting to your TSO id; this is normally fine, so just press <Enter>. You should see a screen telling you the setup was successful.

This exercise is designed to get a COBOL program started. We haven't covered enough to code a complete program yet, but we can write code that will provide a basis for future work.

This is also an opportunity to check out logon id's, the system editor, and other preliminary issues that can distract us later.

*** more ***

Computer Exercise: Starting a COBOL Program, p.2.

Create a program, call it EXER01 (how original) in your TR.COBOL library. This should be both the file or member name and the program-id name.

Code the **identification division**, the **environment division**, and the **file section** of the **data division**.

> Specify two files in your program. One will describe an existing inventory file and the other will eventually be a report file. Records in the inventory file are 100 bytes long. Records in the report file will be 106 bytes long.
>
> Use an fd name of INVNTRY for the input file and REPRT for the report file.
>
> Eventually, your program will read in records from the inventory file and write them out to the report file.

Do not code any options we have not discussed yet. Keep it simple.

You can compile your code, EXER01 using member COBSUBC in your TR.CNTL library; in the third line from the bottom, change the SET instruction to name your program; e.g.:

```
//        SET  O=EXER01
```

Then submit the job; fix any compiler errors and repeat until the code compiles clean.

Note: this job will try to compile, and bind, Right now we only care about the compile part. This exercise is just to get you going for later labs.

Then take a short break and we'll pick it up again.