

COBOL

Lesson 00



Course Goals and Non Goals



➤ Course Goals

- To write COBOL programs efficiently
- To understand File Handling concepts
- To Print reports and statistics.

Pre-requisites



- Knowledge of MVS,JCL

Intended Audience



➤ Programmers

Day Wise Schedule



➤ Day 1

Lesson 1: Introduction to Cobol

- 1.1: Introduction
- 1.2: Features of COBOL
- 1.3: Coding a COBOL program
- 1.4: COBOL character set
- 1.5: COBOL words
- 1.6: Structure of COBOL programs
- 1.7: Divisions in COBOL
- 1.8: COBOL coding rules



Day Wise Schedule

➤ Day 1 (contd...)

Lesson 2: Working with data

- 2.1: Forming Data Names
- 2.2: Types of data
- 2.3: VALUE Clause
- 2.4: Rules for continuation of Literals
- 2.5: Special Characters
- 2.6: How data is stored in memory
- 2.7: USAGE clause
- 2.8: Usage is display
- 2.9: Binary,Comp,Comp-4
- 2.10: REDEFINES Clause
- 2.11: RENAME clause
- 2.12: Justified Clause
- 2.13: Sign Clause
- 2.14: Qualification of names



Day Wise Schedule

➤ Day 1 (Contd..)

Lesson 3: ACCEPT/DISPLAY verbs

3.1: ACCEPT verb

3.2: DISPLAY verb

3.3: STOP RUN

3.4: GoBack

➤ Day 2

Lesson 4: MOVE statements

4.1: Introduction - MOVE

4.2: Types of MOVE

4.3: Move Corresponding



Day Wise Schedule

➤ Day 2 (contd...)

Lesson 5: Arithmetic verbs

- 5.1: Arithmetic verbs
- 5.2: More about Arithmetic verbs
- 5.3: Compute verb
- 5.4: Initialize clause

Lesson 6: Decision Making

- 6.1: COBOL statements
- 6.2: Relational Operators
- 6.3: CONTINUE clause
- 6.4: NEXT SENTENCE clause
- 6.5: Classification of conditions
- 6.6: EVALUATE Statement

Day Wise Schedule



➤ Day 3

Lesson 7: Iterations

- 7.1: Looping statements
- 7.2: Inline Perform
- 7.3: Perform...Test
- 7.4: EXIT statement
- 7.5: GO TO statement



Day Wise Schedule

➤ Day 4

Lesson 8: File Handling (Sequential Files)

- 8.1: Introduction
- 8.2: How Files are processed
- 8.3: Buffers
- 8.4: File operations
- 8.5: File Organization method
- 8.6: Making entries for a file in a program
- 8.7: Input/Output verbs
- 8.8: Access Mode
- 8.9: Collating Sequence
- 8.10: COPY statement
- 8.11: How copy works
- 8.12: How replacing works



Day Wise Schedule

➤ Day 4 (Contd..)

Lesson 09: Sorting

9.1: SORT Statement

Lesson 10: Trapping Runtime Errors

10.1: Trapping runtime errors

➤ Day 5

Lesson 11: Report Using Control Break Processing

11.1: Simple Reports without control break

11.2: Control break processing

Day Wise Schedule



➤ Day 6

Lesson 12: Indexed And Relative File

- 12.1: Indexed Files
- 12.3: READ Statement
- 12.4: DELETE Statement
- 12.5: REWRITE Statement
- 12.6: To Add New Record
- 12.7: ALTERNATE RECORD KEY
- 12.8: START Statement



Day Wise Schedule

➤ Day 7

Lesson 12: Indexed File (Contd..)

12.9: File Updating

Lesson 12: Relative Files

12.10: Read, Write, Rewrite, Delete, Start Verb

➤ Day 8

Lesson 13: Call statement

13.1: Introduction

13.2: Subprograms



Day Wise Schedule

➤ Day 9

Lesson 14: Table Handling

14.1: Table handling - Introduction

14.2: One dimensional

14.3: Two dimensional

14.4: Three dimensional

14.5: OCCURS Clauses

14.6: Defining Array

14.7: Accessing Elements in Array

14.8: PERFORM ... VARYING Statements

14.9: REDEFINES clause

14.10: Multiple-Level OCCURS

14.11: Indexed table

14.12: Table Searching

Day Wise Schedule



➤ Day 10

Lesson 15: String Handling

15.1: String Handling

15.2: STRING Verb

15.3: UNSTRING Verb

15.4: INSPECT Verb



Table of Contents

➤ Lesson 1: Introduction to COBOL

- 1.1: Introduction
- 1.2: Features of COBOL
- 1.3: Coding a COBOL program
- 1.4: COBOL Character Set
- 1.5: COBOL Words
 - 1.5.1: Forming User-defined Words
- 1.6: Structure of COBOL programs
- 1.7: Divisions in COBOL
 - 1.7.1: IDENTIFICATION DIVISION
 - 1.7.2: ENVIRONMENT DIVISION
 - 1.7.3: DATA DIVISION
 - 1.7.4: PROCEDURE DIVISION
- 1.8: COBOL Coding Rules



Table of Contents

➤ **Lesson 2: Working with Data**

2.1: Forming Data Names

2.1.1: Data Names

2.1.2: Level Numbers

2.1.3: Fields

2.1.4: PICTURE Clauses

2.1.5: Working-storage Section

2.2: Types Of Data

2.3: VALUE Clause

2.4: Rules for continuation of Literals

2.5: Special Characters

2.5.1. Level Numeric Field

2.5.1.1. Fields

2.5.1.2: Implied Decimal Point



Table of Contents

2.5.2. Edited Characters

2.5.2.1. . (Decimal Point)

2.5.2.2: Z (Suppressing leading zeros)

2.5.2.3. * (Asterisk or Check Protection)

2.5.2.4: \$ (Dollar Sign)

2.5.2.5. , (Comma)

2.5.2.6: Sign Character(+ and -)

2.5.2.7. DB/CR

2.5.2.8: B (Blank)

2.5.2.9. 0 (Zero)

2.5.2.10: / (Stroke)

2.6: Types Of Data

2.7: REDEFINES Clause

2.8: RENAMEs clause

2.9: Justified clause

2.10: Sign Clause

2.11: Qualification Of Names

Table of Contents



➤ Lesson 3: ACCEPT / DISPLAY verbs

- 3.1. ACCEPT Verb
- 3.2: DISPLAY Verb
- 3.3: STOP RUN
- 3.4: GoBack

➤ Lesson 4: MOVE statements

- 4.1: Introduction - MOVE
- 4.2: Types of MOVE
 - 4.2.1: Numeric MOVE
 - 4.2.2: Non-numeric MOVE
- 4.3: Group Move
- 4.4: More about MOVE
- 4.5: Move Corresponding

Table of Contents



➤ Lesson 5: Arithmetic Verbs

5.1: Arithmetic Verbs

5.1.1: ADD

5.1.2: SUBTRACT

5.1.3: MULTIPLY

5.1.4: DIVIDE

5.1.4.1: REMAINDER Clause in DIVIDE Operation

Table of Contents



5.2: More about Arithmetic verbs

- 5.2.1: ROUNDED option

- 5.2.2: ON SIZE ERROR Option

- 5.2.3: Compute verb

- 5.2.4: Initialize verb

➤ Lesson 6: Decision Making

6.1: COBOL Statements

- 6.1.1: IF statement

6.2: Relational Operators

- 6.2.1: Condition Examples

- 6.2.2: How comparisons performance



Table of Contents

6.3: CONTINUE Clause

6.4: NEXT SENTENCE Clause

6.5: Classification of conditions

- 6.5.1: Relational condition

- 6.5.2: Sign test

- 6.5.3: Class test

 - 6.5.3.1: ALPHABETIC Class Tests

- 6.5.4: Compound condition

 - 6.5.4.1: Hierarchy rules

- 6.5.5: Condition-names

6.6: EVALUATE Statement



Table of Contents

➤ Lesson 7: Iterations

7.1: LOOPING STATEMENTS

7.1.1: Simple PERFORM Statement

7.1.2: PERFORM...THRU Statement

7.1.3: PERFORM...UNTIL Statement

7.1.4: PERFORM...TIMES statement

7.1.5: Inline Perform

7.1.6: Perform with Test

7.2: EXIT Statement

7.3: GO TO Statement



Table of Contents

➤ Lesson 8: File Handling (Sequential Files)

8.1: Introduction to Files

8.2: File Operations

8.3: File Organization Methods

8.3.1: Sequential File Organization

8.3.2: Indexed Sequential File Organization

8.3.3: Relative File Organization

8.4: Making entries for a file in a program

8.4.1: INPUT-OUTPUT SECTION

8.4.2: FILE SECTION

8.4.2.1: FD Entry

8.4.2.2: Label Records Clause

8.4.2.3: RECORD CONTAINS Clause

8.4.2.4: BLOCK CONTAINS Clause

Table of Contents



8.5: Input/Output verbs

8.5.1: OPEN Statement

8.5.2: READ Statement

8.5.3: WRITE Statement

8.5.4: REWRITE Statement

8.5.5: CLOSE Statement

8.6: COPY Statement



Table of Contents

➤ Lesson 09: Sorting

9.1: SORT Statement

9.1.1: ASCENDING, DESCENDING Key

9.1.2: Files Used in SORT

9.1.3: Multiple Sort Keys

9.1.4: Duplicate Key Values

9.1.5: Sample FILE SECTION

9.1.6: Sorting Input Procedure

9.1.7: Sorting Output Procedure

9.1.8: Merging

➤ Lesson 10: Trapping Runtime Errors

10.1: Trapping Runtime Errors

10.1.1: FILE STATUS clause



Table of Contents

➤ Lesson 11: Report Using Control Break Processing

11.1: Simple Reports without Control Break

11.2: Control Break Processing

11.2.1: Terms and Definition

11.2.2: Example

11.2.3: Steps

➤ Lesson 12: Indexed File

12.1: INDEXED FILES

12.1.1: SELECT Statement

12.1.2: Access Modes

12.1.3: RECORD KEY Clause

12.2: WRITE Statement

12.3: READ STATEMENT

12.3.1: READ ... NEXT RECORD

12.4: DELETE Statement



Table of Contents

12.5: REWRITE statement

12.6: Add New Record

12.7: ALTERNATE RECORD KEY

12.7.1: SELECT Example

12.7.2: Random Access by ALTERNATE

12.8: START statement

12.9: File Updation

12.10: Relative File

12.10.1: Read

12.10.2: Write

12.10.3: Rewrite

12.10.4: Delete

12.10.5: Start Verb



Table of Contents

- Lesson 13: Call statement
 - 13.1: Introduction
 - 13.2: Subprograms
 - 13.3: Static and Dynamic Call
 - 13.4: Cancel statement
 - 13.5: Isinitial Statement
 - 13.6: Example of call by reference
 - 13.7: Example of call by content



Table of Contents

➤ Lesson 14: Table Handling

14.1: Table Handling - Introduction

14.2: OCCURS Clauses

14.2.1: Why OCCURS Clauses Used?

14.3: Defining Array

14.4: Accessing Elements in Array

14.4.1: Subscripts

14.4.2: Examples

14.5: PERFORM ... VARYING Statements

14.6: Indexed Tables

14.6.1: SET verb

14.7: Table Searching

14.7.1: SEARCH Statement

14.7.1.1: Sequential Search

14.7.2: SEARCH ALL Statement

14.7.2.1: Limitations of SEARCH ALL

14.8: Sample Program

Table of Contents



➤ Lesson 15: String Handling

- 15.1: Examine verb
- 15.2: String Handling
- 15.3: STRING Verb
- 15.4: UNSTRING Verb
- 15.5: INSPECT Verb

➤ Lesson 16: Appendix



References

- COBOL Programming including MS-COBOL and COBOL-85 – 2nd edition; by M. K. Roy and D Ghosh Dastidar
- Information Systems through COBOL – 2nd edition; by Andreas S. Philippakis and Leonard J. Kazmier
- Structured COBOL Programming - 7th edition; by Nancy Stern and Robert A. Stern

Next Step Courses



- VSAM
- CICS
- DB2

COBOL

Lesson 01 : Introduction to COBOL





Lesson Objectives

- **In this lesson, you will learn about:**
 - History of COBOL
 - Features of COBOL
 - COBOL character set, words, structure of programs
 - Different divisions of a COBOL program

What is COBOL?



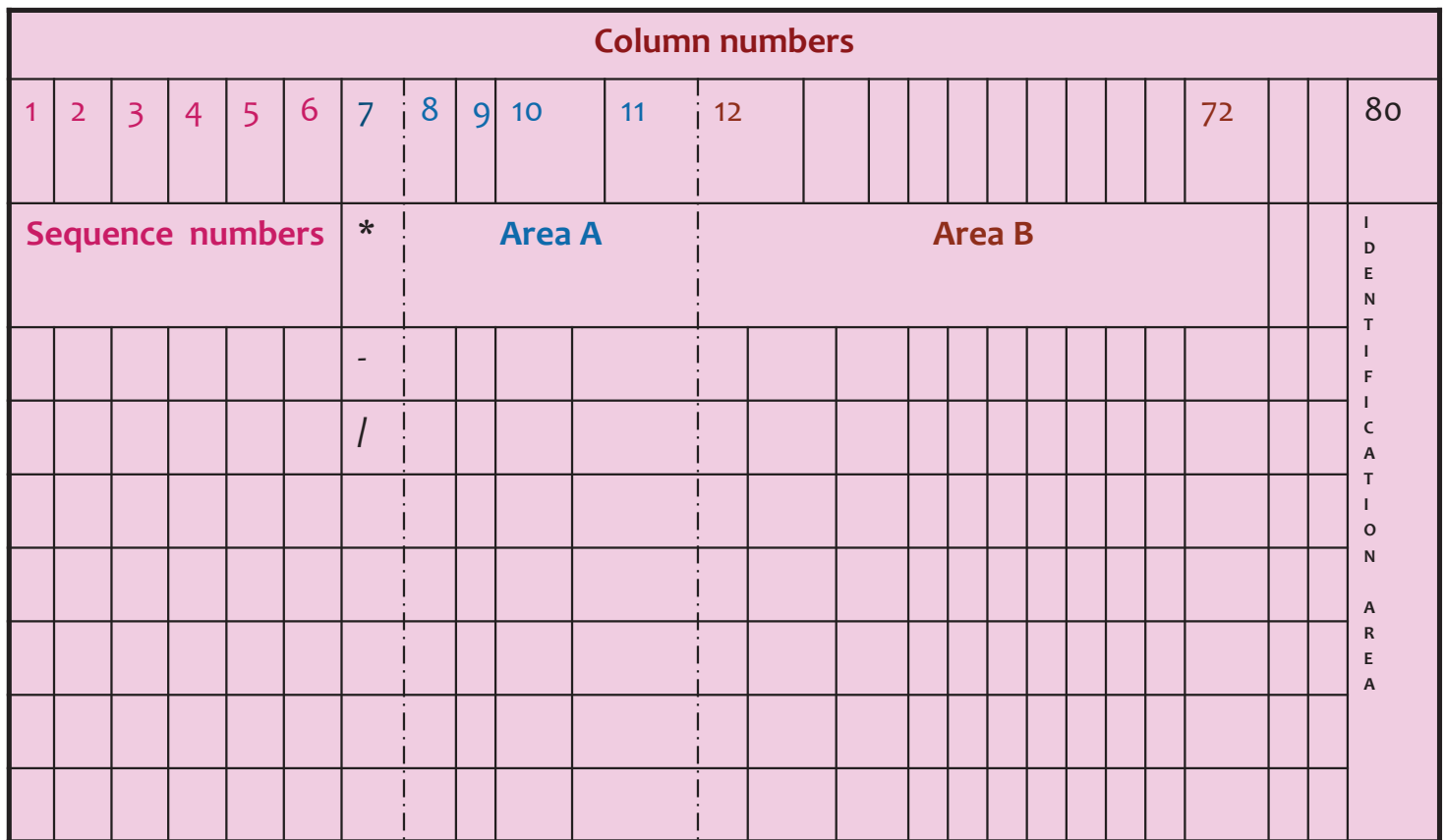
- **COBOL is the acronym of Common Business Oriented Language.**
- **It was developed by the CODASYL committee in 1959.**

Features



- **COBOL is all of the following:**
- It is a Business-Oriented Language.
 - It is a Standard Language.
 - It is an English-like Language.
 - It is Self Documenting.

Description



Character Set



- **COBOL character set consists of 52 symbols. Only these characters can be used in writing the Cobol program.**

- **The following is characters set used in COBOL:**
 - Alphabets (A-Z)
 - Numbers (0-9)
 - Punctuation (, ; . space())
 - Special characters (- = + / * > < \$)

Description



➤ **A “word” is formed using the following character**

- Digits 0-9
- Letters A-Z
- Hyphen -

➤ **There are two types of words:**

- Reserved words
- User-defined words

Description



➤ **User-defined words should follow the following norms:**

- Use 1 to 30 characters
- Use letters, digits, and hyphens (-) only
- Do not use embedded blanks
- Use at least one alphabetic character
- May not begin or end with a hyphen
- Do not use COBOL reserved words such as DATA, DIVISION

1.5: Forming User-defined Words

Example



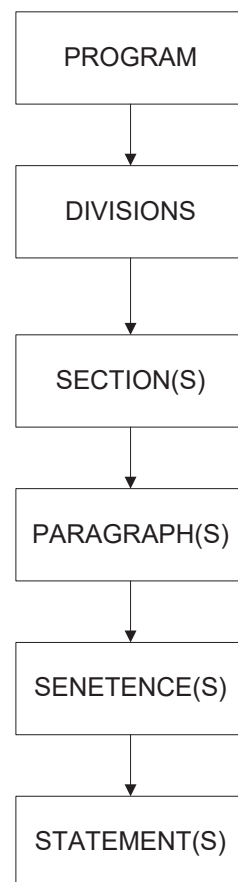
Valid data-names	Invalid
HOURS	DISCOUNT-
SALES-TOTAL	AUTHOR
SUBJECT1	BASIC+HRA
AMOUNT-OF-TRANSACTION-OUT	123

Description



➤ **A COBOL program comprises the following:**

- Character
- Word
- Clause
- Statement
- Sentence
- Paragraph
- Section
- Division



Types of COBOL Entries



➤ **DIVISIONS EXAMPLES**

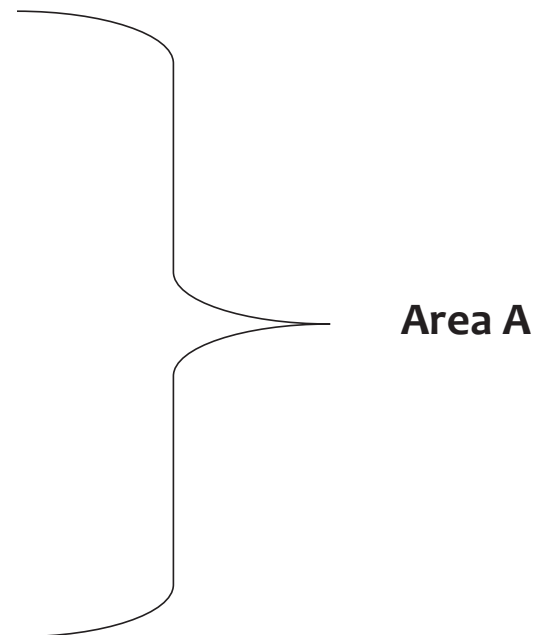
- IDENTIFICATION DIVISION.
- ENVIRONMENT DIVISION.
- DATA DIVISION.
- PROCEDURE DIVISION.

➤ **SECTION EXAMPLES**

- FILE SECTION.
- WORKING STORAGE SECTION.

➤ **PARAGRAPHS**

- 2000-CAL RTN.



1.7: Cobol Entries

Types of COBOL Entries



- **SENTENCES AND STATEMENTS.**
 - SELECT PAYROLL ASSIGN TO 'TEMP.DAT'

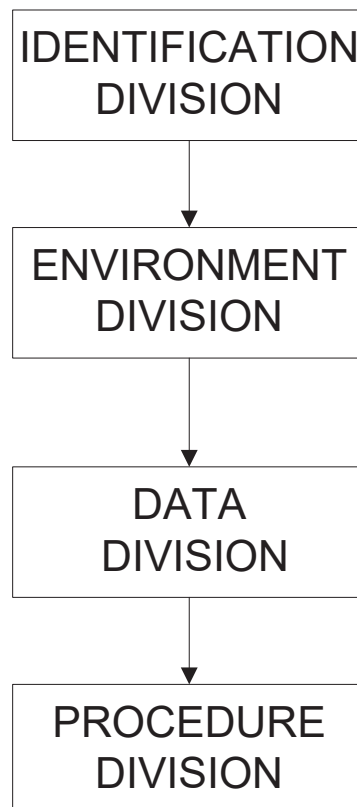


Area B

Division of a COBOL Program



- It consists of one or more paragraphs or sections.



Description



- **Identification Division:**
 - PROGRAM-ID. Program-name.
 - Mandatory
 - Identifies Program to Computer
- **AUTHOR. Author-name.**
 - Optional
- **DATE-WRITTEN. Date.**
 - Optional
- **DATE-COMPILED. Date.**
 - optional

Example



➤ Let us see an example on Identification Division:

IDENTIFICATION DIVISION.
PROGRAM-ID. SALES.
AUTHOR. XYZ.
DATE-WRITTEN. YYMMDD.
DATE-COMPILED. YYMMDD.

Description



➤ **Environment Division:**

- CONFIGURATION SECTION.
 - It indicates computer equipment to be used with the program (only serves documentation purpose)
- INPUT-OUTPUT SECTION.
 - It indicates files used by the program and the associated hardware device.

Example (Contd...)



➤ Let us see an example on Environment Division:

```
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SOURCE-COMPUTER. IBM-PC.  
OBJECT-COMPUTER. IBM-PC.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT DATA-FILE ASSIGN TO DISK.
```

Description



➤ DATA DIVISION.

- FILE SECTION.
 - It describes files used by the program.
- WORKING-STORAGE SECTION.
 - It defines and describes all temporary variables.
- LINKAGE SECTION.
 - It is used by the subprograms.

1.7: Procedure Division

Description



- Procedure Division is divided into sections / paragraphs.
- It consists of Cobol paragraphs and Statements.

Description



➤ **Given below are common COBOL coding rules:**

- Division and Section names:
 - begin in Area A
 - end with a period, and
 - appear on a line alone – that is, with no other entries
- Paragraph names:
 - must begin in Area A,
 - must end with a period, and
 - may appear on a line alone or with other entries.

Description (Contd...)



➤ Sentences

- Must begin in Area B
- Must end with a period, and
- May appear on a line alone or with other entries

Demo



➤ **My first program**

- CH01PG01

Summary



➤ In this lesson, you have learnt:

- COBOL is Business-Oriented, Standard, English like and Self documenting Language
- A COBOL program consists of four divisions, that is Identification, Environment, Data, and Procedure.



Review Question

- **Question 1: Which of the following are division name of a COBOL program?**
 - Option 1: PROCEDURE
 - Option 2: FILE SECTION
 - Option 3: DATA
 - Option 4: PROGRAM

- **Question 2: INPUT-OUTPUT SECTION is part of ____ division.**

- **Question 3: A COBOL paragraph consists of one or more statement**
 - True/False

COBOL

Lesson 2: Working with Data



Lesson Objectives



➤ **In this lesson, you will learn:**

- The concept of level numbers
- The concept of Elementary and Group items and how to define them
- The different types of data
- The use of special characters
- The concept of Edited fields and how to use them

Description



- **A data name is a data-storage item or variable that references a particular memory location, or address, within the computer.**

Description



➤ Level Numbers:

- Level Numbers are used to represent the hierarchical data structures in COBOL programs.
 - Permissible Level Numbers are given below:
 - 01 - 49
 - 66 77 88
- 01 Level Number indicates highest level in data hierarchy and refers to entire data set contained in it.
- There should be only one name at 01 level for each record.
- All subordinate data names are coded on 02 to 49 level numbers.

Description



LEVEL NUMBER	Description	Coding Rules
01	Record Description for files and independent data items	Must begin in Area A
02 – 49	Fields within records and sub-items	Can begin in Area A or Area B
66	RENAMES Clause	Must begin in Area A
77	Independent elementary items	Must begin in Area A
88	Condition Names	Can begin in Area A or Area B

Description



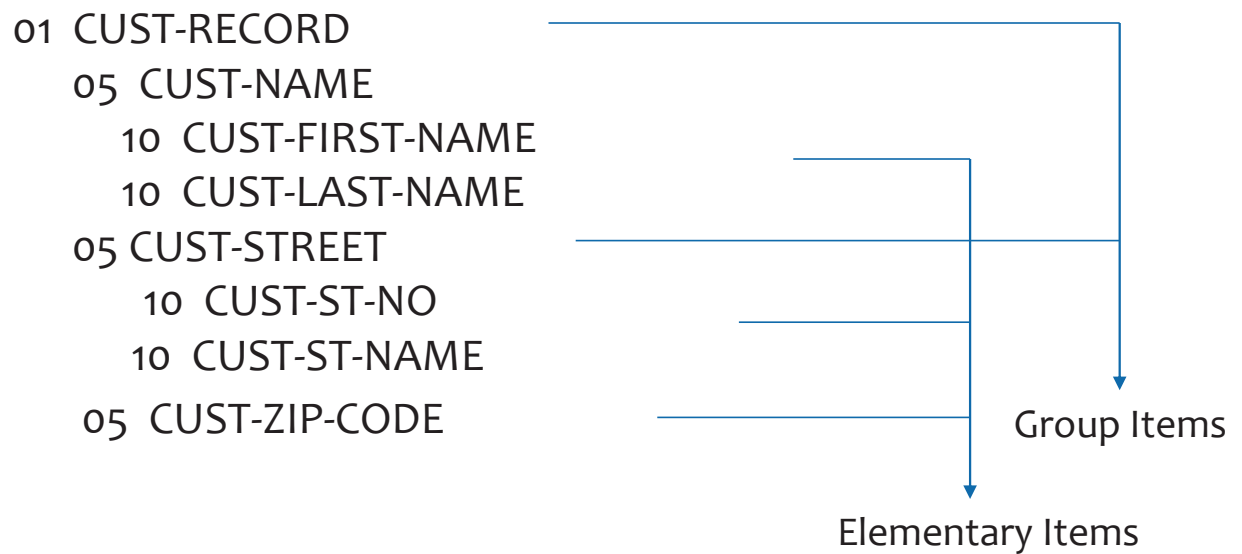
- **Group Item:**
 - It is a data field that is further sub-divided

- **Elementary Item:**
 - It is a data field that is not further divided.

Example of Fields



➤ Example



Description



➤ **Picture Clauses:**

- They specify the type of data contained within an elementary item.
- They indicate the size of the field.

➤ **Syntax:**

PICTURE type(size)

Description



➤ **Types:**

- Alphabetic – A
- Alphanumeric – X
- Numeric – 9

- **There cannot be a space between the type symbol and the number enclosed in parentheses.**

Example



➤ **Example:** Let us see an example using PIC clause:

```
01 CUST-ADDRESS.  
  05 CUST-NAME.  
    10 CUST-FIRST-NAME    PIC X(10).  
    10 CUST-LAST-NAME     PIC X(15).  
  05 CUST-STREET.  
    10 CUST-ST-NO         PIC 9(03).  
    10 CUST-ST-NAME       PIC X(20).  
  05 CUST-ZIP-CODE        PIC 9(06).
```

Description



- **The working-storage section defines and describes all data storage fields required for processing (including constants).**
- **These fields are not part of any input / output file.**
- **It can be used for storing:**
 - Intermediate results
 - Counters, Flags
 - Input / Output Records
 - Tables, and so on

Description



➤ **Data can be of two types – variable and constant.**

- Variable Data:
 - Here value may change during program execution.
- Constant Data:
 - It represents constants.
 - It is also called literals.
 - It is coded directly in the program.

Example: Constant Data



➤ Numeric Literal:

— Example:

- +34 -8.6 .008

➤ Non-Numeric Literal:

— Example:

- 'CODE' '\$ 123'

➤ Figurative Constant:

— Example:

- ZEROS SPACE HIGH-VALUE LOW-VALUE QUOTE ALL

Description



➤ **VALUE Clause:**

- It is used to assign initial values to WORKING-STORAGE fields.
- It is not used in FILE SECTION.
- For example:

```
02 PAGE-TITLE  PIC A(14) VALUE "SAMPLE PROGRAM".  
02 TAX-RATE    PIC V99  VALUE IS 0.03.
```

Rules



➤ **Let us see some rules for Continuation of Literals:**

- Begin the literal in the VALUE clause with a quotation mark.
- Continue the literal until position 72, that is the end of the line, is reached. Do not end with a quotation mark on this line.
- Place a hyphen on the next line in the position beginning in Area B of the second line. Begin with a quotation mark.
- End the literal with a quotation mark.

2.4: Continuation of Literals

Demo



➤ Continuation of Literals:

- CH02PG01.CBL

Description



➤ **Implied Decimal Point:**

- It is indicated by symbol V.
- It indicates position of an assumed decimal point.
- It does not occupy any storage space.
- Information about assumed decimal point is stored else where.

2.5: Picture Clause V program

Demo



➤ Using PICTURE clause V program:

- CH02PG02.CBL

Example



➤ Let us see an example using Implied Decimal Point:

PIC Description	Value	Stored as
99V9	38.50	385
9(4)V99	452.39	045239
9(4)V99	102.4	010240

Description



➤ Signed Numbers:

- They are used to designate a signed Numeric field.
- All fields are considered positive, unless S is used to designate negative values.
- It must be the left most character.
- It does not take any storage space. Hence it is shown as “-” on top of rightmost digit, that is “overpunct”.

2.5: Signed Numbers: Picture Clause S Program

Demo



➤ **Using picture clause S program:**

- CH02PG03.CBL

Example



➤ Let us see an example using Signed Numbers:

PIC Description	Value	Stored as
S9999V99	156.29	01562I
S9999(4)V99	-1251.33	12513L

Introduction



➤ Data Editing:

- It is mostly done for report fields.
 - For example: Leading zero suppression, Inserting commas and decimal points, Sign indication, Currency sign
- Edit characters are included in picture description of data fields.
- Data is moved from simple to edited field
- Some Edit Characters are shown below;
 - Z * \$ - + CR DB . , B o /

Description



➤ **Decimal Point:**

- It is used to insert decimal point.
- It cannot appear more than once.
- It cannot appear with V.
- It cannot be right most character.
- If the sending field has a V, alignment of numerical positions take place.

Description



- If zero suppression is specified, then period stops zero suppression on its right.
- If all the digits before and after decimal point are zero, then the resulting field will be blank.

2.5: Decimal Point

Demo



- **A demo with a program using . as edited character:**

- CH02PG04.CBL

2.5: Z (Suppressing leading zeros)

Description



➤ **Z PICTURE Character:**

- It is similar to 9.
- By using Z, the leading zeros in the digit positions indicated by Z will be suppressed.
- It can appear before as well as after decimal point.

2.5: Z (Suppressing leading zeros)

Demo



- **Let us see a program using Z as edited character:**

- CH02PG05.CBL

2.5: Z (Suppressing leading zeros)

Example



➤ Let us see an example using Z PICTURE Character:

PIC Description	Value	Stored as
Z99	25	b25
ZZZV99	0.10	bbb10
ZZZVZZ	0.052	bbb05
ZZZVZZ	0.00	bbbbb

2.5: * (Asterisk or Check Protection)

Description



➤ * Asterisk Character:

- Edit character * (asterisk or check protection) is identical to Z.
- It replaces leading zeros by * instead of space.
 - For example :

PIC Description	Value	Stored as
**999	04678	*4678
***99	00052	***52

2.5: \$ (Dollar Sign)

Description



➤ \$ Dollar Sign:

- When a \$ character is inserted, it appears at the left most position in the picture.
- Multiple occurrence of \$ results in leading zero suppression.
- Only one currency symbol is inserted to the left of the first non-zero digit of the data.
- Exception is that field may be left blank if value is zero before and after decimal point.

2.5: \$ (Dollar Sign)

Example



- Let us see an example using the \$ Dollar sign:

PIC Description	Value	Stored as
\$999V99	125.13	\$12513
\$9(5)V99	100.00	\$0010000
\$\$99V99	12.49	b\$1249
\$\$\$9V99	150.10	b\$15010
\$\$\$\$V99	0.15	bbb\$15
\$\$\$\$V\$\$	0.0	bbbbb

2.5: , (Comma)

Description



➤ , Comma Character:

- It is used to insert “,” wherever it appears.
- There can be more than one comma.
- It is replaced by space, if zero suppression takes place to the right of comma.
- It cannot appear as left most or right most character.

2.5: , (Comma)

Example



- Let us see an example using the “,” (Comma) Character:

PIC Description	Value	Printed as
\$9,999.99	2350.22	\$2,350.22
\$9,999.99	150.31	\$0,150.31
\$\$,999.99	150.31	bb\$150.31
\$\$,\$\$\$\$.99	24.40	bbb\$24.40
\$\$,\$\$\$\$.999	0.019	bbbbb\$.019

2.5: , (Comma)

Example



PIC Description	Value	Printed as
\$\$,\$\$\$.\$\$\$	0.009	bbbbbb\$.009
\$\$,\$\$\$.\$\$\$	0.0	bbbbbbbbbbb
\$\$,\$\$\$9.999	2,210.2	\$2,210.200
\$\$,999.9	2,210.2	\$2,210.2
\$\$,999.9	2,210.256	\$2,210.2
\$9,999.9999	23	\$0,023.0000

Description



- **- (Minus) and + (Plus) Characters / Sign Characters:**
 - Minus sign is inserted for a negative value.
 - Plus sign inserted for positive value, if sign edit character is +.
 - Blank inserted for positive value, if sign edit character is -.
 - Multiple occurrences of +/- can be used for leading zero suppression
 - The floating insertion (zero suppression) characters \$ + - Z * are mutually exclusive.

2.5: Sign Character(+ and -)

Example: + (Plus sign Character)



➤ Let us see an example using + (Plus sign character):

PIC Description	Value	Printed as
+999.9	35.2	+035.2
999.9+	35.2	035.2+
999.9+	-35.2	035.2-
++9.9	-001.3	b-1.3
+++9.99	.05	bb+0.05
+++9.99	-.05	bb-0.05
++++.++	.01	bbb+.01

2.5: Sign Character(+ and -)

Example: - (Minus sign Character)



➤ Let us see an example using - (Minus sign character):

PIC Description	Value	Printed as
----.--	0.0	bbbbbbb
--99.99	-10.25	b-10.25
--999.99	100.25	b100.25
999.9-	-10.2	010.2-

2.5: DB/CR

Description



➤ **DB / CR:**

- If the value is negative, then a DB or CR is inserted depending on which one is used in the picture string.
- The editing characters + - DB CR S are mutually exclusive.

2.5: DB/CR

Using DB/CR



➤ Using + - DB CR

PIC CHARACTER	Storage when Data is Positive	Storage when Data is negative
+	+	-
-	Blank	-
DB	Blank	DB
CR	Blank	CR

2.5: DB/CR

Example



➤ Let us see an example using DB / CR:

PIC Description	Value	Printed as
\$999.99DB	135.26	\$135.26bb
\$999.99DB	-135.26	\$135.26DB
,\$,\$99.99CR	-10.50	bbb\$10.50CR

2.5: B (Blank)

Description



➤ **B (Blank):**

- It is a blank insertion editing character.
- It results in blanks being entered in the designated positions.
- For example:

```
05 NAME PIC ABABA(10) VALUE 'RBSMITH'.  
NAME = RbBbSMITHbbbbbb
```

2.5: o (Zero)

Description



➤ o (Zero):

- The zero insertion editing character, causes zeros to be inserted into positions in which it appears.
- For example :

```
o5 NAME PIC ABABA(10) VALUE 'RBSMITH'.
```

```
NAME = RbBbSMITHbbbbbb
```

2.5: / (Stroke)

Description



➤ / (Stroke):

- It causes a / to be inserted in the positions specified.
- For example:

```
05 PRINT-DATE PIC 99/99/99 VALUE 040798.  
    PRINT-DATE = 04/07/98
```

2.5: / (Stroke)

Demo



➤ **Program using / as edited character:**

- CH02PG013.CBL

Description



- The memory unit of computer consists of large number of storage elements called bits (binary digits) can be 0 or 1.
- Any kind of information (both data and instructions) that stores in the memory must be represented as a sequence of bits known as bit strings.
- Memory is so organized that a group of bits can be stored or retrieved from the memory by a single operation such as group and called a WORD.
- The number of bits that constitutes a word is fixed for a particular computer and is called word length which varies from 8 bit to 64 bit.
- Each word of memory for the purpose of identification is assigned a unique integral number called address.



Description

- **Normally an instruction or single data can be stored in a word.**
- **If the word is very small, consecutive words are used to store a single instruction or data other hand if the word length is large, more than one instruction or data's can be combined in a single word.**
- **Character Data**
 - Any data can be considered to be a characters where a character is either 0 – 9 or A – Z or special characters.
- **Numeric data**
 - This data can be represented as a strip of characters or it can be represented as a binary number. Arithmetic operation can be performed more efficiently.
- **Instruction**
 - Every computer has a fixed set of instructions known as instruction set.
 - Each instruction in the set is assigned a unique operation code called op-code consisting of 5 to 8 bits.

Description



- **The USAGE clause is used to specify how a data item has to be stored in the computer's memory.**

- For example:

```
USAGE IS    { DISPLAY,  
              COMPUTATIONAL / COMP,  
              COMPUTATIONAL-1,  
              COMPUTATIONAL-2,  
              COMPUTATIONAL-3 }
```

USAGE IS DISPLAY



- Stores data in EBCDIC format (Fo thru F9)
- For signed numbers, the zone bits in the rightmost byte of the field are hex C for positive and hex D for negative and hex F for non-printing plus sign.
- Number of bytes = Size of the data
- When USAGE DISPLAY is in effect for a data item (either because you have coded it, or by default), each position (byte) of storage contains one decimal digit. This means the items are stored in displayable form.

USAGE IS DISPLAY



- External decimal (also known as zoned decimal) items are primarily intended for receiving and sending numbers between your program and files, terminals, or printers. You can also use external decimal items as operands and receivers in arithmetic processing. However, if your program performs a lot of intensive arithmetic and efficiency is a high priority, COBOL's computational numeric types might be a better choice for the data items used in the arithmetic.
- Example:
 - 77 DATA-1 PIC 99 USAGE IS DISPLAY.

Example



- **Example 1: Let us see an example where there is no explicit USAGE clause:**

```
02 AMOUNT-1 PIC 99.  
02 AMOUNT-2 PIC 99 USAGE DISPLAY.  
02 AMOUNT-3 PIC 99 USAGE COMPUTATIONAL.  
02 AMOUNT-4 PIC 99 USAGE COMP.
```

- When there is no explicit USAGE clause, the default - USAGE IS DISPLAY - is applied.

Example



➤ Example 2:

05 W02-BALANCE PIC S9999 USAGE IS COMP.

- This item requires only 2 bytes of storage as distinct from 4 bytes had usage DISPLAY been specified.
- It utilizes COMP-3 or Packed decimal format (two digits per byte, except for the right-most byte which holds one digit and the sign of the field.)

Description



➤ **REDEFINES Clause:**

- It is used to allow the same storage location to be referenced by different data-names.
- The REDEFINES clause cannot be used for the following:
 - at the 01 level in the FILE SECTION
 - when the levels of data-name-1 and data-name-2 are different
 - when the level number is 66 or 88

Example



➤ Let us see an example using REDEFINES Clause:

```
01 SAMPLE.
```

```
    02 RECEIVABLE.
```

```
        03 CUSTOMER-NUMBER          PIC 9(8).
```

```
        03 CUSTOMER-NAME            PIC X(11).
```

```
        03 AMOUNT                   PIC 9(4)V99.
```

```
    02 PAYABLE REDEFINES RECEIVABLE.
```

```
        03 VENDOR-NUMBER            PIC 9(6).
```

```
        03 VENDOR-NAME              PIC X(12).
```

```
        03 VENDOR-OWED-AMT          PIC 9(5)V99.
```

Description



➤ **RENAMES clause forms a regrouping of data-items.**

- For example:

```
01      TAX-RECORD.  
02      SOC-SEC-NUMBER PIC X(9).  
02      NAME.  
        03      FIRST-NAME      PIC X(10).  
        03      LAST-NAME       PIC X(15).
```


Example



➤ Example

02 TOTAL-YTD.

03 GROSS-PAY PIC 9(8)V99.

03 NET-PAY PIC 9(8)V99.

03 TAX PIC 9(5)V99.

66 L-GROSS RENAMES LAST-NAME THRU NET-PAY.

Description



- **Always used with receiving field**
- **Used with only Alphanumeric and Alphabetic data movement**
- **Declaration must be in working-storage section.**
- **Example**
 - 77 P PIC A(5) VALUE ' HELLO'.
 - 77 Q PIC A(10) JUSTIFIED RIGHT.
- **In Procedure division**
 - MOVE P TO Q
 - DISPLAY Q. Displays HELLO from right ie. bbbbbbHELLO

Description



- Applicable when the PICTURE string contain 'S'.
- Default is TRAILING SIGN.
- 'S' does not take any space & it stored along with last digit.

<p>+1 = A , +2 = B , +3 = C , +4 = D , +5 = E , +6 = F , +7 = G , +8 = H , +9 = I -0 = } , -1 = J , -2 = K , -3 = L , -4 = M , -5 = N , -6 = O , -7 = P , -8 = Q , -9 = R</p>

Description



- **A data name that is not unique must be qualified in COBOL verbs.**
 - For example :

```
01 IN-REC
```

```
    05 NAME    PIC X(10).
```

```
    05 AGE     PIC 99.
```

```
01 OUT-REC.
```

```
    05 NAME    PIC X(10).
```

```
    05 B-DATE  PIC 9(6).
```

```
MOVE NAME OF IN-REC TO NAME OF OUT-REC.
```



Summary

➤ In this lesson, you have learnt:

- REDEFINES clause can be used to allow the same storage location to be referenced by different data-names.
- Using RENAMES clauses you can achieve regrouping of elementary data items.
- USAGE clause is used to define the physical storage format of the content of the data item.
- Using different edited characters, we can make data more suitable for human reading.
- V denotes implied decimal point and S denotes Signed numbers.
- Level numbers along with PICTURE and VALUE clauses can be used to describe a data item.
- Justified Clause.



Review Question

- Question 1: ZEROS is called as ____.
- Question 2: A picture clause 9V9 indicates a ____ length numeric field.
- Question 3: What is wrong with the following entry?

01 TRANSACTION-REC.

05 DATE-OF-SALE PICTURE 999.

10 MONTHPICTURE 99.

10 YEAR PICTURE 99.



Review Question

- **Question 4: Is the use of level numbers TRUE / FALSE for the following snippet?**

01 IN-REC.

05 NAME-IN.

10 LAST-NAME PICTURE X(10).

10 FIRST-NAME PICTURE X(10).

05 ADDRESS-IN.

10 STREET PICTURE X(10).

10 CITY PICTURE X(10).

10 STATE PICTURE X(10).

Review Question



- Question 5: _____ clause is always used with the receiving field .
- Justified clause
 - SYNC
 - Rename clause

COBOL

Lesson 3: ACCEPT / DISPLAY Verbs





Lesson Objectives

- In this lesson, you will learn:
 - Use of ACCEPT, DISPLAY, and STOP RUN verbs

Description



- Enables the user to enter input data directly from a JCL during execution.

Format 1. ACCEPT Identifier [FROM Mnemonic - name]

**Format 2. ACCEPT Identifier FROM {
DATE
DAY
DAY - OF - WEEK
TIME}**

- DATE option returns 6 digits current date in yyyyymmdd
- DAY returns 5 digits current date in yyddd
- TIME returns 8 digits RUN TIME in hhmmssstt
- DAY-OF-WEEK returns single digit whose value can be 1-7(Monday-Sunday)
 - Example
 - ACCEPT MY-NAME FROM SYSIN.

Description



- Can reference a series of identifiers or literals.
- From time to time it may be useful to display messages and data values on the SPOOL.
- A single DISPLAY can be used to display several data items or literals or any combination of these.
- The WITH NO ADVANCING clause suppresses the carriage return/line feed.

DISPLAY { **Identifier**
Literal } [{ **Identifier** }
{ **Literal** }] ...
[**UPON Mnemonic - Name**] [**WITH NO ADVANCING**]

➤ Example :

- DISPLAY MY-NAME.
- DISPLAY "HELLO".



Description

- STOP RUN - Should be included as last logical statement in procedure division. It returns control back to OS.
- EXIT PROGRAM - Is the last executable statement of sub-program. It returns control back to main program.
- GOBACK - Can be coded in main program as well as sub program as the last statement. It just gives the control back from where it received the control.

➤ Example :

PROCEDURE DIVISION.

MAIN-MODULE.

-----> INTERMEDIATE SENTENCES-

STOP RUN. -----> LAST STATEMENT

Demo



- DISPLAY and ACCEPT verbs:
 - CH03PG01.CBL

Summary



➤ In this lesson, you have learnt:

- ACCEPT verb enables the user to enter input data directly from a keyboard.
- DISPLAY verb can reference a series of identifiers or literals.
- STOP RUN verb terminates program execution.

Review Question



- Question 1: Which of the following verb is used to terminate program execution?
- Option 1: ACCEPT
 - Option 2: DISPLAY
 - Option 3: STOP RUN

COBOL

Lesson 04: MOVE Statements





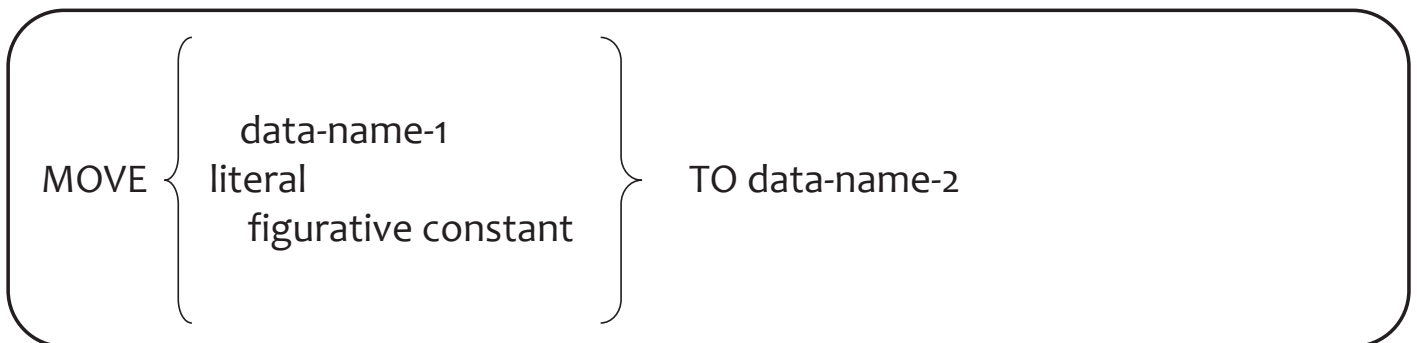
Lesson Objectives

- In this lesson, you will learn:
- Different types of MOVE statements
 - How do you avoid truncation during MOVE?
 - The concept of MOVE CORR



Introduction

- The MOVE statement moves data from sending field to receiving field.
- Contents of receiving field are changed after the operation.
- Syntax:





Example

➤ Let us see an example using the MOVE operation:

```
MOVE TOTAL TO PRINT-TOTAL.  
MOVE 'INVALID' TO MESSAGE.  
MOVE ZEROS TO TOTAL.
```



Categories

➤ There are two types of MOVE operations:

- Numeric Move
 - Same PIC clause
 - Different PIC clause
- Non-numeric Move



Mechanics of a Numeric MOVE

- A numeric move is one in which a numeric field or literal is moved to a numeric receiving field.
 - Sending and Receiving fields have the same PIC clauses:
 - The contents of receiving field will be replaced with the contents of sending field.
 - The content of sending field will be unchanged.



Rules of Numeric MOVE

➤ Rule1: Moving integer portions of numeric fields:

- All non-filled high-order (leftmost) integer positions of the receiving field are replaced with zeros.
- Example:

```
05 AMT-IN          PIC 999 VALUE 123.  
05 AMT-OUT PIC 9(4) VALUE 4567.
```

```
MOVE AMT-IN TO AMT-OUT.
```

- Result:

AMT-OUT = 0123



Rules of Numeric MOVE

- Avoiding Truncation:
 - During a numeric MOVE, if the receiving field has fewer integer positions than the sending field, then the most significant digits will be truncated.
- Example:

```
05 AMT-IN                PIC 999 VALUE 123.  
05 AMT-OUT              PIC 9(2) VALUE 45.  
MOVE AMT-IN TO AMT-OUT.
```

- Result:

AMT-OUT = 23



Rules of Numeric MOVE

➤ Rule 2: Moving decimal portions of numeric fields:

- Movement is from left to right, beginning at the implied decimal point.
- Low-order (rightmost) non-filled decimal portions of the receiving field are replaced with zeros.



Rules of Numeric MOVE

- Case 1: When receiving field has more decimal positions than the sending field:

- Example:

```
05 AMT-IN                                PIC 99V99 VALUE 12.34.  
05 AMT-OUT          PIC 99V999 VALUE 56.789.  
MOVE AMT-IN TO AMT-OUT.
```

- Result

AMT-OUT = 12.340



Rules of Numeric MOVE

- Case 2: When receiving field has fewer decimal positions than the sending field:

- Example:

```
05 AMT-IN          PIC V99 VALUE.34.  
05 AMT-OUT PIC V9  VALUE .5.  
MOVE AMT-IN TO AMT-OUT.
```

- Result
AMT-OUT = .3



Mechanics of Non-Numeric MOVE

- A non-numeric MOVE operation occurs in the following cases:
- Moving an alphanumeric or alphabetic field, defined by a PICTURE of Xs or As, to another alphanumeric or alphabetic field.
 - Moving a non-numeric literal to an alphanumeric or alphabetic field.
 - Moving a numeric field or numeric literal to an alphanumeric field or to any group item.



4.4: Non-numeric MOVE

Mechanics of Non-Numeric MOVE

- Case 1: When receiving field is larger than the sending field:
 - Example:

```
05 NAME-IN  PIC XXX VALUE "ABC".  
05 NAME-OUT          PIC X(5) VALUE "DEFGH".  
  
MOVE NAME-IN TO NAME-OUT
```

- Result

NAME-OUT = ABCbb



4.4: Non-numeric MOVE

Mechanics of Non-Numeric MOVE

- Case 2: When receiving field is smaller than the sending field:
 - Example:

```
05 NAME-IN  PIC XXX VALUE "ABC".  
05 NAME-OUT          PIC XX  VALUE "PQ".  
  
MOVE NAME-IN TO NAME-OUT.
```

- Result:

NAME-OUT = AB



Mechanics of Non-Numeric MOVE

- Case 3: When the sending field is numeric integer and the receiving field is non-numeric:
 - Example:

```
05 NAME-IN  PIC 999 VALUE 321
05 NAME-OUT          PIC X(5) VALUE "DEFGH".
MOVE NAME-IN TO NAME-OUT.
```

- Result:

NAME-OUT = 321bb



4.4: Non-numeric MOVE

Mechanics of Non-Numeric MOVE

- Case 4: When the sending field is a non-numeric literal:
 - Example:

```
05 NAME-OUT          PIC X(5) VALUE "DEFGH".
```

```
MOVE "XYZ" TO NAME-OUT.
```

- Result

```
NAME-OUT = XYZbb
```




4.4: Non-numeric MOVE

Mechanics of Non-Numeric MOVE

- Case 5: When the sending field is a figurative constant:
 - Example:

```
05 NAME-OUT          PIC X(5) VALUE "DEFGH".  
MOVE SPACES TO NAME-OUT.
```

- Result

NAME-OUT = bbbbb



Reference Modification

➤ Moving part of a field:

- Example:

```
01 W01-NAME-IN      PIC X(10) VALUE "IGATE".  
01 W01-NAME-OUT     PIC X(10).
```

```
MOVE W01-NAME-IN(2:2) TO W01-NAME-OUT.
```

- Result:

W01-NAME-OUT = GA

4.5: Group Move

Demo



- Moving part of a field program
(SUB.CBL)

4.5: Group Move

Description

- A group move is considered as a non-numeric move.





Example

➤ Example 1:

```
05 DATE-OUT.  
  10 MONTH-OUT PIC 99 .  
  10 YEAR-OUT   PIC 99.
```

```
MOVE 1 TO MONTH-OUT. MOVE 94 TO YEAR-OUT.
```

➤ Result:

```
DATE-OUT = 0194
```



Example

➤ Example 2:

MOVE 194 TO DATE-OUT.

➤ Result

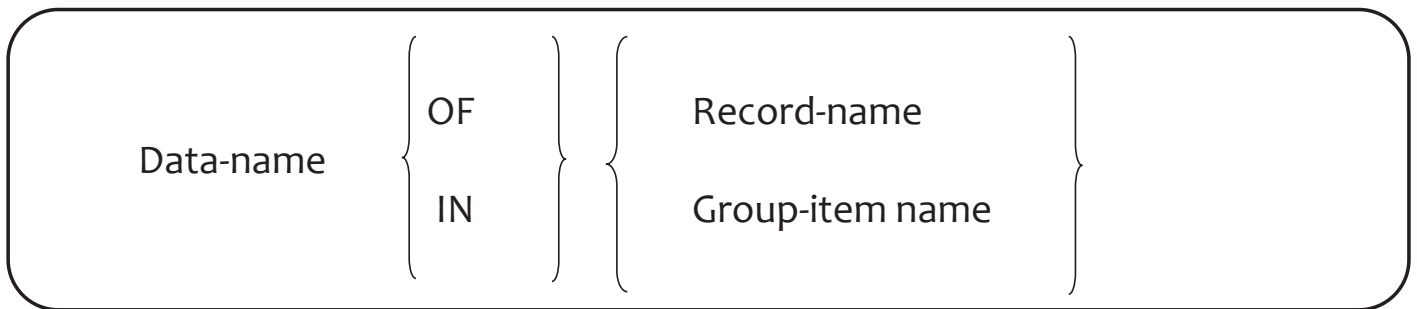
DATE-OUT = 194b



4.6: MOVE for Non-unique Data Names

Description

- Data Name that is not unique must be qualified in COBOL verbs.
- Syntax:





Example

➤ Let us see an example of MOVE for Non-unique Data Names.

01 IN-REC

05 NAME PIC X(10).

05 AGE PIC 99.

01 OUT-REC.

05 NAME PIC X(10).

05 B-DATE PIC 9(6).

MOVE NAME OF IN-REC TO NAME OF OUT-REC.

4.7: MOVE CORRESPONDING



Description

- The MOVE CORRESPONDING statement moves multiple items from one group item to another group item, using a single MOVE statement.

Example



01 INREC.

05 X

05 Y

05 Z

01 OUTREC.

05 X

05 P

05 Y

05 Q

MOVE X OF INREC TO X OF OUTREC

MOVE Y OF INREC TO Y OF OUTREC

INSTEAD

MOVE CORR INREC TO OUTREC.

Legal Move's



Receiving field

Sending field

	Alphabetic	Alphanumeric	Edited Alphanum eric	Numeric	Numeric non integer	Edited numeric
Alphabetic	Y	Y	Y	N	N	N
Alphanumeric	Y	Y	Y	N	N	N
Edited Alphanumeric	Y	Y	Y	N	N	N
Numeric	N	Y	Y	Y	Y	Y
Numeric non integer	N	N	N	Y	Y	Y
Edited numeric	N	Y	Y	Y	Y	Y



Summary

➤ In this lesson, you have learnt:

- The MOVE statements can be categorized based on the receiving field as numeric and non-numeric MOVEs.
- A group item can be moved, as well.
- By using MOVE CORR statement, all the elementary items within the sending group-item, which have the same names as corresponding elementary items in the receiving group-item, can be moved.



Review Questions

- Question 1: A Group move is considered a ____ move.
- Question 2: If the code given below is executed, what will be the value?

```
01  A      PIC      X(4) VALUE 'AKASH'.  
01  B      PIC      X(6)  
MOVE A TO B
```

- Option 1: AKASbb
- Option 2: bbAKASH
- Option 3: bAKASHb

COBOL

Lesson 5: Arithmetic Verbs





Lesson Objectives

➤ In this lesson, you will learn:

- The different arithmetic verbs such as ADD, SUBTRACT, MULTIPLY, and DIVIDE
- The method to use ROUNDED, ON SIZE ERROR along with arithmetic verbs.
- The use of COMPUTE verb



Introduction

- Primarily, the following arithmetic verbs are used in COBOL:
- ADD
 - SUBTRACT
 - MULTIPLY
 - DIVIDE
 - COMPUTE



Description

➤ Syntax of ADD is as follows:

ADD	$\left\{ \begin{array}{l} \text{Identifier-1} \\ \text{literal-1} \end{array} \right\}$	TO identifier-m [identifier-n]
ADD	$\left\{ \begin{array}{l} \text{Identifier-1} \\ \text{literal-1} \end{array} \right\}$	GIVING Identifier-m [Identifier-n]



Description

➤ Let us see an example of ADD operation:

ADD A TO B.

$$B = A + B$$

ADD A B C TO D.

$$D = A + B + C + D$$

ADD 15 A TO B.

$$B = 15 + A + B$$

ADD A B GIVING C.

$$C = A + B$$

ADD A B GIVING C D E. $C = D = E = A + B$



5.3: SUBTRACT

Description

➤ Syntax of SUBTRACT is as follows:

SUBTRACT	$\left\{ \begin{array}{l} \text{Identifier-1} \\ \text{literal-1} \end{array} \right\}$ FROM identifier-m....
SUBTRACT	$\left\{ \begin{array}{l} \text{Identifier-1} \\ \text{literal-1} \end{array} \right\}$ FROM identifier-m.. [, GIVING Identifier-m.....]



Example

➤ Let us see an example of SUBTRACT operation:

SUBTRACT A FROM B.
SUBTRACT A B FROM C
SUBTRACT A B FROM C GIVING D
SUBTRACT 15 FROM A B

SUBTRACT A B FROM 50
GIVING C

$B = B - A$
 $C = C - A - B$
 $D = C - A - B$
 $B = B - 15$
 $A = A - 15$
 $C = 50 - (A+B)$



Description

➤ Syntax for MULTIPLY is as follows:

MULTIPLY

{
Identifier-1
literal-1
}

BY

identifier-2....

MULTIPLY

{
Identifier-1
literal-1
}

BY

literal-2

{
Identifier-2
}

GIVING Identifier-3.....



Example

➤ Let us see an example of MULTIPLY operation:

MULTIPLY A BY B.
MULTIPLY A BY B GIVING C.
MULTIPLY A BY B C D.

$B = B \times A$
 $C = A \times B$
 $B = A \times B$
 $C = A \times C$
 $D = A \times D$
 $TAX-BASE = 0.05 \times TAX$

MULTIPLY TAX BY 0.05
GIVING TAX-BASE



Description

➤ Syntax of DIVIDE: 1st Method:

DIVIDE { Identifier-1
 literal-1 } INTO identifier-2....

GIVING Identifier-3 [REMAINDER Identifier-4]

➤ Syntax of DIVIDE: 2nd Method:

GIVING Identifier-3 [REMAINDER Identifier-4]



Example

➤ Let us see an example of Divide operation:

DIVIDE 5 INTO A	$A = A/5$
DIVIDE 5 INTO A GIVING B	$B = A/5$
DIVIDE 3 INTO A GIVING B C	$B=C=A/3$
DIVIDE 2.5 INTO A B GIVING C D. $C=A/2.5$	$D = B/2.5$
DIVIDE A BY 3 GIVING C.	$C = A/3$
DIVIDE A INTO B GIVING C	$C = B/A$
REMAINDER D.	D= REMAINDER



Description

- REMAINDER clause is useful to store the remainder of a division operation for additional processing.
- DIVIDE verb can be used by including a REMAINDER clause.



Description

- Result should be truncated, if computed value requires more fractional positions.
- The ROUNDED option allows rounding of the result.



Example

➤ Let us see an example of the ROUNDED option:

01 A PIC 9V99 VALUE ZEROS.

➤ Result:

ADD 1.288 TO A. A = 1.28

ADD 1.288 TO A ROUNDED A = 1.29

5.8: ROUNDED Option

Demo

- Using Rounded option
 - CH05EX04.CBL





5.9: ON SIZE ERROR Option

Description

- By using ON SIZE ERROR option, the computed value is truncated if available digit positions in result field is less.
- It generates error condition.
- It can be trapped using TRANSFERS CONTROL TO statement.



Example

➤ Let us see an example of the ON SIZE ERROR option:

MULTIPLY RATE BY HRS GIVING GROSS ON SIZE ERROR PERFORM
GROSS-TO-HIGH



Example

- Using ON SIZE ERROR program:
 - CH05PG06.CBL



Description

- The COMPUTE verb is used for evaluating arithmetic expressions.
- It uses arithmetic operators and data-names.
- Arithmetic operators used are as follows:
 - ** Exponentiation
 - * Multiplication
 - / Division
 - - Subtraction
 - + Addition



Description

- Hierarchy of operations to be used while using COMPUTE verb is as follows:
- Parentheses is used to clarify order of operations and has highest priority for expression contained in it.
 - Exponentiation is used
 - Multiplication and Division are from left to right
 - Addition and Subtraction are from left to right



Example

- Let us see an example of COMPUTE verb using hierarchy of operations:

COMPUTE GROSS ROUNDED = (HRS * RATE) + 1.5
(OVERTIME * RATE)

ON SIZE ERROR PERFORM GROSS-TOO-HIGH



Example

- Let us see an example of COMPUTE verb using hierarchy of operations:

COMPUTE GROSS ROUNDED = (HRS * RATE) + 1.5
(OVERTIME * RATE)

ON SIZE ERROR PERFORM GROSS-TOO-HIGH



Description

➤ INITIALIZE

- VALUE CLAUSE is used to initialize the data items in the working-storage section whereas INITIALIZE is used to initialize the data items in the procedure division.
- INITIALIZE sets the Alphabetic, Alphanumeric fields & alphanumeric edited items are set to SPACES, Numeric and Numeric edited items set to ZERO. FILLER , OCCURS DEPENDING ON items left untouched.
- This can be overridden by REPLACING option of INITIALIZE.

```
INITIALIZE <id-1>  
    ALPHABETIC  
    ALPHANUMERIC  
    REPLACING    NUMERIC  
                ALPHANUMERIC-EDITED  
                NUMERIC-EDITED
```

```
DATA BY <id-2> / <lit-1>
```



Example

➤ Example

01 EMP.

05 EMP-NUM PIC 9(3).

05 EMP-NAME PIC A(5).

05 EMP-ADRS PIC X(7).

05 EMP-SAL PIC 9(3).9(2).

INITIALIZE EMP-NUM.

INITIALIZE EMP-ADRS.

INITIALIZE EMP.

INITIALIZE EMP REPLACING NUMERIC DATA BY 123

REPLACING ALPHABETIC DATA BY "JACOB"

REPLACING ALPHANUMERIC DATA BY "ENGLAND"

REPLACING NUMERIC-EDITED DATA BY 235.50.



Summary

➤ In this lesson, you have learnt:

- The ADD, SUBTRACT, MULTIPLY, and DIVIDE verbs can be used to perform arithmetic operations.
- The REMAINDER clause is used in the DIVIDE operation to store the remainder.
- The ROUDED and ON SIZE ERROR is used along with arithmetic verbs.
- The COMPUTE verb can be used to perform complex and arithmetic operations.



Review Questions

➤ Question 1: Which of the following option is correct with respect to the following code?

SUBTRACT A, B FROM C GIVING D.

- Option 1: Sum of A and B will be subtracted from C and result will be stored in D.
- Option 2: Sum of A and B will be subtracted from C and result will be stored in C as well as D.
- Option 3: A will be subtracted from C and result will be stored in D, B will be ignored.



Review Questions

- Question 2: Both the phrases BY and INTO can be used in the DIVIDE verb.
 - True / False
- Question 3: What will be the value of A ? (Assume A =15)

DIVIDE	A	INTO	5
--------	---	------	---

- Option 1: 3
- Option 2: 0
- Option 3: Code incorrect

COBOL

Lesson 6: Decision Making



Lesson Objectives



➤ **In this lesson, you will Learn about:**

- COBOL Statements
- Relational Operators
- CONTINUE clause
- NEXT SENTENCE clause
- Classification of Conditions
- EVALUATE Statements

Categories



- **COBOL statements are of two categories:**
 - Conditional statements
 - Performs operations depending on existence of some condition
 - Coded with IF-THEN-ELSE structure
 - Imperative statements
 - Performs operation regardless of existing conditions
 - MOVE, ADD are examples in COBOL

IF Statement



➤ Syntax

```
IF condition-1
  [THEN]
    imperative statement-1 ...
  [ELSE
    imperative statement-2 ... ]
[END-IF.]
```

Execution



- **Execution of IF statement is as follows:**
 - When IF condition exists or is true:
 - Statement(s) after THEN are executed
 - ELSE clause is ignored
 - When IF condition does not exist or is false:
 - Statement(s) after ELSE are executed
 - Statement(s) after THEN are ignored

Example



➤ Example

If Disc-Code = 1 Then

 Multiply Amt By .15 Giving WS-Discount

Else

 Move 0 To WS-Discount.

Description



- **If Disc-Code is 1, condition is true**
 - MULTIPLY statement executed
- **If Disc-Code is not 1, condition is false**
 - MOVE statement executed
- **After execution of selected statement, program continues with statement after END-IF**
- **Else may be omitted if the operation is required only when a condition exists:**

Description



➤ **Example:**

```
If Acct-Balance < 0 Then
```

```
    Display 'Account overdrawn'
```

```
End-If.
```

- **DISPLAY** executed only if Acct-Balance is less than zero, otherwise it is ignored.

Description



➤ Symbols for simple relational conditions are:

Symbol	Description
<	is less than
>	is greater than
=	is equal to
<=	less than or equal to
>=	greater than or equal to

Examples



- Following example will explain usage of relational operators:
- Assume L, M and N are numeric with following values:

L = 12, M = 7, N = 3, then:

Condition	Result
$L \geq M$	True
$M < 7$	False
$M > N + 6$	False
$M + N \leq 10$	True

How Comparisons are Performed?



- **Nonnumeric fields are compared alphabetically**
 - ABLE < BAKE < BARK
- **Blanks on right do not affect comparison**
- **All of these are considered equal:**
 - ABC ABCbb ABCbbbbbb

How Comparisons are Performed?



- **Comparisons are performed as per following rules:**
 - Fields are compared to other fields or literals of same data type.
 - Numeric fields are compared algebraically:
 - 005 < 026 < 539
 - All of these are considered equal:
 - 012 12.00 12 +12

Description



- Continue clause indicates that no operation is to be performed when a condition exists.
- Example:

If Amt1 = Amt2

Then

Continue

Else

Add 1 to Total

End-If

{ No operation performed if Amt1 = Amt2,
program continues with statement after End-If

Description



- **NEXT SENTENCE** means that on execution control will go to the next logical statement.
- **Example:**

If Amt1 = Amt2

NEXT SENTENCE

Else

Add 1 to Total

{ No operation performed if Amt1 = Amt2,
program continues with statement after End-If

Classification



➤ **Conditions are classified into:**

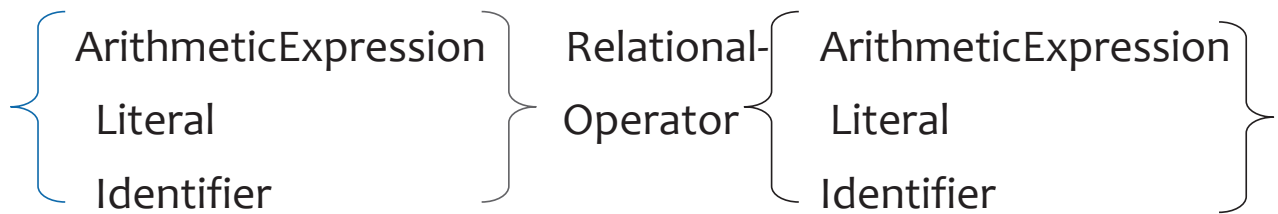
- Relational Condition
- Sign Condition
- Class Condition
- Compound Condition
- Condition-name Condition

Relational Condition



➤ **Operand-1 Relational-Operator Operand-2**

➤ **Syntax:**



Example



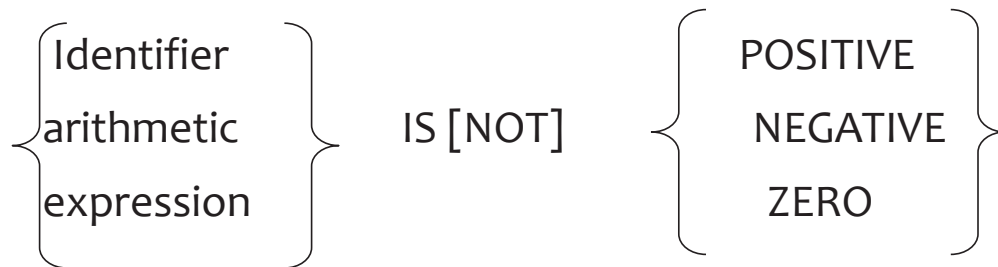
➤ Example

```
IF A GREATER THAN B
    MOVE A TO LARGER
ELSE
    MOVE B TO LARGER.
```

Sign Test



- To test if field is **POSITIVE**, **NEGATIVE** or **ZERO**



Example: Sign Test



➤ Condition

- If Amt Is Positive
- If Amt Is Negative
- If Amt Is Zero

➤ Result

- True if Amt is greater than 0
- True if Amt is less than 0
- True if Amt equals 0

Example: Sign Test



➤ Example

IF BALANCE IS ZERO

MOVE “NO BALANCE” TO MESSAGE-FIELD

Class Test



- To test whether type of data field is NUMERIC or ALPHABETIC

- Syntax:

{ Identifier } IS [NOT] NUMERIC
ALPHABETIC

Example



➤ Condition

- If Amt Is Numeric
- If Code Is Alphabetic

➤ Result

- True if Amt = 153
- False if Amt = 15B
- True if Code = PQR
- False if Code = P23

ALPHABETIC Class Test



➤ Reserved Word

- ALPHABETIC
- ALPHABETIC-UPPER
- ALPHABETIC-LOWER

➤ Result

- A-Z, a-z, and blank
- A-Z and blank
- a-z and blank

➤ Example

IF AMT-IN IS NUMERIC
PERFORM 300-CALC-RTN.

6.5: Class Condition Program

Demo



➤ Using Class Condition program

- CHo6PGo3

Compound Conditions



- A compound condition is formed by combining two or more simple conditions and logical operators.
- It enables testing of several conditions within one statement.
- AND, OR or valid combination of AND, OR is used to code compound conditions
 - Syntax:

Condition-1 {AND, OR} Condition-2

Description



- IF statement containing OR conditions indicate that if any of the several conditions exists, then the imperative statements will be executed.
- IF statement containing AND conditions indicate that if all of the several conditions are satisfied, then the imperative statement will be executed.

Rules for Compound Conditions



- **Following is the hierarchy of usage for compound conditionals:**
 - NOT
 - AND
 - OR

Description



- NOT is evaluated first.
- Conditions surrounding the word AND are evaluated first.
- Conditions surrounding the word OR are evaluated last.
- When there are several AND or OR connectors, the AND conditions are evaluated first, as they appear in the statement, from left to right. Then the OR conditions are evaluated, also from left to right.

Condition-names



- Provide meaningful names, defined for specific values, that an identifier can assume
- Associate names with employee pay code values
- Example:

<u>Pay-Code</u>	<u>Condition-name</u>
H	Hourly
S	Salaried

Example



05	Pay-Code	Pic X.
88	Hourly	Value 'H'.
88	Salaried	Value 'S'.

- Define field in DATA DIVISION
- Use level 88 to define condition-name and associated value

Description



- Use any place a condition can be used in PROCEDURE DIVISION

- Example

```
If Hourly
    Perform Calc-Hourly-Pay
End-If.
```

- If Pay-Code field has a value of 'H', condition Hourly is true
- Hourly same as condition Pay-Code='H'

Description



- **Condition-name must be unique**
- **Literal in VALUE clause must be same data type as field preceding it**
- **May be coded with elementary items with level numbers 01-49**

Description



- **88-level may specify multiple values:**

05	Opt-Num	PIC	9.
88	Valid-Options		Value 1 Thru 5

- **Valid-Options true if Opt-Num = 1, 2, 3, 4 or 5**

Example



```
05  MARITAL-STATUS          PIC X.  
    88  SINGLE                VALUE "S".  
    88  MARRIED               VALUE "M".  
  
IF MARITAL-STATUS = "S"  
    PERFORM 1000-SINGLE-ROUTINE.  
IF SINGLE  
    PERFORM 1000-SINGLE-ROUTINE.
```

Demo



- **Using condition-names to calculate insurance amount**
 - CH06PG04.CBL
- **Using nested Ifs program:**
 - NESTEDIF.CBL

Description



- **Evaluate statements are used to:**
 - Implement Case structure
 - Test for series of conditions
- **Evaluate statements may be used in place of IF statements.**
- **Codes are clearer and more efficient with EVALUATE, when multiple conditions need to be checked.**

Example



```
EVALUATE      { identifier-1  
                expression-1  
            }  
    WHEN condition-1  
        imperative-statement-1 ...  
    [WHEN OTHER  
        imperative-statement-2 ...]  
[END-EVALUATE]
```

Example



- Add, subtract or multiply a number by 10 depending on value in Op-Code
- Example:

Evaluate Op-Code

When 'A' Add 10 To Num

When 'S' Subtract 10 From Num

When 'M' Multiply 10 By Num

When Other Display 'Code invalid'.

6.6: EVALUATE Statement

Example



- **When Op-Code is 'A' the ADD statement will be executedTo**
 - Execution will continue with statement after END-EVALUATE
- **If Op-Code is not A, S or M, statement following When Other is executed.**



Summary

- **Simple relational conditions use the operators =, <, >, <=, >=**
- **Comparisons are made:**
 - Algebraically for numeric fields
 - Using collating sequence for alphanumeric fields
- **Compound conditions join simple conditions with AND or OR**
 - ANDs evaluated before ORs in order from left to right
 - Parenthesis are used to override hierarchy rules

Summary



- **Other tests include:**
 - Sign tests - POSITIVE, NEGATIVE, ZERO
 - Class tests - NUMERIC, ALPHABETIC
 - Negated conditionals - may precede any test with NOT
- **Condition-name may be defined at 88 level.**
 - Associates name with value a field may assume
 - Use name as condition in PROCEDURE DIVISION
- **EVALUATE statement is often used as alternative to IF or series of nested IFs**



Review Question

- **Question 1: Briefly explain 88 level number and why it is used.**
- **Question 2: Indicate the difference between the following two routines:**

```
IF A IS EQUAL TO B
  ADD  C  TO  D
  MOVE E  TO  TOTAL.
```

```
IF A IS EQUAL TO B
  ADD  C  TO  D
  MOVE E  TO  TOTAL.
```

COBOL

Lesson 7: Iterations





Lesson Objectives

➤ In this lesson, you will learn about:

- Looping statements
 - Simple Perform statement
 - PERFORM... THRU statement
 - PERFORM... UNTIL statement
 - PERFORM... TIMES statement
 - INLINE PERFORM
 - PERFORM TEST
- EXIT statements
- GO TO statements

Simple PERFORM Statement



➤ The simple PERFORM statement:

- Executes all instructions in the named paragraph.
- Transfers control to the next instruction in sequence, after the paragraph
- Is used to execute a paragraph from different points in a program
- Is used to modularize program:
 - Write each set of related instructions as separate module or paragraph
 - Use PERFORM paragraph-name to execute each module as needed

Example



PROCEDURE DIVISION.

100-MAIN-MODULE.

:

PERFORM 400-HEADING-RTN.

:

200-CALC-RTN.

:

400-HEADING-RTN.

:

PERFORM...THRU Statement



- **Format of the PERFORM...THRU statement is:**

PERFORM PARA-1

THROUGH
THRU

PARA-2

- **The PERFORM...THRU statement:**

- Executes group of paragraphs
- Uses expanded format to execute all statements, including other paragraphs, from paragraph-name-1 through paragraph-name-2

PERFORM...THRU Statement: Example



100-MAIN.

 PERFORM 300-PARA THRU 500-PARA.

 :

200-PARA.

300-PARA.

400-PARA.

 :

500-PARA.

EXIT.

600-PARA.

 :

PERFORM... UNTIL Statement



- Condition is tested before the paragraph is executed.

- Example:

```
MOVE 1 TO COUNTER1.  
  PERFORM 200-DISP-RTN UNTIL COUNTER1 = 4.  
  :  
  STOP RUN.  
200-DISP-RTN.  
  DISPLAY "HELLO".  
  ADD 1 TO COUNTER1.
```

- Result: * This program displays the Hello 3 times.

PERFORM...TIMES Statement



- Executes a sequence of steps a fixed number of times.

- Example:

```
MOVE 3 TO COUNTER1.  
    PERFORM 200-DISP-RTN COUNTER1 TIMES.  
    :  
    STOP RUN.  
200-DISP-RTN.  
    DISPLAY "HELLO".
```

- Result: *This program displays the Hello 3 times.

Demo



➤ Iterations program

- CH07PG01.CBL

INLINE PERFORM Statement



- When the body of the perform will not be used in other paragraphs.
- If the body of the perform is a generic type of code (used from various other places in the program), it would be better to put the code in a separate paragraph and use PERFORM para name rather than in-line perform.
- Example:

```
PERFORM ADD A TO B  
    MULTIPLY B BY C  
    DISPLAY 'RESULT IS'  
END-PERFORM.
```

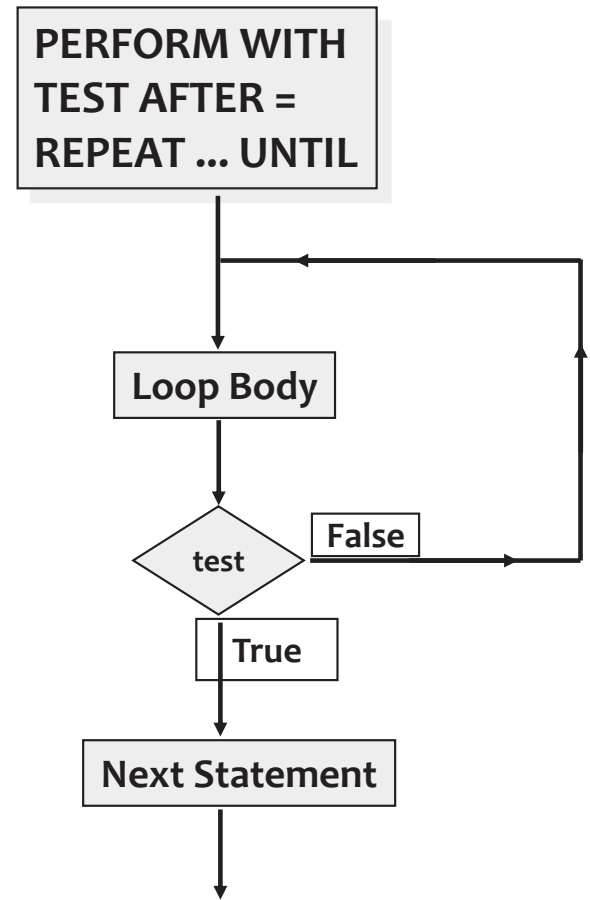
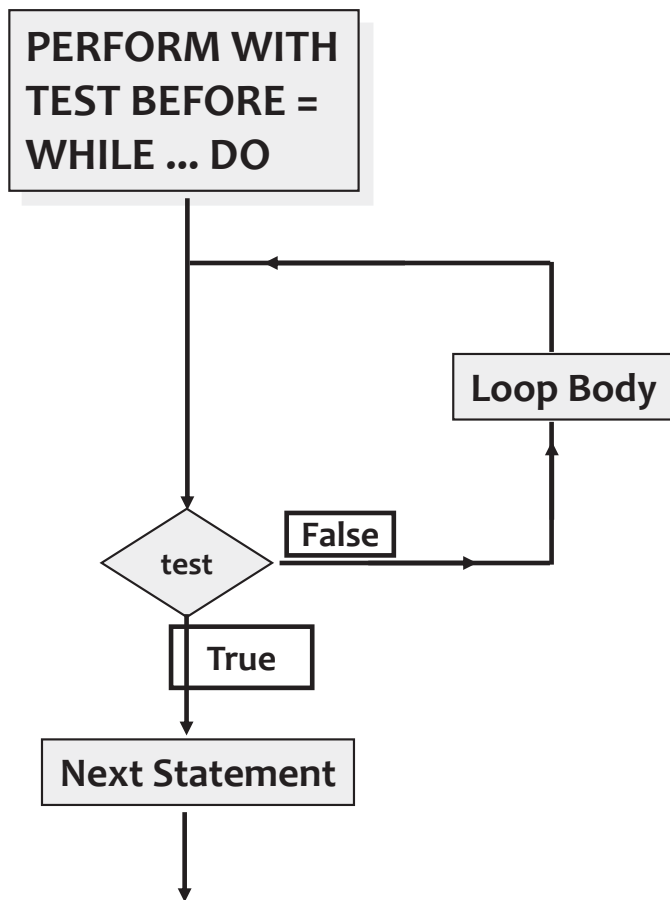
PERFORM with TEST



$$\begin{array}{c} \text{PERFORM} \left[1\text{stProc} \left[\left\{ \begin{array}{c} \text{THRU} \\ \text{THROUGH} \end{array} \right\} \text{EndProc} \right] \right] \left[\text{WITH TEST} \left\{ \begin{array}{c} \text{BEFORE} \\ \text{AFTER} \end{array} \right\} \right] \\ \text{UNTIL Condition} \\ \left[\text{StatementBlock} \text{ END - PERFORM} \right] \end{array}$$

- This format is used where the WHILE or REPEAT constructs are used in other languages.
- If the WITH TEST BEFORE phrase is used the PERFORM behaves like a DO... WHILE loop and the condition is tested before the loop body is entered.
- If the WITH TEST AFTER phrase is used the PERFORM behaves like a DO... UNTIL loop and the condition is tested after the loop body is entered.
- The WITH TEST BEFORE phrase is the default and so is rarely explicitly stated.

PERFORM with TEST



PERFORM with TEST



➤ Syntax

```
PERFORM <para-name> WITH TEST  BEFORE  
                                     AFTER  
VARYING {identifier-1} FROM  identifier-2  
                                     literal  
BY  identifier-3  UNTIL {condition}  
                                     literal
```


Description



➤ **The EXIT word:**

- Is a COBOL reserved word that performs no operation
- Is present only for readability
- When executed, no actions take place

Description



- **The GO TO statement:**
 - Transfers a control to another paragraph
 - Does not return control back

Example



```
:  
    IF CNT1 = CNT2  
        GO TO PARA-EQUAL  
    ELSE  
        PERFORM PARA-CHECK UNTIL CNT1 = CNT2.  
:
```

Lab



➤ Lab 1

Summary



➤ **Formats of PERFORM Statement:**

- Simple PERFORM
- PERFORM para-1 [THRU/THROUGH para-2]
 - Causes execution of instructions in named paragraph(s)
 - After paragraph executed, control returned to statement after PERFORM
- PERFORM... UNTIL
 - PERFORM UNTIL repeats instructions until a condition is met
 - Condition may be tested before or after instructions are executed



Summary

- **PERFORM...TIMES** statement
 - Use when you know exact number of times loop statements are to be executed
- **INLINE PERFORM**
 - When the body of the perform will not be used in other paragraphs
- **PERFORM...TEST**
 - This format is used where the WHILE or REPEAT constructs are used in other languages

- **EXIT statement:**
 - Reserved word that performs no operation
- **GO TO statement:**
 - Permanently transfers control to another paragraph



Review Questions

- **Question 1: Which of the following verb is reserved word that performs no operation?**
 - GO TO
 - EXIT
 - PERFORM
- **Question 2: A Condition Name can be associated to a GROUP Name.**
 - True/False



Review Questions

- Question 3: Following is a **PROCEDURE DIVISION** statement?

```
IF A>B MOVE A TO C  
    ELSE  MOVE B TO C.  
ADD C TO D.
```

- Indicate which of the following gives a correct description of the above statement:
- The value of A will be added to D only if A is greater than B
 - The value of B will be added to D only if B is greater than A

Review Question



- **Question 4: If the WITH TEST BEFORE phrase is used the PERFORM behaves like a DO... WHILE loop and the condition is tested before the loop body is entered.**
 - True/False

COBOL

Lesson 8: File Handling (Sequential Files)





Lesson Objectives

- In this lesson, you will learn about:
- Files and their characteristics
 - File operations
 - File organization methods
 - Making entries for a file in a program
 - INPUT / OUTPUT verbs
 - COPY statement



Introduction

➤ What is a file?

- A file is a collection of data related to a set of entities and typically exists on a magnetic tape or a disk

➤ Characteristics of a file:

- Data contained in a file is logically organized into records.
- Individual data items in a record are called its fields.
- Records present in a file may be of fixed length or variable length.



How Files are Processed

- Files are processed by reading the records into the computer's memory.
- Memory allocated for storing such records is usually called a "buffer"
- To transfer records from an input file to an output file we will have to
 - read the records into the input file's buffer
 - transfer it to the output file's buffer
 - write the data to the output file from the buffer



Buffers

➤ Program

IDENTIFICATION DIVISION.

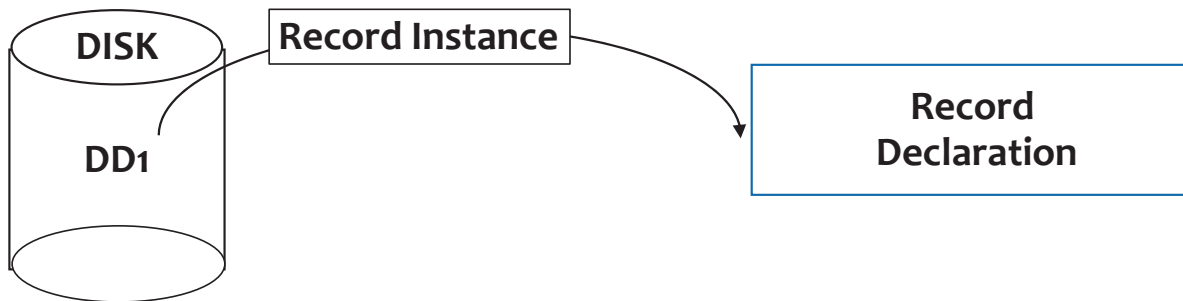
.....

ENVIRONMENT DIVISION.

.....

DATA DIVISION.

FILE SECTION.





Description

➤ File operations are:

- Create
 - Refers to producing a brand new file and writing one or more logical records into it
- Retrieve
 - Refers to reading the logical records from a file.
- Update
 - Refers to maintenance of records in a file to ensure that it is up to date. There are three types of updating operations. They are Record deletion, Record insertion and Record modification.



Categories

➤ File organization methods are:

- Sequential
- Indexed
- Relative



Description

➤ In sequential file organization:

- Records are always read in sequence
- First record is read and processed, then second record is read and processed and so on

➤ Example:

Payroll System records in order by employee number may be processed in sequence for updating or printing reports.



Description

- Consists of two files:
 - Data file - contains records in sequence
 - Index file - contains keys and pointers
- For random access, look up key field in index file to find address:
 - Then access record in data file directly



Description

➤ In relative file organization:

- Logical order and physical order of records do not necessarily correspond with one another.
- A file is thought of as a string of record areas, each of which contains a single record.



Description

- Following entries are required in a program for any file:
- File description entries
 - Specify physical aspects of the data
 - Record description entries
 - Describe logical records in the file



Description

- INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION:
 - Follows the CONFIGURATION SECTION
 - Supplies information concerning the input and output devices used in the program
- In the FILE-CONTROL paragraph:
 - A file-name is selected for each file to be used in the program
 - Each file-name selected is assigned to a device



Example

```
ENVIRONMENT DIVISION.  
:  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT EMPLOYEE-FILE  
        ASSIGN TO 'EMP.DAT'  
        ORGANIZATION IS SEQUENTIAL.
```

- The SELECT statement is coded in Area B
- ORGANIZATION clause identifies the logical structure of the file



Access Mode

➤ Access mode

- The way the records can be written into or read from.

ENVIRONMENT DIVISION.

SELECT <logical-filename> ASSIGN TO <physical-filename>

ORGANIZATION IS <type>

ACCESS MODE IS <type>

➤ Types

- Sequential (Records accessed in entry sequence)
- Random (Records accessed randomly)
- Dynamic (Both Random and Sequential accessing)

➤ Access mode also depends on Organization:

- If Sequential, Access mode can only be Sequential
- If Indexed or Relative, Access mode can be any type



Collating Sequence

- There are two popular collating sequence available in computers.
- IBM and IBM compatible machine use EBCDIC collating sequence whereas most micro and many mainframe systems use ASCII collating sequence.
- The result of arithmetic and alphabetic comparison would be same in both collating sequences whereas the same is not true for alphanumeric comparison.

EBCDIC (Ascending Order)	ASCII (Ascending Order)
Special Character	Special Character
a-z	0-9
A-Z	A-Z
0-9	a-z



Description

➤ FILE SECTION:

- Describes record type/template of every file, used in a program, by means of an FD (file description) entry.
- Each FD entry will describe a file (internal) defined in a SELECT statement in the ENVIRONMENT DIVISION.



Description

- To provide identifying information about the file on disk or tape.
- Are usually created as the first and last records of a disk or tape.
- For input files, labels may be checked to ensure that the file being accessed is the correct one.
- For output files, the first record on disk or tape file will be created as a standard 80-position header label identifying the file to the system.



Types

- There are two types of label records clause:
 - Standard
 - LABEL RECORDS ARE STANDARD: permitted for disk and tape files only
 - Omitted
 - LABEL RECORDS ARE OMITTED: used for devices such as printer

Description



- RECORD CONTAINS clause:
 - Indicates the size of each record.



Description

- Blocking is a technique that increases the speed of input/output operations and makes more effective use of storage space on disk and tape
- A group of logical records is included within one block to maximize the efficient use of a disk or tape area



Example

- For each file defined in the program, we have to define one record format.
- Example:

```
01  EMPLOYEE-REC.  
   05  EMP-NAME.  
       10  EMP-FIRST-NAME  PIC X(10).  
       10  EMP-LAST-NAME   PIC X(15).  
   05  EMP-DEPT             PIC X(4).  
   05  EMP-SALARY           PIC 9(5)V99.  
   05  EMP-DOJ              PIC 9(6).
```



Example

```
DATA DIVISION.  
FILE SECTION.  
FD  EMPLOYEE-FILE  
    LABEL RECORDS ARE STANDARD  
    RECORD CONTAINS 70 CHARACTERS  
    BLOCK CONTAINS 10 RECORDS.  
01  EMPLOYEE-REC.  
    05  EMP-NAME.  
        10  EMP-FIRST-NAME  PIC X(10).  
        10  EMP-LAST-NAME   PIC X(15).  
    05  EMP-DEPT            PIC X(4).  
    05  EMP-SALARY          PIC 9(5)V99.  
    05  EMP-DOJ             PIC 9(6).
```



Types

➤ INPUT/OUTPUT verbs are:

- OPEN
- READ
- WRITE
- REWRITE
- CLOSE



Description

- The OPEN statement:
 - Designates files as either input or output
 - Makes the files available for processing
 - Performs header label routine checks
- A file must be opened before it may be read or written or processed.
- Order in which files are opened in program are not significant.



Description

➤ Modes for opening a file are:

- INPUT
- OUTPUT

➤ Example

```
OPEN INPUT EMPLOYEE-FILE.
```

```
OPEN OUTPUT REPORT-FILE.
```



Description

- The READ statement transmits records from the input device, assigned in the ENVIRONMENT DIVISION, to the input storage area, defined in the FILE SECTION of the DATA DIVISION.
- Each time a READ statement is executed, one record is read into the primary storage.
- AT END clause in the READ statement tests to determine if there is any more inputs.
- After an input file has been opened, it may be read.

Example

➤ Example

```
READ EMPLOYEE-FILE  
  AT END  
    MOVE "YES" TO END-OF-FILE
```





Description

➤ The WRITE statement:

- Takes data in the output area defined in the DATA DIVISION and transmits it to the device specified in the ENVIRONMENT DIVISION i.e. writes the specified record.

➤ Example:

```
WRITE EMPLOYEE-REC.
```



Description

- The REWRITE statement is used:
 - To update an existing record.
 - To read a record successfully into the record buffer.
- Syntax:

```
REWRITE EMPLOYEE-REC [FROM data-name].
```



Description

➤ The CLOSE statement:

- Is coded at the end of the job, after records have been processed, to release these files and deactivate the devices.

➤ Example:

```
CLOSE EMPLOYEE-FILE.
```

8.18: Create a File

Demo

- A program to create a file
 - CH08PG01.CBL





Description

➤ The COPY statement:

- Brings into a program a series of prewritten COBOL entries that have been stored in a library.
- Saves a programmer considerable amount of coding and debugging time.
- Promotes program standardization.
- Reduces the time it takes to make modifications and reduces duplication of effort.
- Library entries are extensively annotated so that they are meaningful to all users.



Example

➤ Contents of EMP.REC:

```
01  EMPLOYEE-REC.  
    05  EMP-NAME.  
        10  EMP-FIRST-NAME  PIC X(10).  
        10  EMP-LAST-NAME   PIC X(15).  
    05  EMP-DEPT             PIC X(4).  
    05  EMP-SALARY           PIC 9(5)V99.  
    05  EMP-DOJ              PIC 9(6).
```



Example

➤ The DATA DIVISION entry using a COPY statement:

```
DATA DIVISION.  
FILE SECTION.  
FD  EMPLOYEE-FILE  
    LABEL RECORDS ARE STANDARD  
    RECORD CONTAINS 70 CHARACTERS  
    BLOCK CONTAINS 10 RECORDS.  
COPY "EMP.REC".
```



How the COPY Works?

- If SAME COPYBOOKS is used more than once in the program, then there will be “duplicate data declaration” error during compilation, as all the fields are declared twice. .In this case, one copybook can be used with REPLACING verb to replace high-level qualifier of all the variable with another qualifier.
 - Example: COPY CUSTOMER REPLACING 'CUST1-' BY 'CUST2-'
- If the REPLACING phrase is not used then the compiler simply copies the text into the client program without change.
- If the COPY does use the REPLACING phrase then the text is copied and each properly matched occurrence of Pseudo-Text-1, Identifier-1, Literal-1 and Word-1 in the library text is replaced by the corresponding Pseudo-Text-2, Identifier-2, Literal-2 or Word-2 in the REPLACING phrase.



How the REPLACING Works?

- Pseudo-Text is any COBOL text enclosed in double equal signs (e.g. `==ADD` `1==`). It allows us to replace a series of words or characters as opposed to an individual identifier, literal or word.
- The REPLACING phrase tries to match text-words in the library text with text-words before the BY in the REPLACING phase. If a match is achieved then as text is copied from the library it is replaced by the matching REPLACING phrase text.
- For purposes of matching, each occurrence of a separator comma, semicolon, space in pseudo-text-1 or in the library text is considered to be a single space.
- Each sequence of one or more space separators is considered to be a single space.
- Comment lines in either the library or REPLACING phrase text are treated as a single space.

8.19: Read a File

Demo



- A program to read a file
 - CH08PG02.CBL



Lab

➤ Lab 2.1

➤ Lab 2.2



Summary

- The three methods of file organization available on disk systems are: sequential, indexed sequential, and relative file organization.
- The INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION follows the CONFIGURATION SECTION and supplies information concerning the input and output devices used in the program.



Summary

- Each file in the program has to be opened before performing any kind of READ/WRITE/REWRITE processing.
- Each file, that has been opened in the program has to be closed.
- A COPY statement is used to bring into a program a series of prewritten COBOL entries that have been stored in a library.
- COPY with REPLACING phrase.



Review Questions

- Question 1: Indicate which of the following methods organizes records in a file in sequential order:
- Option 1: RELATIVE
 - Option 2: INDEXED
 - Option 3: SEQUENTIAL



Review Questions

- Question 2: File Control entries are defined in AREA A.
 - True / False?
- Question 3: _____ verb is used to update the existing record.

COBOL

Lesson 09: Sorting



Lesson Objectives



➤ In this lesson you will learn about:

- The SORT Statement
 - Ascending and Descending keys
 - Files used in SORT
 - Multiple SORT keys
 - Duplicate Key Values
 - Sample FILE SELECTION

Description



- **The SORT statement allows:**
 - Arranging of records in specific order
 - Sequential batch processing to be performed
- **Two techniques for sorting are:**
 - Use sort utility separate from COBOL program
 - Use COBOL's SORT verb in program

9.1: SORT Statement

Format



ASCENDING

SORT work-file ON DESCENDING KEY dataname1
USING in-file GIVING out-file

- **work-file DEFINED WITH SD ENTRY**
- **in-file CONTAINS UNSORTED DATA**
- **out-file CONTAINS SORTED DATA**

Ascending and Descending Keys



- **Sequence specified for key fields can be:**
 - ASCENDING: From lowest to highest
 - DESCENDING: From highest to lowest
- **Key fields can be numeric or nonnumeric**
- **Alphanumeric fields are sorted according to collating sequence (ASCII or EBCDIC) used by computer**

Files Used In SORT



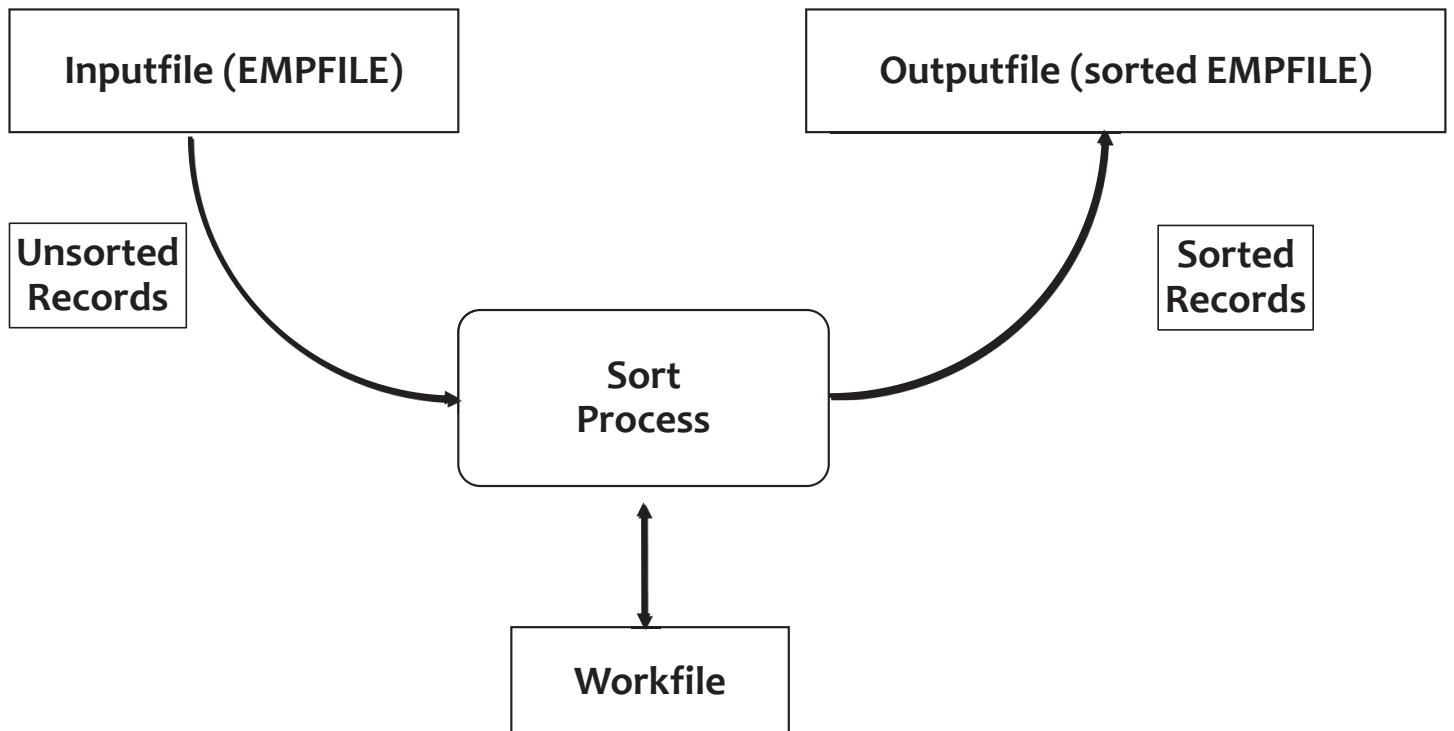
- **Following are the file types used in SORT procedure:**
- Input file: File of unsorted input records
 - Work or sort file: File used to store records temporarily during sorting process
 - Output file: File of sorted output records

File Types Description



- **All file types used in SORT procedure:**
 - Are defined using standard SELECT ... ASSIGN entries
 - Have same record format
 - Are opened and closed automatically by SORT

9.1: SORT Statement Description



Description



- **Input and output files are described with FD entries**
- **Sort work file:**
 - Described with SD entry (sort file descriptor)
 - Temporary file used only during sorting but not saved
 - Sort key fields must be described as part of sort record format

Multiple Sort Keys: Description



➤ **Multiple SORT Keys:**

- Can sequence records with more than one key field
- Can sort payroll file in ascending alphabetic sequence by name, within each level, for each office:
 - Office number - major sort field
 - Level number - intermediate sort field
 - Name - minor sort field

Multiple Sort Keys: Description



➤ For Office 1, desired sequence is:

Office-No	Level-No	Name
1	1	ADAMS, J. R
1	1	BROCK, P. T.
1	1	LEE, S.
1	2	ARTHUR, Q. C.
1	2	SHAH, J.
1	3	RAMIREZ, A. P.

Multiple Sort Keys: Example



- Sorts records into ascending name sequence within level within office.

Sort Sort-File

On Ascending Key Office-No

On Ascending Key Level-No

On Ascending Key Name

Using Payroll-File-In

Giving Sort-Payroll-File-Out

Multiple Sort Keys: Description



- Choose either **ASCENDING** or **DESCENDING** sequence for each key.
- If all key fields are to be sorted in same sequence, condense coding.

Sort Sort-File

On Ascending Key Major-Key

Intermediate-Key

Minor-Key

...

Duplicate Key Values



- Assume records to be sorted in descending order by salary.
- If both 9th and 24th records in input file have salary of 30000, which will appear first in sort file?
- You can specify that records with same value for key field be placed in sort file in the same order that they appear in original input file.

Duplicate Key Values: Example



- **DUPLICATES** clause ensures that 9th record appears before 24th in Sort-File if both have same salary value

Sort Sort-File

On DESCENDING KEY Srt-Salary

With Duplicates In Order

Using Unsorted-File-In

Giving Sorted-File-Out

Sample File Selection



DATA DIVISION.

FILE SECTION.

FD UNSORTED-FILE-IN.

01 UNSORTED-REC-IN.

05 NAME-IN PIC X(20).

05 SALARY-IN PIC 9(6).

Sample File Selection: Example



```
SD  SORT-FILE.
01  SORT-REC.
      05      SRT-NAME          PIC X(20).
      05      SRT-SALARY       PIC 9(6).
FD  SORTED-FILE-OUT.
01  SORTED-REC-OUT.
      05      NAME-OUT         PIC X(20).
      05      SALARY-OUT       PIC 9(6).
```

Input and Output Procedure



➤ INPUT PROCEDURE

- Used to perform validation/editing before sort
- Reads Records from Input File
- Releases Record for Sorting Using `RELEASE verb` (Special Form of `WRITE`)
- Code Written in Procedure Division Sections
- Must have statements to Open and close Input files
- `RELEASE record-name [FROM id-1]`

➤ OUTPUT PROCEDURE

- Used to perform editing/validation after sort but before writing output
- Reads Sorted Records using `RETURN verb` (Special Form of `READ`)
- `RETURN work-file-name [INTO id-1] AT END` imperative statement
- Code Written in Procedure Division Sections
- Must have statements to Open and close Output file

Input and Output Procedure



- **SORT <WK-FILE>**
 - ON ASCENDING KEY < WK-KEY-FLD>
 - INPUT PROCEDURE <SEC-NAME1>
 - OUTPUT PROCEDURE <SEC-NAME2>
- **SORT <WK-FILE>**
 - ON ASCENDING KEY < WK-KEY-FLD>
 - USING INFILE
 - OUTPUT PROCEDURE <SEC-NAME2>
- **SORT <WK-FILE>**
 - ON ASCENDING KEY < WK-KEY-FLD>
 - INPUT PROCEDURE <SEC-NAME1>
 - GIVING OUTFILE

Description



➤ MERGING

- Merges is to combine two or more identically sequenced files on specified key and make the record available in the merged order.
- Uses MERGE verb
- Syntax

ASCENDING

MERGE work-file ON DESCENDING KEY dataname1

USING in-file-1, in-file-2 [,in-file-3]...

GIVING out-file-1

Description



➤ INPUT FILES

- Specified in USING phrase.
- MUST BE Sequential Files Sorted on Merge Keys.
- MERGE Keys at same position in all input files
- Same size records in all files

➤ WORK-FILE

- Defined with SD entry
- Record description must have MERGE Line 3.1.1

➤ OUTPUT FILE

- Specified in GIVING phrase
- Order of records with same key value will be that of input files in USING phrase

Lab



- Lab 2.3 and 2.4 perform sorting for `itemseq.dat` and `custseq.dat` files.

Summary



- **SORT is used for sorting records in either ascending or descending order.**
- **SORT uses work or sort file described with an SD.**
- **Key fields to be sorted are data-names defined within SD or sort file.**
- **Files may be sorted using more than one key field.**

Review Question



- **Question 1: Which of the following files is opened by the SORT verb?**
- Sort file
 - Work file
 - Input file
 - All of the above



Review Question

- **Question 2: It is not necessary to define sort key in the record description of work file**
 - True/False

- **Question 3: _____ Used to perform validation/editing before sort**
 - SORT
 - SORT INPUT PROCEDURE
 - SORT OUTPUT PROCEDURE

COBOL

Lesson 10: Trapping Runtime Errors



Lesson Objectives



- To understand the File Status clause and how to use it in the program.

FILE STATUS Clause



- **The FILE STATUS clause:**
 - Is used for handling I-O exception conditions
 - Is defined in the SELECT entry
- **Field should be defined in the WORKING-STORAGE SECTION as a two-position alphanumeric field.**

Code Snippet



```
SELECT EMPLOYEE-FILE ASSIGN TO "EMP.DAT"  
      ORGANIZATION IS LINE SEQUENTIAL  
      FILE STATUS IS WS-STATUS.  
:  
WORKING-STORAGE SECTION.  
01 WS-STATUS      PIC X(2).
```


Code Snippet



```
OPEN INPUT EMPLOYEE-FILE.  
IF WS-STATUS NOT = "00"  
    DISPLAY "ERROR OPENING EMPLOYEE FILE"  
    STOP RUN.  
READ EMPLOYEE-FILE ....  
:  
:
```

Demo



- **A sample program that adds records to a file and illustrates the use of file status clause:**

- CH10PG01.CBL

Summary



- A data item of X(02) has to be declared in the WORKING-STORAGE SECTION to display File Status value.
- For each file processed in the program, a file status variable should be declared as good programming practice.

Review Question



➤ **Question 1: Which of the following file status variable declaration is correct?**Option 1

- 01 FS01 PIC X(02)
- 01 FS01 PIC 9(02)
- 01 FS01 PIC X(02)
- 05 FS01 PIC X(02)
- 01 FS01 PIC X(03)

COBOL

Lesson 11: Report Using Control Break Processing





Lesson Objectives

- How to generate a simple report.
- How to use Control Break Processing in generation of reports.

11.1: Simple Reports without Control Break

Demo

- Program to list employees
in SALES department
 - CH11PG01.CBL





Description

➤ What is control break procedure ?

- A control break procedure is used if records are in sequence by a control field and the number of records in each control field is variable.



Terms and Definitions

- Reports in business frequently require:
 - Detail printing - The printing of one or more lines for each input record.
 - Summary or group printing - The printing of totals or other summary information for groups of records.
- The Control Break Processing can be easily applied for generation of both detail and/or summary reports.



Terms and Definitions

- A report has both detail and summary printing.
- That is, when input records with the same department number are read, the records are printed and the total amount of sales for each salesperson is accumulated.



Terms and Definitions

- When a change or “break” in the department number occurs, the accumulated total of all amounts of sales is printed as a control total line.
 - We call department number a control field.
- Summary printing is performed as a result of the control break that occurs when there is a change in the department number.



Printing Headings and Footings

- A Report program can designate print lines of the following types:
 - REPORT HEADING (RH) - Prints identifying information about the report only once, at the top of the first page of the report.
 - PAGE HEADING (PH) - Prints identifying information at the top of each page.
 - This may include page numbers, column headings, and so on.



Terms and Definitions

- CONTROL HEADING (CH) - Prints a heading that typically contains new control values when a control break has occurred.
- DETAIL (DE) - Prints for each input record read.
- CONTROL FOOTING (CF) - Prints control totals for the previous group of detail records just printed, after a control break has occurred.



Terms and Definitions

- PAGE FOOTING (PF) - Prints at the end of each page.
- REPORT FOOTING (RF) - Prints only once, at the end of the report.
 - This may include, for example, an 'End of Report' message.



Description

- Produces summary report using control fields to indicate when totals are to print
- Records with same value for control field grouped together in file
- Totals printed for each group of records with same control field value



11.2: CONTROL BREAK PROCESSING

Example

- Consider file of sales records, each with three fields:
 - Salesperson's department number, salesperson's number, sales amount
- Department number is control field
- Records in sequence by control field
 - All records for salespeople in Dept 01 followed by records for those in Dept 02, ...



Example

Sample	Sales	Input Data
01	12345	098855
01	12346	353700
01	12347	003499
02	12222	987700
02	12234	008777

Amt-Of-Sales-In

Sales-person-No

Dept-In



11.3: CONTROL BREAK PROCESSING

Description

- Report includes detail printing
 - One line for each record with salesperson's amount of sales
- Report also includes summary lines or group printing
 - Total line written for each department
 - After all records for Dept 01 read and printed, total for Dept 01 printed
 - Same is done for Dept 02, and so on

11.3: CONTROL BREAK PROCESSING

Single Control Break Sample Report



Dept	Salesperson No	Amt of Sales
Deptno: 1		
01	12345	\$988.55
01	12346	\$3,537.00
01	12347	\$34.99
	Total for Dept is	\$4,560.54
Deptno: 2		
02	12222	\$9,877.00
02	12234	\$87.77
	Total for Dept is	\$9,964.77



Description

- Records must be in sequence by control field (department number)
- Sales records for Dept 01 read in, printed, Dept total accumulated
- Continues until record read in with different Dept number
- Change in Dept triggers printing of Dept total for Dept 01



Control Break Processing Steps

- Step 1: Read the initial record.
- Step 2: Move the control field to a hold area in WORKING-STORAGE.
- Step 3: As long as the control field is equal to the hold area, execute the detail routine for the input record.
 - Add the appropriate amount to a control total.
 - Print the detail record.
 - Read the next record.



Description

- If the control field is not equal to the hold area :
 - Print the control total.
 - Initialize the control total field to zero.
 - Reinitialize the hold field with the new control field value if there are more records.
 - Process the detail record as in step 3.
 - Print headings on a new page if each control total is to appear on a separate page.
- If required, after all records have been processed perform a control break to print the last control group.



Demo

- Program to demonstrate single level control break
 - CH11PG02.CBL



Demo

- Program to demonstrate double level control break
 - CH11PG03.CBL



Lab

- Lab 2.3.2
- Lab 2.4.2

Summary



- For the first record read, move the control field to a hold area in WORKING-STORAGE.
- For each additional record, as long as the control field is equal to the hold area, execute the detail routine for the input record. This means: Add the appropriate amount to a control total, print the detail record
- (if desired), and read the next record.

Summary



- **If, for a specific record read, the control field is not equal to the hold area:**
 - Print the control total
 - Initialize the control total field to zero
 - Reinitialize the hold field with the new control field value if ARE-THERE-MORE-RECORDS is not equal to 'NO'
 - Process the detail record as in Step3
 - Print headings on a new page if each control total is to appear on a separate page.
- **After all records have been processed, perform a control break to print the last control group.**



Review Questions

- Question 1: In order to execute a control break program, input data must be in sequence by the control fields.
 - True/False

- Question 2: When each individual input records results in the printing of an output line, we call this _____.

COBOL

Lesson 13: Call statement



Lesson Objectives



- Understand how to call other programs from the main program.
- Understand how to pass variables between main and sub programs.
- Be familiarized with the linkage section.

Structured Program: Introduction



- Structured programs should consist of a series of independent modules executed from the main module.

Description



➤ Calling program

- Main program that references or calls a subprogram is referred to as the calling program

➤ Called program

- Subprogram that is linked and executed within the main program is referred to as the called program

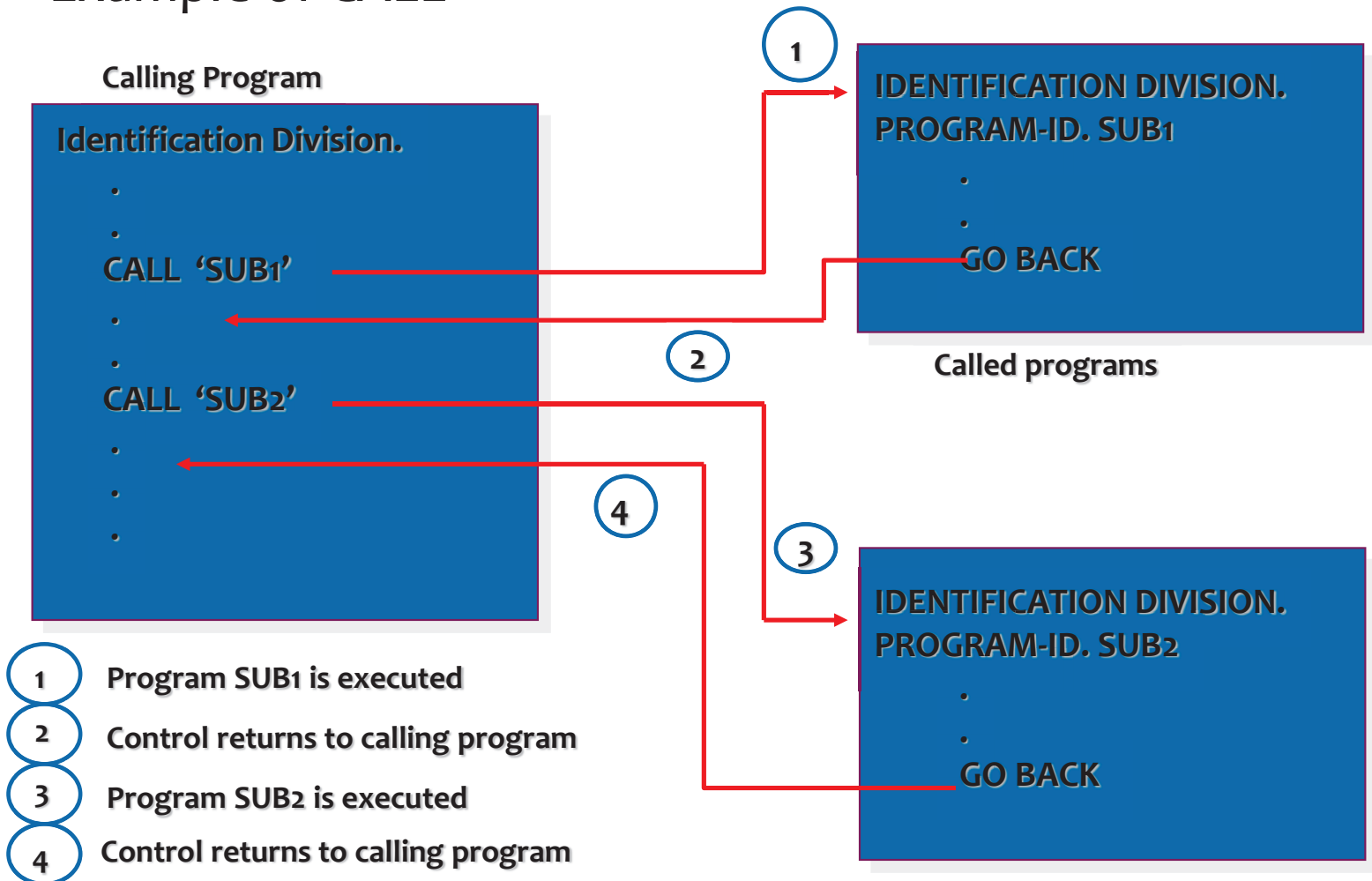
Description



➤ Called program requirements:

- PROGRAM-ID
- LINKAGE SECTION
- PROCEDURE DIVISION USING
- EXIT PROGRAM

Example of CALL



Description



➤ Main Program

WORKING-STORAGE SECTION.

01 DATA-1 PIC 99.

01 DATA-2 PIC 99.

01 RESULT PIC 999.

PROCEDURE DIVISION.

ACCEPT DATA-1.

ACCEPT DATA-2.

CALL "SUBPGM"

USING DATA-1,
DATA-2,RESULT.

DISPLAY RESULT.

STOP RUN.

➤ Sub Program

LINKAGE SECTION.

01 P PIC 99.

01 Q PIC 99.

01 R PIC 999.

PROCEDURE DIVISION

USING P, Q, R.

COMPUTE R = P + Q.

END PROGRAM

Static and Dynamic Calls



➤ STATIC CALL

- Identified by call literal. Ex – CALL 'PGMA'
- Default compiler option is NODYNAM and so all the literal calls are considered as static call.
- If sub programs undergoes any changes, sub program and main program need to be recompiled
- Sub modules are link edited with main module
- Size of load module will be large
- Fast
- Less flexible
- Sub programs will not be in its initial stage the next time it is called unless explicitly use INITIAL or CANCEL after each call.

➤ DYNAMIC CALL

- Identified by call variable and the variable should be populated at run time. Ex – MOVE PGMA to X, CALL X.
- Program should be compiled with DYNAM option
- If sub programs undergoes any changes, recompilation of sub program is enough
- Sub modules are picked up during run time from the load library
- Size of load module will be less
- Slow compared to static call
- More flexible
- Sub programs will be in its initial stage the next time every time it is called

Cancel/IS Initial Program



- A sub program may be initialized with certain values which are altered during execution
- CANCEL statement resets all data items to the values they had before execution of the sub program
- Appears in calling program
- Can also be possible with
 - PROGRAM-ID. SUBPRGNAME IS INITIAL PROGRAM. statement in sub program

Call by Reference: Example



WORKING-STORAGE SECTION.

01 DATA-1 PIC 99.

01 DATA-2 PIC 99.

01 RESULT PIC 999.

PROCEDURE DIVISION.

ACCEPT DATA-1.

ACCEPT DATA-2.

CALL "SUBPGM"

BYREFERENCE USING DATA-1,
DATA-2, RESULT.

DISPLAY DATA-1,DATA-2,

RESULT.

STOP RUN.

LINKAGE SECTION.

01 P PIC 99.

01 Q PIC 99.

01 R PIC 999.

PROCEDURE DIVISION

USING P, Q, R.

COMPUTE R = P + Q.

MOVE 0 TO P,Q.

EXIT PROGRAM.

Call by Content: Example



WORKING-STORAGE SECTION.

01 DATA-1 PIC 99.

01 DATA-2 PIC 99.

01 RESULT PIC 999.

PROCEDURE DIVISION.

ACCEPT DATA-1.

ACCEPT DATA-2.

CALL "SUBPGM"

BYCONTENT USING DATA-1,

DATA-2, RESULT.

DISPLAY DATA-1, DATA-2,

RESULT.

STOP RUN.

LINKAGE SECTION.

01 P PIC 99.

01 Q PIC 99.

01 R PIC 999.

PROCEDURE DIVISION

USING P, Q, R.

COMPUTE R = P + Q.

MOVE 0 TO P, Q.

EXIT PROGRAM.

13.2: Programs and Subprograms

Demo



- Main Program
 - CH14PG01.CBL
- Sub Program
 - CH14PG02.CBL

13.2: Programs and Subprograms

Lab



➤ Lab 2.11

Summary



➤ Called Program

- Needs to have a LINKAGE SECTION
 - This section describes variables passed to the program.
- EXIT PROGRAM statement
 - Transfers control back to the calling program

➤ Calling Program

- PROCEDURE DIVISION needs to have a USING clause
 - Identifies variables passed to the program



Review Questions

- **Question 1: Which of the following sections is used to receive the values of variable passed to the program?**
 - Option 1: LINKAGE SECTION
 - Option 2: WORKING-STORAGE SECTION
 - Option 3: FILE SECTION
- **Question 2: The _____ is the last executed statement in the called program**

COBOL

Lesson 14: Table Handling



Lesson Objectives



➤ Familiarizes you with the following:

- Define array using OCCURS clause
- Access and manipulate data stored in an array or table
- Rules to use OCCURS clause in DATA DIVISION
- Use of SEARCH or SEARCH ALL for table look-up

Introduction



➤ Table:

- Collection of logically related entries or a set of values
 - Stored in consecutive storage locations
 - Assigned one data-name

Description



➤ OCCURS integer TIMES:

- Lists number of occurrences of identical fields within a record
- Defines series of related fields with the same format as an array or table
- Cannot be defined at 01 or 77 level
- Reference table elements using Subscript only
- Very useful in to handle tables
- Reduces the amount of writing

Description



➤ Arrays – Example:

- Suppose 72-character input record consists of 24 hourly temperature fields.
- Each field is three positions long:
 - Fields represent temperature for given city at particular hour

Example



- To code record with 24 independent hourly fields is cumbersome

01	TEMP-REC.			
	05	ONE-AM	PIC S9(3).	} 24 entries
	05	TWO-AM	PIC S9(3).	
	
	05	MIDNIGHT	PIC S9(3).	

Example



➤ To obtain average temperature you need to sum 24 fields

```
COMPUTE AVG-TEMP = (ONE-AM +  
                    TWO-AM + ... + MIDNIGHT) / 24
```

Example



- **All 24 fields have the same PICTURE**
 - Define entire 72-position area as array
 - Array is divided into 24 three-position fields, called elements

```
01 TEMP-REC.  
   05  TEMPERATURE OCCURS 24 TIMES PIC S9(3).
```

Description



➤ Identifier TEMPERATURE is an array name

- Use array name along with a subscript to access fields or elements within the array
- Subscript indicates which of the 24 elements to access

Statement

Output

DISPLAY TEMPERATURE (2) 2 AM value

DISPLAY TEMPERATURE (23) 11 PM value

Subscripts: Description



- Valid values are 1 to number of elements in array.
- For array TEMPERATURE valid subscripts are 1 to 24.
- Invalid use of subscripts:
 - DISPLAY TEMPERATURE (0)
 - DISPLAY TEMPERATURE (25)

Subscripts: Description



- Integers or numeric fields with integer values.
- If SUB is defined in Working-Storage:

```
05      SUB  PIC 99      VALUE 5.
```

- Display temperature at 5 AM:

```
DISPLAY TEMPERATURE (SUB)
```

Subscripts: Description



- Use data-name as a subscript
 - This allows its contents to vary
- Each time the value of a data-name changes, TEMPERATURE (SUB) refers to a different element in the array
- Single routine can be used to process all array elements

Examples



➤ OCCURS CLAUSE – Table Definition:

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 STATE-POPULATION-TABLE.  
    05 POPULATION PIC 9(09) OCCURS 24 TIMES.
```

➤ Subscripts – Table Processing:

```
ADD POPULATION(I) TO TOTAL.
```


14.4: Accessing Elements in Array

One-Dimension Table



01 JeansTable.

One-Dimension Table



1	2	3	4

01 JeansTable.

02 Province OCCURS 4 TIMES.

03 SalesValue PIC 9(8)V99.

03 NumSold PIC 9(7).

One-Dimension Table



1	2	3	4

01 JeansTable.
 02 Province OCCURS 4 TIMES.
 03 SalesValue PIC 9(8)V99.
 03 NumSold PIC 9(7).

12346.99

309

--	--

	Province	
SalesValue		NumSold

One-Dimensional Table: Example



DATA DIVISION.

WORKING-STORAGE SECTION.

01 WS-DAILY-TEMPERATURE.

05 WS-HOURLY-TEMPERATURE PIC 9(2)V9
OCCURS 24 TIMES.

One-Dimensional Table: Example



- Initialize one-dimensional table with VALUE clause

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 WS-MONTHS VALUE "JANFEBMARAPRJUNJULAUGSEPOCT  
NOVDEC".  
    05 WS-MONTH OCCURS 12 TIMES PIC A(3).
```

Two-Dimension Table



1	2	3	4

01 JeansTable.

02 Province OCCURS 4 TIMES.

Two-Dimension Table



1		2		3		4	
1	2	1	2	1	2	1	2

01 JeansTable.

02 Province OCCURS 4 TIMES.

03 Gender OCCURS 2 TIMES.

04 SalesValue PIC 9(8)V99.

04 NumSold PIC 9(7).



Two-Dimension Table: Example

DATA DIVISION.

WORKING-STORAGE SECTION.

01 WS-TEMPERATURE.

05 WS-DAYS OCCURS 7 TIMES.

10 WS-HOURS OCCURS 24 TIMES.

15 WS-TEMP PIC 9(2)V9.

Three-Dimension Table



1	2	3	4

01 JeansTable.

02 Province OCCURS 4 TIMES.

Three-Dimension Table



1		2		3		4	
1	2	1	2	1	2	1	2

01 JeansTable.

02 Province OCCURS 4 TIMES.

03 Gender OCCURS 2 TIMES.

Three-Dimension Table



1						2						3						4							
1			2			1			2			1			2			1			2				
1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3		

01 JeansTable.

02 Province OCCURS 4 TIMES.

03 Gender OCCURS 2 TIMES.

04 Colour OCCURS 3 TIMES.

Three-Dimension Table



1				2				3				4			
1		2		1		2		1		2					
1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	

01 JeansTable.

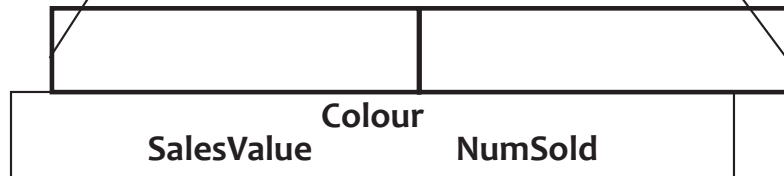
02 Province OCCURS 4 TIMES.

03 Gender OCCURS 2 TIMES.

04 Colour OCCURS 3 TIMES.

05 SalesValue PIC 9(8)V99.

05 NumSold PIC 9(7).



Format



➤ PERFORM VARYING

```
PERFORM  para-1  [THRU para-2]
      VARYING  identifier-1
                index-name-1
                FROM  identifier-2
                       index-name-2
                       literal-1
                       BY
                identifier-3, literal-2
                UNTIL condition
```

Example 1



➤ Example 1:

```
MOVE ZEROS TO TOTAL-TEMP.  
PERFORM VARYING SUB  
    FROM 1 BY 1 UNTIL SUB > 24  
    ADD TEMPERATURE (SUB) TO TOTAL-TEMP.  
COMPUTE AVG-TEMP = TOTAL-TEMP / 24.
```

Example 2



➤ **Example 2:**

```
PROCESS-PARA.  
  MOVE 0 TO TOTAL.  
  PERFORM ADD-PARA VARYING I FROM  
    1 BY 1 UNTIL I > 24.  
    .  
    .  
ADD-PARA.  
  ADD POPULATION(I) TO TOTAL.
```

14.5: Using One-Dimensional Array

Demo



- Program using one dimensional array
 - CH15EX01.CBL

14.6: REDEFINES clause

Description



- Allows data elements to share same storage areas
- For both data items, storage area requirement should be the same
- Define both items on same level
 - Defined item must immediately follow the item it redefines

Example



➤ Example :

```
WORKING-STORAGE SECTION.
```

```
    05 ITEM-1 PIC X(05).
```

```
    05 ITEM-3 REDEFINES ITEM-1 PIC 9(05).
```

```
    05 ITEM-2 PIC X(05).
```

➤ Use REDEFINES to assign constant values to table elements

Example



WORKING-STORAGE SECTION.

01 MONTH-NAME.

05 FILLER PIC X(09) VALUE IS 'JANUARY '

05 FILLER PIC X(09) VALUE IS 'FEBRUARY '

:

05 FILLER PIC X(09) VALUE IS 'NOVEMBER '

05 FILLER PIC X(09) VALUE IS 'DECEMBER '

01 MONTH-TABLE REDEFINES MONTH-NAME

02 MONTH PIC X(09) OCCURS 12 TIMES.

14.6: OCCURS with VALUE and REDEFINES

Demo



- Program using OCCURS with VALUE and REDEFINES
 - CH15PG02.CBL

Description



➤ COBOL permits upto seven levels of OCCURS

- For example, define an array to store hourly humidity readings for each day of a given youek
- You need a two-dimensional array with 7 rows, each with 24 columns

Description



➤ Define array as follows:

```
01      HUMIDITY-ARRAY.  
      05      DAY-IN-WEEK OCCURS 7 TIMES.  
          10      HOUR OCCURS 24 TIMES.  
              15  HUMIDITY      PIC S9(3).
```

➤ 24-hour figures for each DAY-IN-WEEK

- Each consists of HUMIDITY three integers long

Double-Level Subscripts: Description



- To access humidity, use data-name on lowest OCCURS level or any data-name subordinate to it.
 - Either HUMIDITY or HOUR could be used
- As HUMIDITY is defined with two OCCURS, use two subscripts to access each hourly temperature
 - For example, HUMIDITY (3, 5) refers to the humidity for the third day, fifth hour

Table with Subscripts



Humidity Array						
Hour	1 AM	2 AM	3 AM	4 AM	...	12 Mid
Day-In-week						
Day 1 (Sun)	(1,1)	(1,2)	(1,3)	(1,4)	...	(1,24)
Day 2 (Mon)	(2,1)	(2,2)	(2,3)	(2,4)	...	(2,24)
Day 3 (Tue)	(3,1)	(3,2)	(3,3)	(3,4)	...	(3,24)
Day 4 (youd)	(4,1)	(4,2)	(4,3)	(4,4)	...	(4,24)
...
Day 7 (Sat)	(7,1)	(7,2)	(7,3)	(7,4)	...	(7,24)

14.7: Multiple-Level OCCURS

Accessing Double-Level Array



- Find average humidity for entire youek
- Add all array entries and divide by 168 (7 x 24)
- Use nested PERFORMS
 - First PERFORM varies major (row) subscript from 1 to 7
 - Second PERFORM varies minor (column) subscript from 1 to 24

Example



➤ Code Snippet:

```
PROCEDURE DIVISION.  
  MOVE 0 TO TOTAL.  
  PERFORM INITIAL-HUMID VARYING DAY-SUB FROM 1 BY 1  
    UNTIL DAY-SUB > 7.  
  COMPUTE WEEKLY-AVERAGE = TOTAL / 168  
  
INITIAL-HUMID.  
  PERFORM ADD-PARA VARYING HOUR-SUB FROM 1 BY 1  
    UNTIL HOUR-SUB > 24  
  
ADD-PARA.  
  ADD HUMIDITY (DAY-SUB, HOUR-SUB) TO TOTAL.
```

Description



➤ Indexing, COBOL's alternative to subscripting.

➤ INDEX

- Data item associated with a table or particular table dimension with the use of INDEXED BY phrase of an OCCURS clause.

➤ **Example:**

```
01 SG-TOTAL-TABLE.  
   05 SALES-GIRL OCCURS 5 TIMES INDEXED BY  
                                           SG-  
COUNTER.  
   10 SALES-GIRL-TOTAL PIC 9(7)V99.
```

Description



➤ Indexes:

- No separate definition in the DATA DIVISION
- Represents displacement from the address of the first table element
- Naturally associated with a table via the OCCURS clause
- Compiler automatically defines data item, specifies in the INDEXED
- Data item comprises a USAGE INDEX
- If one table level is indexed, all other levels also need to be indexed
- Indexes are manipulated only by the SET, SEARCH and PERFORM statements

Description



➤ Index data-Item:

- Defined like data items in the DATA DIVISION
- Used to preserve the value of an index
- Elementary item is defined with “USAGE IS INDEX” clause without PICTURE.
- Referred directly only in SET and PERFORM verbs.
- Example :
 - 01 I USAGE IS INDEX

SET Verb: Description



➤ **Sets, increments or decrements the value of indexes.**

- Format 1:

```
SET index-name-1 [, index-name-2] ... TO integer
```

- Example:

```
SET I TO 4  
SET I1, I2, I3, TO 3.
```

- Format 2:

```
SET identifier-1 [, identifier] .. TO index-name-1
```

SET Verb: Description



- Example:

```
SET A TO I
```

- Format 3:

```
                                UP BY    id
SET index-name [, index-name-2] DOWN BY integer
```

- Example:

```
SET I UP BY 2
SET I DOWN BY A.
```

Description



- To Locate Specific table entry
 - Use SEARCH Verb
- Search Technique
 - Sequential/Binary
- The identifier used with the SEARCH verb is the table entry name specified on the OCCURS level, not on the 01 level

SEARCH Statement: Description



➤ Format

```
SEARCH identifier-1  
      [AT END imperative-statement-1]  
      WHEN condition-1 { imperative-statement-2 ... }  
                        CONTINUE
```

- Performs linear search on a one-dimensional table
- Use in place of PERFORM to search tables

SEARCH Statement: Description and Rules



➤ Rules to use SEARCH statement:

- Apply to a table only if it is index using INDEXED BY
- Initialize the index using SET before you apply a SEARCH statement
- Index has an unpredictable value if the SEARCH statement terminates without finding the particular element in a table
- SEARCH statement does a sequential or serial table search

Description



- Each entry (usually starting with first) is checked in order until:
 - Specified condition is met
 - Table is completely searched
- Best used when:
 - Entries are not in order by table argument value (numerical or alphabetical order)
 - Entries are organized
 - First values are searched most frequently
 - Search time is minimized

SEARCH Statement: Example



➤ Example

```
SET X1 TO 1.  
SEARCH TABLE-ENTRIES  
  AT END MOVE 0 TO WS-SALES-TAX  
  WHEN ZIP-IN = WS-ZIPCODE (X1)  
  COMPUTE .....
```

Index vs Subscript



Subscript	Index
Represents an occurrence of an array or table element	Represents a displacement from the first address in the array or table
Defined in a separate WORKING-STORAGE entry	Defined along with the OCCURS for the array or table
To change a subscript's value, use a PERFORM ... VARYING or any of the following:	To change an index value, use a PERFORM ... VARYING or any of the following:
MOVE 1 TO SUB ADD 1 TO SUB SUBTRACT 1 FROM SUB	SET X1 TO 1 SET X1 UP BY 1 SET X1 DOWN BY 1

SEARCH ALL Statement: Description



- When table entries are arranged in sequence by some field, such as EMP-ID, the most efficient type of lookup is a binary search
 - SEARCH ALL verb is used to perform binary search
 - Use ASCENDING or DESCENDING options with the table declaration
 - During search, table is arranged either in ascending or in descending order

SEARCH ALL Statement: Format



➤ Format

```
SEARCH ALL identifier-1  
      [AT END imperative-statement-1]  
      WHEN condition-1  
      { imperative-statement-2  
        NEXT SENTENCE      }
```

SEARCH ALL Statement: Example



- **Find a particular zip code is present in the table or not, the following procedure statement may be used:**

```
SEARCH ALL TABLE-ENTRIES  
  AT END DISPLAY "ZIP CODE IS NOT PRESENT"  
  WHEN TABLE-ENTRIES(X1) = WS-ZIPCODE  
  DISPLAY ZIPCODE.
```


SEARCH ALL Statement: Limitations



- Only one WHEN clause can be used
- The condition following the word WHEN can only test for equality
- IF the condition following the WHEN is a compound conditional, then
 - Each part of the conditional can only consist of a relational test that involves an equal condition
 - The only compound condition permitted is with ANDs and not Ors
- OCCURS item and its index, that define the table argument, must appear to the left of the equal to sign
- VARYING option cannot be used

Search and Search ALL: Differences



Search	Search All
Performs a serial search	Performs a binary search
Table entries need not be in any sequence	Tables entries must be in sequence by the table argument or even the table function. The field that is in sequence is specified in an ASCENDING or DESCENDING KEY clause as part of the OCCURS entry
Can include any relational test with the WHEN clause (<, >, =, <=, >=) or any compound conditional	Does not need a SET prior to the SEARCH ALL
	Can only have a single = condition tested with the WHEN clause
May include multiple WHEN clauses	May only have one WHEN clause

Table Rules for Multi-Dimensional Array



- **When describing an area of storage, more than one level of OCCURS may be used.**
- **As many as seven levels of OCCURS are permitted with COBOL 85, and as many as three levels are permitted with COBOL 74.**
- **RULES FOR USING A DOUBLE-LEVEL OCCURS**
 - If an item is defined by a double-level OCCURS clause, it must be accessed by two subscripts.
 - The first subscript refers to the higher-level OCCURS; the second subscript refers to the lower-level OCCURS.
 - The subscripts must be enclosed in parentheses.
 - Subscripts may consist of positive integers or data-names with positive integer contents.
 - On most systems, the left parenthesis must be preceded by at least one space; similarly, the right parenthesis must be followed by a period, if it is the end of a sentence, or at least one space. The first subscript within parentheses is followed by a comma and a space.

Tables with Varying Sizes



- When table with varying size is required, OCCURS DEPENDING ON clause is used.
- Example: Assuming the number of zones a person is handling vary, say from 2-5

```
01 EMPLOYEE.  
   05 EMP-ID PIC X(5) OCCURS 4 TIMES.  
      10 A PIC 9.  
      10 ZONE OCCURS 2 TO 5 TIMES  
                                     DEPENDING ON A.
```

- Result: ZONE occurs depending on the value of A and varies from two to five

COBOL Tables: Tables with Varying Sizes



➤ **Format:**

```
OCCURS      Integer1 TIMES  
            Integer1 TO integer2 TIMES  
            DEPENDING ON data-name1
```

- Integer1 and Integer2 must be positive
 - Integer1 must have value less than Integer2
 - Data-name1 must be defined in DATA DIVISION
- **Initial value can not be assigned to item containing DEPENDING ON clause or item subordinate to them**
- **Entry containing the DEPENDING ON clause and all its subordinate entries can not be redefined**

14.9: Table Searching

Demo: Program on Tables



➤ Program on Tables

- GRADES.CBL

14.9: Table Handling

Lab



➤ Lab 2.12



Summary

➤ **OCCURS clause**

- Specifies repeated occurrence of items with the same format
- Use in 02-49 level entries
- Use with elementary or group items
- COBOL- 85 permits seven levels of OCCURS

➤ **REDEFINES clause**

- Enables different data elements to share the same storage area



Summary

- **SEARCH statement**
 - Use SET or PERFORM ... VARYING to change index value
 - SET index to 1 before you use SEARCH
- **SEARCH ALL statement**
 - Used to perform binary search
 - Can test only an equal condition



Review Questions

- Question 1: Determine the total number of bytes in the following cases:

01	REC-1		
	05	FIRST-GROUP.	
		10	A1 PIC X(4).
		10	A2 PIC 99.
	05	SECOUND-GROUP REDEFINES FIRST-GROUP	
	10	A3	PIC 999.
	10	A4	PIC 999.

- Option 1 : 6
- Option 2 : 12
- Option 3 : 5

COBOL

Lesson 15: String Handling



Lesson Objectives



➤ To learn string handling with the following:

- EXAMINE VERB
- STRING
- UNSTRING
- INSPECT verb



Purpose and Syntax for Format 1

➤ Purpose

- To scan a string to find the number of occurrences of a given character in it.
- It can also be used to replace some or all occurrences of the said character by another character

➤ Syntax

- Format 1:

EXAMINE {identifier-1} TALLYING

{
LEADING
UNTIL FIRST
}

literal-1.

15.1: EXAMINE Verb
Example



WORKING-STORAGE SECTION.

01 EXAMINE-DATA.

05 E-1 PIC A(10) VALUE 'ABCADCADCD'.

05 E-2 PIC 9(10) VALUE 0150250350.

.....

PROCEDURE DIVISION.

TEST-1.

EXAMINE E-1 TALLYING ALL 'D'.

IF TALLY NOT EQUAL TO ZERO

PERFORM PROCESS-PARA

ELSE

PERFORM EXIT-PARA.

Example for Format 1



01 EXAMINE-DATA.

05 E-1 PIC A(10) VALUE 'ABCADCADCD'.

05 E-2 PIC 9(10) VALUE 0150250350.

EXAMINE E-2 TALLYING LEADING 0.

EXAMINE EXAMINE-DATA TALLYING UNTIL FIRST 2.

15.1: EXAMINE Verb

Syntax for Format 2



➤ Format 2

EXAMINE {identifier} REPLACING

Literal-1 BY literal-2

{
ALL
LEADING
FIRST
UNTIL FIRST
}

Example for Format 2



01 EXAMINE-DATA.

05 E-1 PIC A(10) VALUE 'ABCADCADCD'.

05 E-2 PIC 9(10) VALUE 0150250350.

—EXAMINE E-1 REPLACING ALL “A” BY “E”.

—EXAMINE E-1 REPLACING LEADING “A” BY “E”.

15.1: EXAMINE Verb

Example for Format 2



01 EXAMINE-DATA.

05 E-1 PIC A(10) VALUE 'ABCADCADCD'.

05 E-2 PIC 9(10) VALUE 0150250350.

EXAMINE E-1 REPLACING FIRST "D" BY "L".

EXAMINE E-1 REPLACING UNTIL FIRST "D" BY "L".

Syntax and Example for Format 3



➤ Format 3

EXAMINE {identifier} TALLYING

{ ALL
LEADING
UNTIL FIRST }

REPLACING BY literal-2

➤ Example

05 E-1 PIC A(10) VALUE 'ABCADCADCD'.

EXAMINE E-1 TALLYING ALL "D" REPLACING BY "M".

Introduction



- A string refers to sequence of characters
- Any data-name whose USAGE is DISPLAY is regarded as string in COBOL
- “String handling” refers to each of the following
 - Comparison
 - Concatenation
 - Join two or more strings
 - Segmentation
 - Reverse of Concatenation.

Introduction



- Scanning
 - Search a string for the appearance of a specific character or a group of characters
 - Count the number of occurrences of character (s).
- Replacement
 - Replace a specified character (s) by another character (s))

➤ COBOL provides three verbs for string handling:

- INSPECT
- STRING
- UNSTRING

Description



- **Combine two or more strings to form one string.**
- **Format :**

STRING id-lit-1 [id-lit-2]
DELIMITED BY id-lit-3 [id-lit-4 [id-lit-5]
DELIMITED by id-lit-6]

INTO identifier-7 [WITH POINTER identifier-8]
[ON OVERFLOW imperative-statement]

15.2: STRING Verb

Description



- **Use DISPLAY for Sending or Receiving fields.**
- **If part of receiving field does not get any character, original characters are retained.**

Description



➤ POINTER phrase:

- Specifies position in receiving field where onwards the characters are stored.
- Indicated by identifier-2
- If not indicated, defaults to 1
- If initially, identifier-2 contains a value less than 1 or greater than size of id-1, no characters are transferred
- Count the number of characters actually moved to the receiving field if it is initialized at 1

Description



➤ OVER FLOW phrase:

- Specifies the operation (s) to perform if the receiving field is not large enough to accommodate the result
 - Executed if end of identifier-1 is reached and some characters are yet to be transferred
- If omitted, after the STRING statement is terminated, control is transferred to next statement in the sequence

Example



01	W01-CNT1	PIC	X(4)	VALUE	“ABCD”.
01	W02-CNT2	PIC	X(4)	VALUE	“MA IN ”.
01	W03-CNT3	PIC	X(9)	VALUE	“121,34,56”.
01	W04-CNT4	PIC	X(14)	VALUE	SPACES.

15.2: STRING Verb

Example



```
STRING  W01-CNT1, W02-CNT2, W03-CNT3  
        DELIMITED BY    “ “, “,”    INTO W04-CNT4
```

➤ **Output: ABCDMA121**

15.2: STRING Verb

Demo



- **Program using STRING verb**
- **CH16PG01.CBL**

Description & Syntax



➤ **Splits one string to many substrings.**

- Format

```
UNSTRING identifier-1  
[DELIMITED BY [ALL] id-lit-2 [OR [ALL] id-lit-3] ....]  
INTO identifier-4      [, DELIMITER IN identifier-5]  
                      [, COUNT IN identifier-6]  
    identifier-7      [, DELIMITER IN identifier-8]  
                      [, COUNT IN identifier-9]]....  
WITH POINTER identifier-10]  
[TALLYING IN identifier-11]
```

Description



- **Data in identifier-1 is separated and placed in multiple receiving fields i.e. identifier-4, identifier-7 etc.**
- **Sending field i.e. identifier-1 can only be alphanumeric with usage DISPLAY.**
- **Receiving fields i.e. identifier-4 etc. can be alphabetic, numeric and alphanumeric with usage DISPLAY.**
- **identifier-6, identifier-9, identifier-10 and identifier-11 must be elementary numeric integer data items.**

Description



- **TALLYING phrase:**
 - Value of indicated identifier is the initial value plus the number of fields acted upon
- **The [WITH POINTER identifier] and [ON OVERFLOW imperative-statement] clauses:**
 - Used in the same way as with the STRING

Example



```
01 STUDENT-RECORD.  
   05 FIRST-NAME      PIC X(10).  
   05 MIDDLE-NAME     PIC X(10).  
   05 LAST-NAME       PIC X(12).  
   05 STANDARD        PIC XX  
01 FREE-FORM-RECORD  PIC X(34)  VALUE  
   "AKASHbbbbbbSATISH,MAHLOTRA,10".
```


15.3: UNSTRING Verb

Example (contd..)



UNSTRING FREE-FORM-RECORD

DELIMITED BY ALL SPACES OR ‘,’

INTO FIRST-NAME

MIDDLE-NAME

LAST-NAME

STANDARD

15.3: UNSTRING Verb

Example



➤ Output:

- | | |
|---------------|------------|
| — FIRST-NAME | - AKASH |
| — MIDDLE-NAME | - SATISH |
| — LAST-NAME | - MAHLOTRA |
| — STANDARD | - 10 |

15.3: UNSTRING Verb

Demo



➤ Program using UNSTRING verb

- CH16PG03.CBL



Description & Syntax

- Replace a character in a string.
- Count occurrences of a character in a string.
- Format :

INSPECT	identifier-1	TALLYING	identifier-2	FOR
CHARACTERS	$\left(\begin{array}{c} \{ \text{BEFORE} \} \\ \{ \text{AFTER} \} \end{array} \right)$	INITIAL	$\left\{ \begin{array}{c} \text{identifier-3} \\ \text{literal-1} \end{array} \right\}$	$\left. \right)$
$\left\{ \begin{array}{c} \text{ALL} \\ \text{LEADING} \end{array} \right\}$	$\left\{ \begin{array}{c} \text{identifier-4} \\ \text{literal-2} \end{array} \right\}$	$\left(\begin{array}{c} \{ \text{BEFORE} \} \\ \{ \text{AFTER} \} \end{array} \right)$	INITIAL	$\left\{ \begin{array}{c} \text{identifier-5} \\ \text{literal-3} \end{array} \right\}$
				$\left. \right)$

Description



- All identifiers must be elementary items with usage DISPLAY. Contents of identifier-1 can be counted for the specified character
- In case of ALL phrase:
 - Characters specified in identifier-3 or literal-1 are matched against characters of identifier-1
 - For each match, value of identifier-2 is incremented by 1

15.4: INSPECT Verb

Description



- For LEADING phrase
 - Only the contiguous occurrence is examined
- For character phrase
 - Identifier-2 is incremented by 1 for each character in identifier-1
- BEFORE or AFTER limits the limits the length of identifier-1 to be searched

Example



```
01 CTR-1      PIC 9 VALUE 0.  
01 WS-NAME    PIC X(10).
```

```
ACCEPT WS-NAME.
```

```
INSPECT WS-NAME TALLYING CTR-1 FOR ALL SPACES.
```

Lab



➤ Lab 2.12

Summary



➤ INSPECT verbs

- Tally or replace occurrences of a single or groups of characters in a data field.

➤ **STRING verbs**

- Form one longer string from two or more character strings.

➤ **USTRING verbs**

- Split one string into many substrings.

Review Questions



- Question 1: Along with which COBOL verb, can TALLYING and REPLACING verbs be used ?
 - Option 1: STRING
 - Option 2: UNSTRING
 - Option 3: INSPECT
- Question 2: POINTER option is used with INSPECT verb.
 - True / False
- Question 3: Which verb is used to replace some or all occurrences of the said character by another character?
 - INSPECT
 - STRING
 - EXAMINE

COBOL

Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
29-May-2009	5.0	Arjun Singh	Content Creation
08-Oct-2009	5.0	CLS Team	Review and template application
06-July-2010	6.0	Vaishali Kasture	Review and Modification of Labs
22 nd August 2012	6.1	Vaishali Kasture	Revamped after Assignment Review

Table of Contents

Document Revision History.....	2
Table of Contents.....	3
Getting Started.....	5
□ Overview.....	5
□ Setup Checklist for COBOL.....	5
□ Instructions.....	5
□ Learning More (Bibliography if applicable).....	5
□ Prerequisites.....	5
Lab 1: Sequential File Creation	6
□ 1.1: Write a program to input the data given below:.....	6
□ 1.2: Write a program to input the data given below:.....	8
Lab 1.3 Analysis and Debugging	10
□	10
□ 1.4: Modify the program cobass01.....	10
□ 1.5: Modify program cobass01. 13.....	
□ 1.6: Print Item Class wise Report. 14.....	
Lab 1.7 Analysis and Debugging	16
□ 16.....	
□ TO DO: The participants have to analyze the program, remove the error and successfully execute the program. 16.....	
□ TO DO: The program has to be debugged by the participants till it becomes error-free. 16.....	
Lab 1.8 Report-ControlBreak	17
Lab 1.9: Indexed File from Sequential File	18
Lab 1.10: Create an Indexed File from Sequential File	19
1.11: Indexed File –Append & Analysis	20
Lab 1.12: Updating Indexed File	21
□ Update Index file “itemmast.dat” from a sequential file “GRN.DAT”......	21
Lab 1.13: Order booking for a Customer	22
□ Order booking by Marketing for a customer.	22
Lab 1.14 Enhancement Assignment.....	25
Lab 1.15 Analysis and Debugging	26
Lab 1.16 Bank Transaction Updation and Report Generation.....	27
Lab 1.17 Working with 3 Indexed Files.....	27
Lab 1.18 Online Bus Booking	34
Lab Assignment 1.19: Call Statement-Debugging.....	34
Lab Assignment 1.20: Call Statement-Enhancement Assignment.....	39
Lab 1.21: Table Handling Lab Assignment.....	40
Lab Assignment 1.22: Table Handling- Analysis Assignment.....	42
Lab 1.23: String Handling	43
Lab 1.24: Mobile Services.....	44
□ Stretched Assignment 1:.....	46
□ Itemwise Sales Order Report.....	46
□ Stretched Assignment 2: Write a program to print the following report using the file created in program COBASS2.	47
□ Case Study.....	47
Appendices	54
□ Appendix A: Breakup of Marks.....	54
□ 1. Compiling, Linking, and Executing COBOL programs.....	54
□ 1. Debugging a COBOL program.....	61

□	3. Invoking Subroutines	65
	COBOL STANDARDS.....	56

Getting Started

- **Overview**

This lab book is a guided tour for learning COBOL. It comprises solved examples and 'To Do' assignments. Follow the steps provided in the solved examples and work out the 'To Do' assignments given.

- **Setup Checklist for COBOL**

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- NA

Please ensure that the following is done:

- NA

- **Instructions**

- Create a directory by your empid_name in drive <drive> for storing all COBOL programs.

- **Learning More (Bibliography if applicable)**

NA

- **Prerequisites**

It is expected that participants know the following:

- MVS
- JCL

Day1:Lab 1:Working with Sequential Files and Report Handling

Goals

- Learning COBOL assignments in a real world COBOL application system. This is a basic system with minimum functional complexities.
- To develop a COBOL business application

Introduction:

A company named "ABC Co. Ltd" manufactures computer parts and sells them. The company's stockyard gets orders from customers which are collected by the marketing department.

Everyday the stockyard receives the material through the manufacturing department. These items are received through a document called "Goods Receipt Note" (GRN). The GRN contains the GRN number, date, item number, and the quantity of the item received. Using the GRN, the Item Master is updated to reflect the new inventory stock in hand for a particular item. That is, the **received qty** and the **dispatch qty** fields of the **Item master** are initially set to zero at the beginning of the month. Subsequently, these fields are updated on a regular basis during the month.

Marketing department gets the stock status every day after the updating of the stock based on the Goods Receipt Note. Based on the stock position, order balance, customer priority etc., the Marketing department sends **delivery instructions** to the Stockyard. The **delivery instruction** contains the order, which has to be processed, along with the items and quantities, based on priority of orders. In short, orders are not automatically processed. Depending on the availability of items, a **status flag** is updated to indicate whether the order requirements have been fully/partially supplied. The stockyard prepares the invoice from the Delivery Instruction and this also reduces the stock balance.

At the end of the month, a fresh **stock master file** is created and the last month's file is taken to backup. Final updation of the **Item master** takes place on a monthly basis.

Files are created, updated, and Reports are generated from this data.

- **Day 1:Lab 1.1: Write a program to input the data given below:**
[Duration 1 hour]

Record structure:

IT-ITEMNO	Itemno	pic x(6)	format : char 1-2 "Item Class", char 3 – 6 4 digit serial no.
IT-DESC	Item description	pic x(25)	
IT-MONOPBAL	Item month op. Bal qty	pic 9(6)	
IT-RCPTQTY	Item month receipts qty	pic 9(6)	
IT-DESPQTY	Item month dispatch qty	pic 9(6)	
IT-RATE	Item rate	pic 9(5)v99	

IT-PROCMY	Processing MMYYYY	pic 9(6)	
-----------	-------------------	----------	--

Item's Classes are as follows:

10 – Internal Components, 20 - Input Devices, 30 - Cards Boards, 40 – Storage Devices, 50 – Output Devices, 60 – Cables, 70 – Storage Media

Sample data:

IT-ITEMNO		IT-DESC	IT-MONOPBAL	IT-RCPTQTY	IT-DESPQTY	IT-RATE	IT-PROCMY
IT-Class	IT-SR-No						
10	0001	Celeron 1 GHz	80	25	45	3300	012004
10	0002	P III 833 MHz	40	5	10	3400	012004
10	0003	P IV 2 GHz	36	7	14	4500	082004
10	0004	RAM 128 MB	100	10	40	900	012004
10	0005	RAM 256 MB	70	30	20	1200	112003
10	0006	RAM 512 MB	70	20	30	1800	012004
20	0001	Logitech serial mouse	20	5	11	300	112003
20	0002	Logitech PS/2 mouse	15	3	5	400	122003
40	0001	Sony SDT – 9000	2	3	4	8000	122003
60	0001	40 pin IDE cable	100	40	80	40	012004
60	0002	80 pin IDE cable	90	30	60	50	122003
60	0003	Parallel port cable	115	20	60	80	012004

Solution:

Step 1: Check that the data is of a record structure as shown in the above table. Item class can have values as multiples of 10 from 10 – 70 inclusive only.

Step 2: Input all the items in the data given in the above table. Once the data of a record is complete, the program should display the full group as an item. The program will then ask the user "Do you want to Input More (Y/N)".

Step 3: Select an option (Y/N). The program should accept the user's response.

- If the response is "Y", then the program should accept the details on another item and repeat the above steps.

- If the response in “N”, then the program should stop.
- Perform a Peer Review and record the TestCases in TestCase.xls
- **Day 1:Lab 1.2: Write a program to input the data given below: [Duration 1 hour]**

Record structure:

CU-CUSTNO	Custno	pic x(6)	format : char 1-2 The 1st two letters of customer name, char 3-6 4 digit serial no.
CU-CUSTNAME	Cust Name	pic x(25)	
CU-ADDLIN1	Cust add. Line 1	pic x(25)	
CU-ADDLIN2	Cust add. Line 2	pic x(25)	
CU-REGION	Region	pic x(3)	Coded as 3 Char of city as below only: MUM,DEL,CHN,CAL,BLR,HYD,PUN.
CU-MAXCR	Max credit balance	pic 9(6)	
CU-CURRBAL	Current Balance	pic 9(6)v99	

CU-CUSTNO	CU-CUSTNAME	CU-ADDLIN1	CU-ADDLIN2	CU-REGION	CU-MAXCR	CU-CURRBAL
AB0034	ABC Computer Services	MIDC,	Andheri	MUM	1,50,000	1,00,000
DA7201	Datacomp Services	Anna Salai,	Teynampet	CHN	2,00,000	1,75,000
NE9944	Network Pvt. Ltd	Vasant Vihar,	Thane	MUM	2,00,000	1,00,000
EX1021	Excellent Computers	MIDC,	Bhosari	PUN	1,25,000	90,000

Solution:

Step 1: Check that the data is of a record structure as shown in the above table.

Step 2: Input all items of the data given in the above table. Once the data of a record is complete, the program will display the full group as an item. The program will then ask the user “Do you want to Input More (Y/N)”.

Step 3: Select an option (Y/N). It should accept the user’s response.

If the response is “Y”, then the program should accept the details on another customer and repeat the above steps. If the response is “N”, then the program should stop.

Day 1 : Lab 1.3 Analysis and Debugging [Duration ½ hour]

TO DO: The participants have to analyze the program, remove the error and successfully execute the program.

TO DO: All the participants have to document the error in the excel sheet along with steps taken to resolve the error.

Files for participants: Sequential File Analysis.cbl

-

- **Day 2: Lab 1.4: Modify the program [Duration 3 hours]**

Modify the program cobass01 to have three choices as listed below. The program should function as per the steps of choices listed below.

When the user executes the program the following screen will be displayed:

A. Input Data
B. Print Data
Q. Quit.

Enter your choice : _

Ensure that the choice entered is A, B, or Q.

Choice 1: If the user enters 'A', then follow the steps given below.

Input data:

Step 1: Open the file "itemseq.dat" in extend mode.

Step 2: Key in the data as in COBASS01. However, instead of displaying on the screen it is to be written to a sequential file named "itemseq.dat" with the same format as the group. The program will then ask the user "Do you want to enter more (y/n)?".

Step 3: Select an option (Y/N). It will accept the user's response.

- If the user response is Y, then the program will once again take input and write to file.
- If the user response is N, then the program will close files and go back and display the menu once again.

Choice 2: If the user enters 'B', then follow the steps given below.

Print Data:

Step 1: Open the sequential file “itemseq.dat” in **input mode** and the “itemrep.dat” in **output mode**.

- It should read the data sequentially until End of file.
- For every valid record that is read, it should print as per the format given below.
- There should be a max of 5 item records per page. The **Run Date** is the system date and the **month/ Year** is the mmyyyy of the data in itemseq.dat’s first record.
- At the end of the report, a grand total of the closing value field will be printed and all files will be closed.

ABC CO. LTD.							Date : dd/mm/yyyy
List of Items in Stock For <month > / Year							Page : 999
Itemno	Item Description	<----- Month Details ----->				Rate	Clo. Value
		Op.Bal	Receipts	Dispatch	Clo-Bal.		
xxxxxx	xxxxxxxxxxxxxxxxxxxxxx	999999	999999	999999	999999	99999	9999999
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
PAGE TOTAL :							99999999

List of Items in Stock For <month > / Year							Page : 999
Itemno	Item Description	<----- Month Details ----->				Rate	Clo. Value
		Op.Bal	Receipts	Dispatch	Clo-Bal.		
xxxxxx	xxxxxxxxxxxxxxxxxxxxxx	999999	999999	999999	999999	99999	9999999
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
PAGE TOTAL :							99999999
GRAND TOTAL :							99999999

Choice 3: If the user enters ‘Q’, the program should terminate.

Sample output

ABC CO. LTD.

List of Items in Stock For 01/2004

Date : 08/01/2004

Page : 1

<----- Month Details ----->							
Itemno	Item Description	Op.Bal	Receipts	Dispatch	Clo. Bal.	Rate	Clo. Value
100001	Celeron 1 GHz	80	25	45	60	3300.00	198000.00
100002	P III 833 MHz	40	5	10	35	3400.00	119000.00
100003	P IV 2 GHz	36	7	14	29	4500.00	130500.00
100004	RAM 128 MB	100	10	40	70	900.00	63000.00
100005	RAM 256 MB	70	30	20	80	1200.00	96000.00

PAGE TOTAL : 606,500.00

List of Items in Stock For 01/2004

Page : 2

<----- Month Details ----->							
Itemno	Item Description	Op.Bal	Receipts	Dispatch	Clo. Bal.	Rate	Clo. Value
100006	RAM 512 MB	70	20	30	60	1800.00	108000.00
200001	Serial mouse	20	5	11	14	300.00	4200.00
200002	PS/2 mouse	15	3	5	13	400.00	5200.00
400001	Sony SDT-9000	2	3	4	1	8000.00	8000.00
600001	40 pin IDE000	100	40	80	60	40.00	2400.00

PAGE TOTAL : 127,800.00

List of Items in Stock For 01/2004

Page : 3

<----- Month Details ----->							
Itemno	Item Description	Op.Bal	Receipts	Dispatch	Clo. Bal.	Rate	Clo. Value
600002	80 pin IDE000	90	30	60	60	50.00	3000.00
600003	Parallel port	115	20	60	75	80.00	6000.00

PAGE TOTAL : 9,000.00

GRAND TOTAL : 743,300.00

• **Day 2: Lab 1.5: Modify program [Duration 1.5 hours]**

Modify program cobass02 to have three choices as listed below. The program should function as per the steps of choices listed below.

When the user executes the program the following screen is displayed:

A. Input Data B. Print Data Q. Quit. Enter your choice: <u> </u>
--

Ensure that the choice entered is A, B, or Q.

Choice 1: If the user enters 'A', then follow the steps given below:

Input data:

Step 1: Open the file "**custseq.dat**" in extend mode.

The data is entered as in COBASS01. However, instead of displaying on the screen it is to be written to a sequential file named "**custseq.dat**" with the same format as the group. The program will then ask "Do you want to enter more (y/n)?".

Step 2: Select an option (Y/N). It will accept the user's response.

- If the user response is Y, then the program will once again take input and write to the file.
- If the user response is N, then the program will close files and go back and display the menu once again.

Choice 2: If the user enters 'B', then follow the steps given below.

Print Data:

Step 1: The program will open the sequential file "**custseq.dat**" in **input mode**, and the "**custrep.dat**" in **output mode**.

- It should read the data sequentially until eof.
- For every valid record read, it should print as per the format given below.
- There should be a maximum of 3 customer records per page. The **Run Date** is the system date.
- At the end of the report, a grand total of the Current Balance field should be printed and all files closed.

ABC CO. LTD. List of Customers				Date : dd/mm/yyyy Page : 999	
Custno	Customer Name	Region	Address Line 1 Address Line 2	Credit-Limit	Current Bal.
xxxxxx	xxxxxxxxxxxxxxxxxx	xxx	xxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxx	999999	999999

:	:	:	:	:	:
:	:	:	:	:	:
:	:	:	:	:	:
:	:	:	:	:	:
				PAGE TOTAL:	99999999
List of Customers				Page : 999	
Custno	Customer Name	Region	Address Line 1 Address Line 2	Credit-Limit	Current Bal.
xxxxxx	xxxxxxxxxxxxxxxxxx	xxx	xxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxx	999999	999999
:	:	:	:	:	:
:	:	:	:	:	:
:	:	:	:	:	:
:	:	:	:	:	:
				PAGE TOTAL :	99999999
				GRAND TOTAL :	99999999

Choice 3: If the user enters 'Q', then the program will terminate.

- Day 4: Lab 1.6: Print Item Class wise Report [Duration 1.5 hours]**

This program takes the **itemseq.dat** created in **COBASS03**, and prints a control break report on the itemclass (1st two bytes of the itemno field) as per the format given below.

Item's Classes are as follows:

10 – Internal Components, 20 - Input Devices, 30 - Cards Boards, 40 – Storage Devices, 50 – Output Devices, 60 – Cables, 70 – Storage Media

For each item class, print the total closing balance and closing value. At the end of the report, print the grand totals of the closing value.

ABC CO. LTD.					Date : dd/mm/yyyy		
Item Class wise Report					Page : 999.		
<----- Month Details ----->							
Itemno	Item Description	Op.Bal	Receipts	Dispatch	Clo-Bal.	Rate	Clo. Value
Item Class: xxxxxxxxxxxxxxxxxxxxxx							

xxxxxx	xxxxxxxxxxxxxxxxxxxxxx	999999	999999	999999	999999	999.99	9999999.99
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:

Item Class Total :						9999999	99999999.99

Item Class: xxxxxxxxxxxxxxxxxxxx							
xxxxxx	xxxxxxxxxxxxxxxxxxxxxx	999999	999999	999999	999.99	9999999.99	
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:

Item Class Total :						9999999	99999999.99

Grand Total :							99999999.99

Day 4: Lab 1.7 Analysis and Debugging[Duration 1 hour]

- **TO DO: The participants have to analyze the program, remove the error and successfully execute the program.**
- **TO DO: The program has to be debugged by the participants till it becomes error-free.**

TO DO: The program should be able to successfully calculate vendor total for every vendor.

TO DO: All the participants have to document the error in excel sheet along with steps taken to resolve the error.

Files for participants: - ReportHandlingProgram and Vendor.Dat File

Day4::Lab 1.8 Report-ControlBreak

Goals	<ul style="list-style-type: none">• The program uses the logic of control break processing for generating the report city wise.
Time	1/2 hour

This program generates the electricity bill city wise for the customer file. SRT_ELEC_REC.DAT sorted file.

TO DO:

The participants have to modify the code according to the changes mentioned in the comments.

Files for participants: - ReportHandlingControlBreak.CBL

Day 5:Lab 1.9: Indexed File from Sequential File

Goals	<ul style="list-style-type: none">• Create an Indexed file from an Items sequential file.
Time	1.5 hour

Note: Take the format for “**itemseq.dat**” and create the “**itemmast.dat**” in the same format, except that the field name prefix “IT-” should be replaced with “ITM-”.

Read the sequential file “**itemseq.dat**” and create an indexed file “**itemmast.dat**” with the same format as the input file, and having index keys as:

Itemno as the record Key.

The program should take care of erroneous transactions like, invalid itemno and duplicates or junk transactions (Blank records), and so on.

Note that Month Receipts and Month Despatches are to be initialized to zero before creation.

At the end of program it should display the following:

- “Transactions Read =”,
- “Transactions Created =” and
- “Transactions Rejected =”

Day 5: Lab 1.10: Create an Indexed File from Sequential File

Goals	<ul style="list-style-type: none">• Create an Indexed file from the customer sequential file.
Time	1.5 hour

T

Note: Take the format for “**custseq.dat**” from program **COBASS02** and create the “**custmast.dat**” in the same format, except that the field name prefix “CU-” should be replaced with “CUM-”.

Read the sequential file “**custseq.dat**” and create an indexed file “custmast.dat” with the same format as the input file, and having index keys as:

Custno as the record Key.

The program should take care of erroneous transaction like, invalid custno and duplicates or junk transactions (Blank records) etc.

Note that Current balance is to be initialized to zero before creation.

At the end of program it should display the following:

- “Transactions Read =”,
- “Transactions Created =” and
- “Transactions Rejected =”

Day 5:1.11: Indexed File –Append & Analysis

Goals	• Appending Records to exsisting File.
Time	1.5 hours

The programs append records to the existing file IN_ELEC_REC.DAT (This file must be given to the participant).

The participants have to use existing IN_ELEC_REC.DAT as the input file. The file has 2 records with key values IN-CUST-ID as 2000, 2001.

The participants have to add a record in the input file having IN-CUST-ID Value less than the existing record keys (less the 2000). The record should be written in the file successfully.

The participants have to analyze the program, remove the error and successfully execute the program.

All the participants have to document the error in the excel sheet along with steps taken to resolve the error.

Day 6: Lab 1.12: Updating Indexed File

Goals	<ul style="list-style-type: none"> Update Index file "itemmast.dat" from a sequential file "GRN.DAT".
Time	2.5 hours

A file called "Goods Receipt Note" has data of the days receipts of various items which have been manufactured. The structure of the *Sequential* GRN file is as shown below:

GR-GRNNO	Grn No.	fpic x(5)
GR-DATE	Date	pic 9(8) yyyyymmdd
GR-ITEMNO	Itemno	pic x(6)
GR-QTYMFD	Qty. Mfd	pic 9(5)

Sample data:

GR-GRNNO	GR-DATE	GR-ITEMNO	GR-QTYMFD
GR001	20040402	600002	100
GR002	20040408	200002	30
GR003	20040408	100005	80
GR004	20040506	400001	75
GR005	20040520	200002	29
GR005	20040520	100005	120
GR006	20040520	100009	100
GR006	20040520	600001	70

Logic steps:

Step 1: Take processing dd mm yyyy as input (for example: 20040403 or 20040408).

Step 2: Check if **grn.dat** contains data for the date input. If not, display suitable message and stop run, otherwise continue to step 3.

Step 3: Read a record from the GRN, fetch the corresponding item record from **itemmast.dat** using **itemno** as the key.

- If the record is not found, then display a suitable message as "Item not found in the item master.. Skipping" + itemno.
- If found then add the qty. manufactured to receipts qty.

Step 5: Read the next GRN record, and repeat steps 2 through 5 until end of file of GRN data.

Day 6: Lab 1.13:Order booking for a Customer

Goals	• Order booking by Marketing for a customer.
Time	3.5 hours

Layout of “ORDERS.DAT”:

ORD-ORDNO	Order Number	x(5)	Record key
ORD-DATE	Order date	9(8)	yyyymmdd
ORD-CUSTNO	Order Customer no.	x(6)	
ORD-TOTVALUE	Order Total Value	9(7)v99	
ORD-TX1-PER	Tax Percentage – Tax 1	99v99	
ORD-TX2-PER	Tax Percentage – Tax 2	99v99	
ORD-TX3-PER	Tax Percentage – Tax 3	99v99	
ORD-TOTSUPVAL	Order Total Supplied Value	9(7)v99	
ORD-CUSTREFNO	Customer Order Ref. No.	x(10)	
ORD-STATUS-TAG	Order Status Tag	9	0 – New, 1-Partly Supplied 2 – Fully Supplied.

Layout of “ORDITEM.DAT”:

OIT-ORDNO	Order No.	x(5)	OIT-KEY
OIT-ITEMNO	Item Number	x(6).	Record Key
OIT-ITEMNO1	Item Number	x(6).	OIT-ALTKEY-1 Alternate Key with duplicates
OIT-DATE	Order date	9(8)	yyyymmdd
OIT-QTY	Item Order Quantity	9(6)	
OIT-UOQ	Item UOQ	X(3)	
OIT-RATE	Item Negotiated Rate	9(5)v99	
OIT-QTYSUPP	Item Qty. Supp	9(6)	
OIT-AMTBILL	Item Value Billed	9(7)v99	
OIT-STATUS-TAG	Order-Item Status Tag	9	0 – New, 1-Partly Supplied 2 – Fully Supplied.

The program should take input from the keyboard of a customer's order and create the order on an index file called "**ORDERS.DAT**" with order header information. The order status tag should be set to '0'.

Multiple items may be ordered against a single order. The same item may not be ordered multiple times against the same order. The details of the items of the order are written in to the order child table called "**ORDITEM.DAT**".

Sample Data:

ORD-ORDNO	ORD-CUSTNO
OR001	DA7201
OR002	AB0034

OIT-ORDNO	OIT-ITEMNO	OIT-QTY	OIT-RATE
OR001	100005	100	800
OR001	200002	2	375
OR002	100005	60	875
OR002	400001	25	7900
OR002	600001	40	40

Steps for accepting a new order are as follows:

Step 1: Accept the **ORD-ORDNO** and **ORD-CUSTNO** from the user. Move the **system date** to **ORD-DATE**.

Step 2: Repeat the following steps for each item of a given order:

- Move **ORD-ORDNO** to **OIT-ORDNO**, and **ORD-DATE** to **OIT-DATE**.
- Accept the **OIT-ITEMNO** from the user, and check if it exists in the **ITEMFILE**.
- If it is a valid item (exists in the **ITEMFILE**), then accept the **OIT-QTY**, **OIT-RATE**, **OIT-UOQ** (IGATE, Doz) from the user.
- Move 0 to **OIT-STATUS-TAG**.
- Write the record.
- If the record is successfully written, then compute the item value ($OIT-QTY * OIT-RATE$).
- Add the item value to **ORD-TOTVALUE**.

Step 3: After all the items of an order are entered, move 0 to **ORD-STATUS-TAG**, and write the record to **ORDERS.DAT**.

The user is prompted to enter the order number.

- If the order number is valid, then the user is asked to enter the item no.
- If the record is found, then the user should be allowed to modify the order quantity if the order status is '0' viz 'New'.
- If the order number is invalid, then an error message should be displayed.

The user is prompted to enter the order number.

- If the order number is valid, then the order details are displayed.
- If the order number is invalid, then an error message should be displayed.

Note: This can also be done using one file with two formats, that is two level 01 record description having some common fields and rest depending on the data – **one record** describing the contents of **orders.dat** and the **other record** describing the contents of **orditem.dat**.

Day 6: Lab 1.14 Enhancement Assignment

Goals	• Adding new Functionality to exsisting Lab
Time	1 hour

Add new functionality in the program i.e. 'Modify an existing order'
It should be the second option in the menu.
Accordingly modify the menu

When the user choose this option:

The user is prompted to enter the order number. If the order number is valid, the user is asked to enter the item no. If the record is found the user should be allowed to modify the order quantity if the order status is 'O' viz 'New'. If the order number is invalid, an error message should be displayed.

Files for participants: - Enhancement.CBL, ORDERS.DAT ,ORDITEM.DAT

Day 7:Lab 1.15 Analysis and Debugging

Goals	<ul style="list-style-type: none">This program takes Current-date and Bill-month as input from user and generates Telephone bill for that month for every customer included in an input file .
Time	1.5 hours

TO DO: The participants have to analyze the program, remove the error and successfully execute the program.

TO DO:All participants have to document an error in the excel sheet along with steps taken to resolve the error.

Files for participants: - - CurrentDateBillMonth.CBL and CUSTREC.DAT
Sample Report:- SAMPLE_TELBILL.REP

Day 7:Lab 1.16 Bank Transaction Updation and Report

Goals	A program to read Account Transaction file, Generate a Report & Update the Account Master file.
Time	8 hours

File Structure
1.1 Transaction File Record Structure

Transaction file is a sequential file of LRECL 80.

Valid Account Types are

- 'CC' – Credit Card
- 'SB' – Savings Acct
- 'CR' – Current Acct

Valid Payment Modes are

- 'CS' – Cash
- 'CQ' – Cheque

```

01 ACCOUNT-TRANS.
05 AT-ACCT-NO      PIC S9(09) COMP-3.
05 AT-ACCT-TYPE    PIC X(02).
05 AT-DUE-DATE.
  10 AT-DUE-YYYY    PIC 9(04).
  10 FILLER          PIC X.
  10 AT-DUE-MM       PIC 9(02).
  10 FILLER          PIC X.
  10 AT-DUE-DD       PIC 9(02).
05 AT-TRANS-DATE.
  10 AT-TRANS-YYYY   PIC 9(04).
  10 FILLER          PIC X.
  10 AT-TRANS-MM     PIC 9(02).
  10 FILLER          PIC X.
  10 AT-TRANS-DD     PIC 9(02).
05 AT-DUE-AMT      PIC S9(09) COMP-3.
05 AT-AMT-CR       PIC S9(09) COMP-3.
  
```

```
05 AT-AMT-DB      PIC S9(09) COMP-3.  
05 AT-PAY-MODE    PIC X(02).  
05 FILLER         PIC X(36).
```

1.2 Master File Record Structure

Master file is an Indexed file of LRECL 100 with AM-ACCT-NO as the Key.

Valid Account Status are

‘ACT’ – Active Account

‘BLK’ – Blocked Account

```
01 ACCOUNT-MAST.  
05 AM-ACCT-NO      PIC S9(09) COMP-3.  
05 AM-ACCT-TYPE    PIC X(02).  
05 AM-ACCT-STATUS  PIC X(03).  
05 AM-NAME.  
    10 AM-FNAME     PIC X(15).  
    10 AM-LNAME     PIC X(15).  
05 AM-DOB          PIC X(10).  
05 AM-AVAIL-BAL    PIC S9(09) COMP-3.  
05 AM-CONTACT-NO   PIC 9(10).  
05 AM-ADDRESS.  
    10 AM-ADDR1     PIC X(10).  
    10 AM-ADDR2     PIC X(10).  
05 FILLER          PIC X(15).
```

1.3 Report REPRT of LRECL 160.

2. Program Specification:

1. Main Program: DREPPGM

Read transaction file...

Step 1: For ‘CC’ account type

1a. if the due amt ≤ 0 , read next record.

1b. if the due amt > 0 & due date is prior to current date & credited amt $<$ due amt, then:

- Add credited amt to available balance, change acct status to 'BLK' & update master file.
- Generate a report for the blocked account.

1c. if the due amt > 0 & due date is prior to current date & credited amt \geq due amt, then:

- Add credited amt to available balance, change acct status to 'ACT' & update master file.
- Generate a report.

1d. if the due amt > 0 & due date is current date & credited amt $<$ due amt, then:

- Add credited amt to available balance & update master file.
- Generate a report.

1e. if the due amt > 0 & due date is current date & credited amt \geq due amt, then:

- Add credited amt to available balance & update master file.

1f. if the due amt > 0 & due date is greater than current date & credited amt $<$ due amt, then:

 If due date is within the next 2 days from current date then:

- Add credited amt to available balance & update master file.
- Generate a report.

 Else

- Read next record.

1g. if the due amt > 0 & due date is greater than current date & credited amt \geq due amt, then:

- Add credited amt to available balance & update master file.

Step2: For 'SB' or 'CR' account type

2a. if credited amt > 0 & debited amt = 0, then:

- Add credited amt to available balance & update master file.
- Generate a report.

2b. if credited amt = 0 & debited amt > 0, then:

- Subtract debited amt from available balance & update master file.
- Generate a report.

2c. if credited amt > 0 & debited amt > 0, then:

- Add credited amt & subtract debited amt from available balance & update master file.
- Generate a report.

2. Sub-program: VALIDATE

Accept Due Date from Driver Pgm (DREPPGM)

Step 1. Date Validation.

Step 2. Leap Year Validation.

Step 3. Validate to check whether due-date falls within the coming 2 days from the current date.

Day 8: Lab 1.17 Working with 3 Indexed Files

Goals	<ul style="list-style-type: none"> Retrieving Data from 3 indexed files
Time	6 hours

iLearn is required to generate a Purchase Report two months before the beginning of each semester to its MS program. This report lists the books that have to be purchased for the coming semester. Trainers/lecturers requirements of books are stored in a Requirements file. This file contains details of the lecturers' book requirements for semesters 1 and 2. There are two other files: Publisher and Book files, which contain publisher and book details respectively.

The participants have to analyze the program and document the Logic of the program.

Wherever require, participants have to add comments to increase the understandability of the program.

Requirements File [Indexed]

There is a record for each book title required by a lecturer.

Note: A book may be required by more than one lecturer.

File Structure

Field	Key Type	Type	Length	Value
PR-Number	Primary	9	4	1-9999
Trainer	Alt with duplicates	X	20	--
Book-id	Alt with duplicates	9	4	1-9999
Copies-required	--	9	3	1-9999
Semester	--	9	1	1/2

Sample Data for the Requirement File

0001	GERARD	0012	025	1
0002	HUTCHISON	0112	150	1
0003	KINSELLS	1111	050	1

Book File [Indexed]
File Structure

Field	Key Type	Type	Length	Value
Book-id	Primary	9	4	1-9999
Publisher-Number	Alt with duplicates	9	4	1-9999
Title	--	X	30	--

Sample Data for the Book File

0012	1111	**NO MATCHING**
0112	2222	FRENCH

1111	2222	MATHS
------	------	-------

Publisher File [Indexed]

File Structure

Field	Key Type	Type	Length	Value
Publisher-Number	Primary	9	4	1-999
Publisher-Name	Alt with duplicates	X	20	--
Publisher-Address	--	X	40	--

Sample Data for the Publisher File

1111	QUE	ENGLAND
2222	PRENTICE	USA
3333	MICROSOFT	USA
4444	MCGRAW-HILL	INDIA

Note :

Accept the semester from the user.

Generate the requirements report that shows book requirements for the semester entered by the user.

The report should be sequenced on ascending publisher name and should only list purchase requirements for the semester asked by the user.

Each publisher should appear on a new page.

Assumptions: One book may be required by multiple trainers.

Files for participants: - PurchaseReport.CBL and all input files

Day 8: & Day 9: Lab 1.18 Online Bus Booking

Goals	.Program to read Booking Trans file, Check the availability of the seat in the Bus Detail file, Generate Ticket no., PNR no., Update the Booking Master file, Update Bus Detail file, Print Tickets & Print canceled report in case of cancellation.
Time	8 hours

1. File Structure
1.1 Booking Transaction File Record Structure

Booking Transaction file is a sequential file of LRECL 120.

```

01 BOOK-TRANS-REC.
  05 BT-PNR-NO      PIC X(15) VALUE SPACES.
  05 BT-PASSENGER-NAME.
    10 BT-FNAME     PIC X(15) VALUE SPACES.
    10 BT-LNAME     PIC X(15) VALUE SPACES.
  05 BT-GENDER      PIC X(01) VALUE SPACES.
  05 BT-CONTACT-NO  PIC 9(10) VALUE ZEROES.
  05 BT-AGENT-NO    PIC 9(05) VALUE ZEROES.
  05 BT-BUS-TYPE     PIC X(03) VALUE SPACES.
  05 BT-FROM-LOC    PIC X(10) VALUE SPACES.
  05 BT-TO-LOC      PIC X(10) VALUE SPACES.
  05 BT-SEAT-NO     PIC X(02) VALUE SPACES.
  05 BT-TRAVEL-DATE PIC X(08) VALUE SPACES.
  05 BT-TRAVEL-TIME.
    10 BT-TRAVEL-TIME-HH PIC 9(02) VALUE ZEROES.
    10 BT-TRAVEL-TIME-MM PIC 9(02) VALUE ZEROES.
  05 FILLER         PIC X(22) VALUE SPACES.
  
```

1.2 Master File Record Structure

Booking Master file is an Indexed file of LRECL 180 with BM-PNR-NO as the Key.

```

01 BOOK-MAST-REC.
  05 BM-PNR-NO      PIC X(15) VALUE SPACES.
  05 BM-TICKET-NO   PIC X(12) VALUE SPACES.
  05 BM-STATUS      PIC X(05) VALUE SPACES.
  05 BM-BUS-NO      PIC X(10) VALUE SPACES.
  05 BM-BUS-NAME    PIC X(10) VALUE SPACES.
  05 BM-AGENT-NO    PIC 9(05) VALUE ZEROES.
  05 BM-SEAT-NO     PIC X(02) VALUE SPACES.
  05 BM-FROM-LOC    PIC X(10) VALUE SPACES.
  05 BM-TO-LOC      PIC X(10) VALUE SPACES.
  05 BM-BOOK-DATE   PIC X(08) VALUE SPACES.
  05 BM-CANCL-DATE  PIC X(08) VALUE SPACES.
  05 BM-TRAVEL-DATE.
    10 BM-TRAVEL-DD  PIC X(02) VALUE SPACES.
    10 BM-TRAVEL-MM  PIC X(02) VALUE SPACES.
    10 BM-TRAVEL-YYYY PIC X(04) VALUE SPACES.
  05 BM-TRAVEL-TIME  PIC 9(04) VALUE ZEROES.
  05 BM-FARE         PIC ZZZ9(02).99 VALUE ZEROES.
  05 BM-CANCL-AMT    PIC ZZZ9(02).99 VALUE ZEROES.
  05 BM-PASSENGER-NAME.
    10 BM-FNAME      PIC X(15) VALUE SPACES.
    10 BM-LNAME      PIC X(15) VALUE SPACES.
  05 BM-CONTACT-NO   PIC 9(10) VALUE ZEROES.
  05 BM-BUS-TYPE     PIC X(03) VALUE SPACES.
  05 FILLER          PIC X(14) VALUE SPACES.
  
```

1.3 Bus Detail File Record Structure

Bus Detail file is a sequential file of LRECL 160.

```

01 BUS-DETAIL-REC.
  05 BD-BUS-NO      PIC X(10) VALUE SPACES.
  05 BD-BUS-NAME    PIC X(10) VALUE SPACES.
  05 BD-BUS-TYPE    PIC X(03) VALUE SPACES.
  05 BD-AVAIL-SEATS  PIC X(100) VALUE SPACES.
  05 BD-FROM-LOC    PIC X(10) VALUE SPACES.
  05 BD-TO-LOC      PIC X(10) VALUE SPACES.
  05 BD-FARE        PIC ZZZ9(02).99 VALUE ZEROES.
  05 FILLER         PIC X(09).
  
```

1.4 Report PRNTTKT of LRECL 122.

1.5 Report RFNDTKT of LRECL 122.

2. Program Specification:

1. Program: BUSBOOK

Read booking transaction file...

Step 1: For new bookings

1a. Read bus detail file & Check the availability of Bus & Seat no.

1b. If the preferred bus type & preferred seat no is available for the said date of route.

- Book the ticket (Generate Ticket #, PNR #).
- Update Booking History file.
- Update Bus Detail file
- Print Ticket.

1c. If the preferred bus type & preferred seat no is not available for the said date of route. Book any other available seat no.

- Book the ticket (Generate Ticket #, PNR #).
- Update Booking History file.
- Update Bus Detail file
- Print Ticket.

Step2: For Cancellation

2a. If the cancellation date is less than 24 hrs from the date of journey.

- Print cancellation report without any refund amount.

2b. If the cancellation date is more than 24 hrs from the date of journey. Deduct 15% of the fare

Towards cancellation charges.

- Print cancellation report with the refund amount.

Day 9:Lab Assignment 1.19:Call Statement-Debugging

Goals	• Call Statement
Time	15 minutes

.If program A is called with a call is A using X,Y.Refer to below code
Linkage-Section.

05 M
05 P

Procedure Division using X,Y

----- EXIT.

Debug the program and rectify the error

Day 9:Lab Assignment 1.20:Call Statement-Enhancement Assignment

Goals	• Calling Sub programs
Time	2 hours.

- Write a separate program which will calculate the tax for the employee and try to call the same program in the EMP file created.
- Write a COBOL program (using a subroutine to calculate the Gross) for an employee of a Company whose income details (Basic Pay, HRA, PF, DA) are given in the file INCOME.DAT, create the gross pay in OUTPUT.DAT.
 - INPUT.DAT
 - Emp1 12000 4000 2000 7000
 - Emp2 12500 3750 2300 6450
 - Emp3 12340 3750 2345 5430
 - Emp4 13450 3540 2340 6000
 - Emp5 12000 4000 2300 6500
 - OUTPUT.DAT should contain the employee name and gross pay.
- Redo assignment 3 using the CALL statement. The user should be able to select from the menu. The selected menu option should be available as a separate program and called from the main program.

Day 10: Lab 1.21: Table Handling Lab Assignment

Goals	• Working with Arrays (Table Handling)
Time	2 hours

1) Compute the weekly pay for each employee, the rate of pay as found in table TABPAY is multiplied by the number of hours worked (a three-position field with one decimal position). The weekly pay amount is a five-position field with two decimals after rounding.

TABLE

	TABEMP	TABPAY
1001		5.25
1002		6.25
	1003	5.40
1004		4.50
1005		5.50
1006		4.75
1007		6.50
1008		4.50
1009		5.00
1010		8.00

Input	Positions	Field
	1 - 4	Employee number
	35 - 37	Hours XX.X

Calculations.

Perform a search of the table using employee number as the search word. After a successful search, perform the necessary calculations.

Output

Leave five positions between each field. Output the employee number, hours worked, rate off pay, and gross pay.

Output is as follows.

EMPLOYEE NUMBER	HOURS WORKED	RATE OF PAY	GROSS PAY
1001	15.5	5.25	81.38
1002	31.5	4.50	146.25
1003	40.0	5.50	220.00
1004	5.0	6.50	31.50
1010	25.0	8.00	200.00

2) A table stores Loan Account Number and Loan Outstanding amount (In lakhs). Use SEARCH to find the number of people who have an outstanding amount of
 Over 10 lakhs
 Between 5 – 10 lakhs (Both inclusive).

Less than 5 lakhs

Day 10: Lab Assignment 1.22:Table Handling- Analysis Assignment

Goals	<ul style="list-style-type: none">The referenced programs have to be analyzed and have to made error free by the participants.
Time	1.5 Hours

Define a 20 items of static Data of Country code and Country Name in the program. Write a program to find out Country Code based on Country Name. The program must use the SEARCH and Arrays statements.

Static Data of the Country Code and Country Name

```
00001USA
00002United Kingdom
00003France
00004China
00005Japan
00006Italy
00007Russia
00008India
00009Nepal
```

Files for participants: - Countrycode.Dat file, Lab22.cbl.Perform Code Review using Code Review excel.

Day 10: Lab 1.23: String Handling

Goals	<ul style="list-style-type: none"> Implementing String Handling
Time	1.5 Hours

StreetNo	pic 9(3)
StreetName	pic x(15)
City	pic x(10)
Pincode	pic 9(6)

Print the output in the following format:

StreetNo, StreetName

City – Pincode

- String four variables A1, B1, C1 and D1 whose values are A1 = BHARAT, B1 = SANCHAR, C1 = NIGAM and D1 = LIMITED into a variable RESULT.
- UNSTRING the RESULT variable of previous program and store the values in 4 different variables.

Day 11:Lab 1.24: Mobile Services

Goals	<ul style="list-style-type: none"> To develop mobile services which implements all the concepts covered in Cobol.
Time	7 Hours

- Design a Sequential file which will give the details of Customer like CustomerCode, Customer Mobile Number, Customer Name, Address, City, Country, Pincode, Email, Planopted, ServiceType
- Sort the file based on CustomerCode.
- Include Input procedure by selecting data for specific plan (say Plan 99)
- Create an Indexed File for Payment options which includes fields like Payment Date, Mobile No, CustomerCode, Mode of Payment, DueDate, PaymentDate, AmountPaid, BillAmount, Balance amount, paymentoption(direct debit, cash to mobile, vodaphone shop, telebanking, ATM. For e.g for a specific customer if the amountpaid is 200 Bill amount 300 balance has to be 100.
- Include one field of Late Payment charges with following criteria

	Outstanding amount	Late payment Charges
1	Less than Rs. 100/-	Nil
2	>=Rs. 100/- but less than Rs. 300/-	Rs. 25/-
3	>= Rs. 300/- but less than Rs. 500/-	Rs. 50/-
4	>= Rs. 500/-	Rs. 100/-

- Include 5 records which has done late payment for bill and which fulfill the above criteria
- Vodafone has decided instead of country code MUM they will use only M. Implement that change in Sequential file.
- Create an Indexed file of Services which include Customercode and ServiceType from Customer file. This file will give us the information of Services whether opted for prepaid or postpaid. It may include fields like type of service(prepaid, postpaid) rates for STD, rates of ISD, rates of SMS, incoming calls, outgoing calls, roaming charges.
- Create one indexed file called CallsMade which include call details like calls made from, to, duration, cost(cost will be calculated based on the service they have opted and the std rates, isd rates or roaming charges as applicable.
- Generate the following reports
 - To display details of all customers who reside in Mumbai
 - To display details of all customers who have outstanding amount of 1000
 - To display only those customers who have opted for prepaid service
 - To display the number of calls made for a specific month by a specific customer in short all calls made for a specific month.

Analysis	<p>Application Functionality</p> <p>Provide a Summary Comments in the module providing the application overview</p>
Enhancement	<p>Add new features</p> <p>To display all customers who have paid the bills through ATM and who are from India</p>
Problem Solving	<p>Solve current bugs</p> <p>The total bill amount for the customer is invalid based on the published rates.</p> <p>As shown below:</p> <p>Pre-paid Local Call: 1.4 STD rates : 1.3 ISD rates: 3.1</p> <p>Post-paid Local Call: 1.5 STD rates: 1.5 ISD rates: 3.5</p> <p>For a customer the bill should have been X but it is Y Reason: Switching of the Pre Post rates in the files</p>
Debugging	<p>Test Runs</p> <p>Use debugger for the above problem.</p>

- **Stretched Assignment 1:**
- **Itemwise Sales Order Report**

This program prints the Item wise sales orders report for a given period. The report takes data from **orditem.dat** and prints in the format given below.

ABC CO. LTD.				Date : dd/mm/yyyy		
Items wise Sales From dd/mm/yyyy to dd/mm/yyyy				Page : 999.		
Itemno	Item Description	Date	UOQ	Ord-Qty.	Rate	Order-Value
xxxxxx	xxxxxxxxxxxxxxxxxxxxxx	dd/mm/yyyy	xxx	999999	999.99	9999999.99
		dd/mm/yyyy	xxx	999999	999.99	9999999.99
		dd/mm/yyyy	xxx	999999	999.99	9999999.99
		:	:	:	:	:
		:	:	:	:	:
Item Totals :			9999999		99,999,999.99	
xxxxxx	xxxxxxxxxxxxxxxxxxxxxx	dd/mm/yyyy	xxx	999999	999.99	9999999.99
		dd/mm/yyyy	xxx	999999	999.99	9999999.99
		dd/mm/yyyy	xxx	999999	999.99	9999999.99
		:	:	:	:	:
		:	:	:	:	:
		:	:	:	:	:
Item Totals :			9999999		99,999,999.99	
Grand Total :						99,999,999.99

The program asks for input of start date and end date. It should do a date check of the two dates.

It then starts the **orditem.dat** file to the start date using the alternate keys oit-altkey-1 (Itemno + Date). It should also handle error in case of error in start.

The program should then access the data sequentially. Verify that the data is for the dates in the range, if not then skip the records.

For a valid record, if it is the 1st record of the item, then print the **ItemNo** and the **Item description** (which you get from "itemmast.dat"). Otherwise, for other records, print the details without the **Itemno** and **description**.

Take totals of the order-value and quantity. At the change of item, print the Order total as per the report on top.

At the end of the report, print the last order total and the grand total of the value only.

The reading of the file continues until End of File.

- **Stretched Assignment 2: Write a program to print the following report using the file created in program COBASS2.**

 ABC CO. LTD.

Date : dd/mm/yyyy

Region wise report

Region : XXX

Customer Name	Address
XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXX
	XXXXXXXXXXXXXXXXXX
:	:

Total customers in XXX : 999

Region : XXX

Customer Name	Address
XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXX
	XXXXXXXXXXXXXXXXXX
:	:

Total customers in XXX : 999

Total customers : 9999

- **Case Study**

1. **Write a program to input the data given below.**

Program-Id : STUASS01

ST-ROLLNO	Student rollno	pic 9(4)
ST-NAME	Student name	pic x(15)
ST-CLASS	Student class	pic x(3)
		First 2 digits Std
		3 rd character division.
ST-SUB1-MARKS	Marks of subject 1	pic 9(3).
ST-SUB2-MARKS	Marks of subject 2	pic 9(3).
ST-SUB3-MARKS	Marks of subject 3	pic 9(3).
ST-FEES-PAID	Fees paid	pic 9(4)

Logic :

- The data should be of a record structure as above.
- Std can have values in the range 1-10. Division can have values 'A' or 'B'.
- Marks should be in the range 0-100.
- Input the details for the above record. Once the data of a record is complete, the program should display the full group as an item.

- The program should then ask the user “Do you want to Input More (Y/N)”. It should accept the users response.
- If the response is “Y”, the program should accept the details on another item and repeat the above steps .
- If the response in “N”, the program should stop.

1. Write a program to input the data given below.

Program-Id : STUASS02

MNT-ROLLNO		pic 9(4)
MNT-FLAG	UPDATION FLAG	pic x
		A-Add
		M-Modify (only name can be modified)
		D-Delete
MNT-NAME	STUDENT NAME	PIC X(15)

Logic :

- The data should be of a record structure as above.
- MNT-FLAG can have values the values A, M or D.
- Input the details for the above record. Once the data of a record is complete, the program should display the full group as an item.
- The program should then ask the user “Do you want to Input More (Y/N)”. It should accept the users response.
- If the response is “Y”, the program should accept the details on another item and repeat the above steps .
- If the response in “N”, the program should stop.

3. Modify program STUASS01 to have display the grade of each student.

Program-Id: STUASS03

- Grades are awarded on the following basis :

Percentage	Grade
>= 70	DIST
60-70	CRDT
50-60	PASS
<50	FAIL

If a student scores below 40 in any subject, award a FAIL grade.

At the end display the no. of students passed and failed.

4. Modify program STUASS01 to have three choices as listed below. The program should function as per the steps of choices listed below.

Program-Id : STUASS04

- A . Input Data
- B. Print Data
- C. Quit.

Input data :

- Open the file "studseq.dat" in extend mode.
- The data is entered as in STUASS01 but instead of displaying on the screen it is to be written to a sequential file named "studseq.dat" with the same format as the group.
- When the user enters "Y" to "Do you want to enter more (y/n) ?", the program should once again take input and write to file.
- With user input as "N" , the program should close files and go back and display the menu once again.

Print Data

- The program should open the file "studseq.dat" in sequential input mode and the "studseq.dat" in output mode.
- Accept the class from the user & print a report for all students in the class.
- For every valid record read, it should print as per the format given below.
- There should be a max of 5 lines per page.
- The report should be printed in sequence of Rollno.
- At the end of the report, a grand total of the value filed field should be printed and all files closed.

Date : dd/mm/yyyy

Scholars' Academy

Marksheet for Standard : 99 Division :X

Rollno	Student Name	Subject 1	Subject 2	Subject 3	Grade
9999	XXXXXXXXXXXXXXXXXX	999	999	999	XXXX
:	:	:	:	:	:
:	:	:	:	:	:
:	:	:	:	:	:
Total no. of students :					999

Quit :

The program should stop run.

5. Modify program STUASS02 as listed below.
Program-Id : STUASS05

- Open the file "mntseq.dat" in extend mode.

- The data is entered as in STUASS02 but instead of displaying on the screen it is to be written to a sequential file named "mntseq.dat" with the same format as the group.
- When the user enters "Y" to "Do you want to enter more (y/n) ?", the program should once again take input and write to file.
- With user input as "N", the program should close files and go back and display the menu once again.

Print the following standard wise report.

Program-Id : STUASS06

Scholars' Academy
List of students

STD : 99

RollNo Name

9999	XXXXXXXXXXXXXXXXXX
:	:
:	:

Total No. of students in Std 99 : 99

STD : 99

RollNo Name

9999	XXXXXXXXXXXXXXXXXX
:	:
:	:

Total No. of students in Std 99 : 99

:	:	:
:	:	:
:	:	:

Total no. of students in the school: 999

7. Print the following standard wise, division wise report.

Program-Id : STUASS07

Scholars' Academy
Consolidated Marksheet

STD : 99

 Div : A

RollNo	Name	Grade
--------	------	-------

Total No. of students in Div A : 99	-----

 Div : B

RollNo	Name	Grade
9999	XXXXXXXXXXXXXXXXXX	XXXX
:	:	:
:	:	:
:	:	:

Total No. of students in Div B :	999	-----

 Total No. of students in Std 99 : 999

:	:	:
:	:	:
:	:	:
:	:	:

Distinction	: 99
Credit	: 99
Pass	: 99
Fail	: 99

8. Indexed files

Program-Id : STUASS08

Note : Take the format for "studseq.dat" from program STUASS01 and create the "studmast.dat" in the same format, except that the field name prefix "ST-" should be replaced with "STM-".

Read the sequential file "studseq.dat" and create an indexed file "studmast.dat" with the same format as the input file, and having index keys as ;

STM-ROLLNO as the record Key.
 STM-CLASS as the alternate record key.

The program should take care of erroneous transaction like, invalid class and duplicates or junk transactions (Blank records) etc. write the erroneous records to the file studerr.dat . This file will have an additional field

Remark Pic X(15)

The values for remark can be any of the following
Duplicated
Out of sequence
The access mode of the file should be sequential.

At the end of program it should display the following

"Student Details Entered =",
"Student Records Created =" and
"Student Records Rejected ="

9. Modify program STUASS6 (single level control break).

Program-Id : STUASS09

Accept a the CLASS from the user.
Print a report for the given CLASS.

10. Using the files created in program STUASS05 (mntseq.dat) and STUASS08 (studmast.dat).

Program-Id : STUASS10

Read a record from the maintenance file (mntseq.dat).
For each record read, add/modify/delete the record in the master file (studmast.dat). Repeat this till the end of the maintenance file.

11. Using the file created in program STUASS04 (studseq.dat). This program will have 3 options, as mentioned below :

Program-Id : STUASS11

- A. Create transaction
- B. Update Master
- C. Exit

Create transaction

Write a program to create the following file (studtran.dat)

Rollno PIC 9(4)
Fees_paid Pic 9999

Note : Enter some invalid rollnos. i.e. which do not exist in the studseq.dat.
There can be multiple transaction records for a student.

Update Master

Write a program to update the studseq.dat file using the studtran.dat.

11. For each Class(irrespective of division) find the highest scorer in each subject.

Program-Id : STUASS11

Print the details in the following format

Class	Subject 1		Subject 2		Subject 3	
	Max	Name	Max	Name	Max	Name
99	99	XXXXX 99	XXXXX 99	XXXXX		
:	:	:	:	:	:	:
:	:	:	:	:	:	:

Assumption: There is only 1 highest scorer in each class.

13. Write a program to accept a company name and display it in the center of the screen.

Note: The user may enter leading/trailing spaces.

14. Modify the second option (Print Data) of the program STUASS04 to receive the class from the main program and print the report.

15. Accept details from the user in the following format :

```

StreetNo          pic 9(3)
StreetName        pic x(15)
City              pic x(10)
Pincode           pic 9(6)
  
```

Print the output in the following format

StreetNo, StreetName

City - Pincode

-----*****-----

Appendices

- **Appendix A: Breakup of Marks**

Each assignments carries 30 marks.

Breakup up of marks is as follows:

- Flowchart/Logic – 5
- Standards - 5
- Coding – 10
- Error Handling - 5
- Output - 5

- **1. Compiling, Linking, and Executing COBOL programs**

Creating Source Code:

Step 1: Run the **CA-Realia workbench** software. Select **file** → **new** and save the file with extension as **.cbl**. On saving the file as **.cbl** it highlights all the COBOL keywords.

Compiling and Linking:

Step 1: Select the menu option **Options** → **Other** from the **Build** menu, and compile the **Source code**.

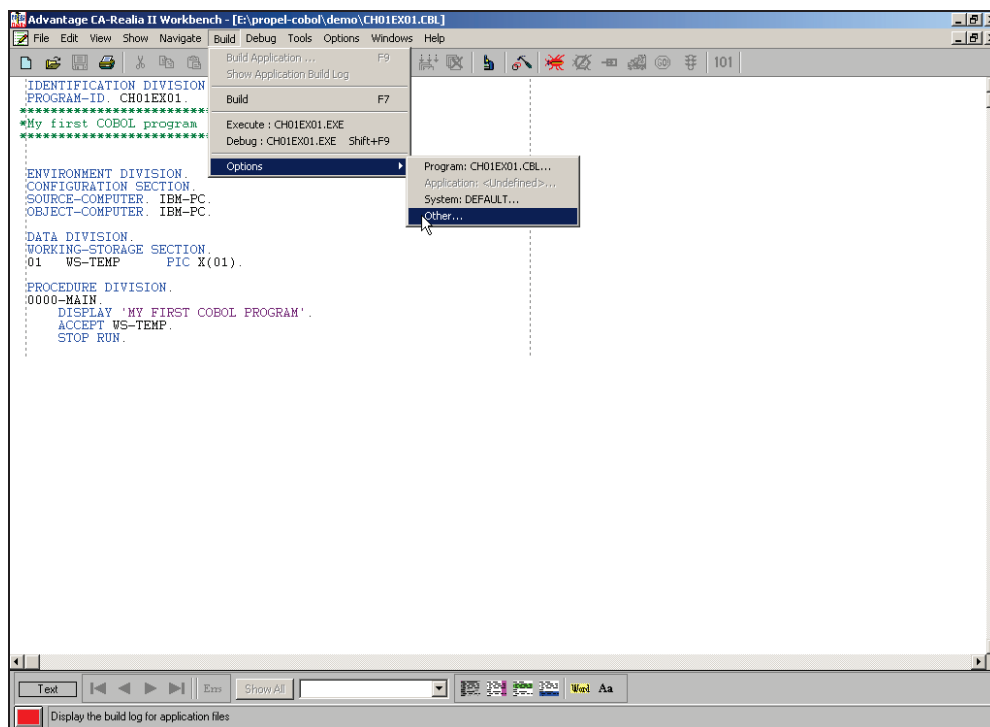


Figure 1: Creating Source Code

Step 2: Select the application type as EXE Application.

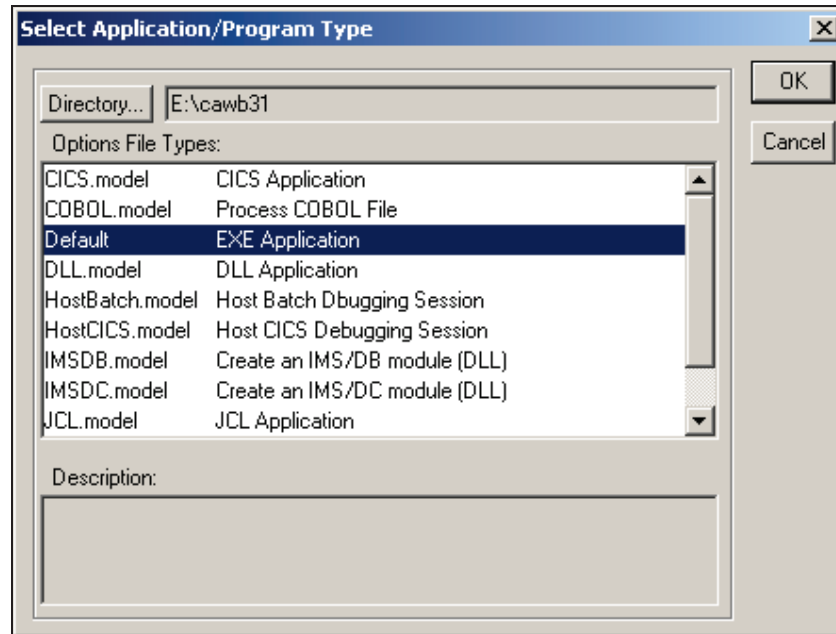


Figure 2: Select Application/Program Type

Step 3: Click **OK**. Select the **Link** tab. Click the **.EXE/LIB** button, and select the path for the file. The exe file will be generated in this directory.

(**Note:** Preferably select the same directory where you have your .cbl file)

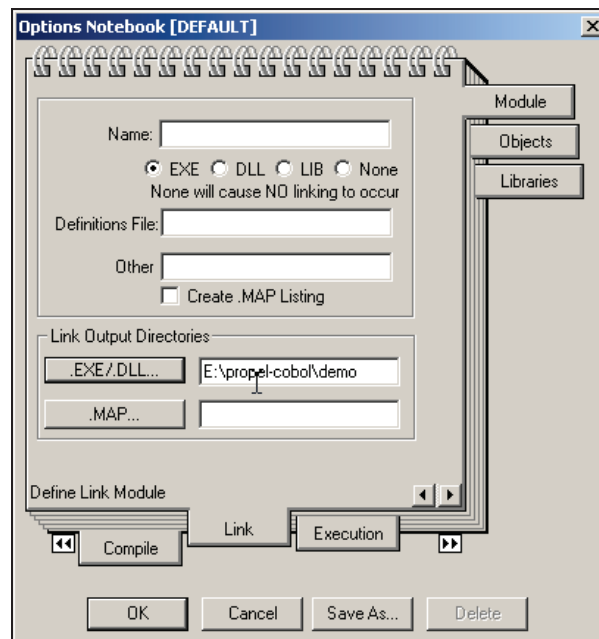


Figure 3: Options Notebook

Step 4: Click **OK**. Select **Build** → **Build**. On successful compilation, the message as shown in the following screenshot is displayed. In this message window, click **No**.

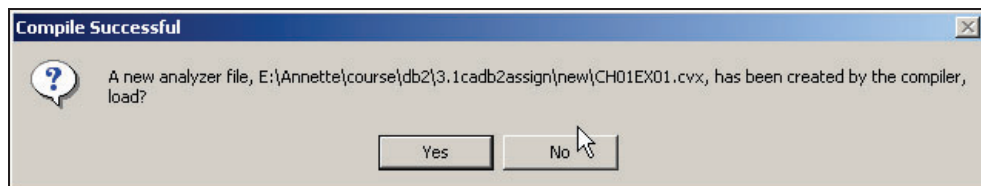


Figure 4: Compile Successful message

Run:

Let us now create the execution profile.

Step 1: Select the following option in **CA-Realia**. Select **Build** → **Options** → **Program** for building the execution profile.

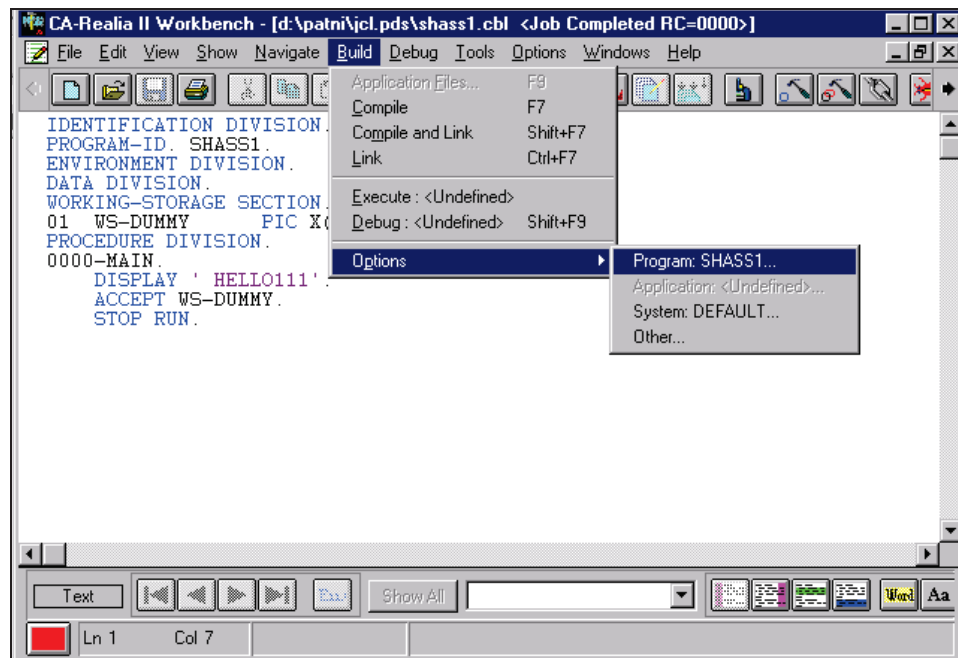


Figure 5: Building execution profile

Step 2: Select the **Execution** tab from **Realia Cobol session** options, and create a new profile.

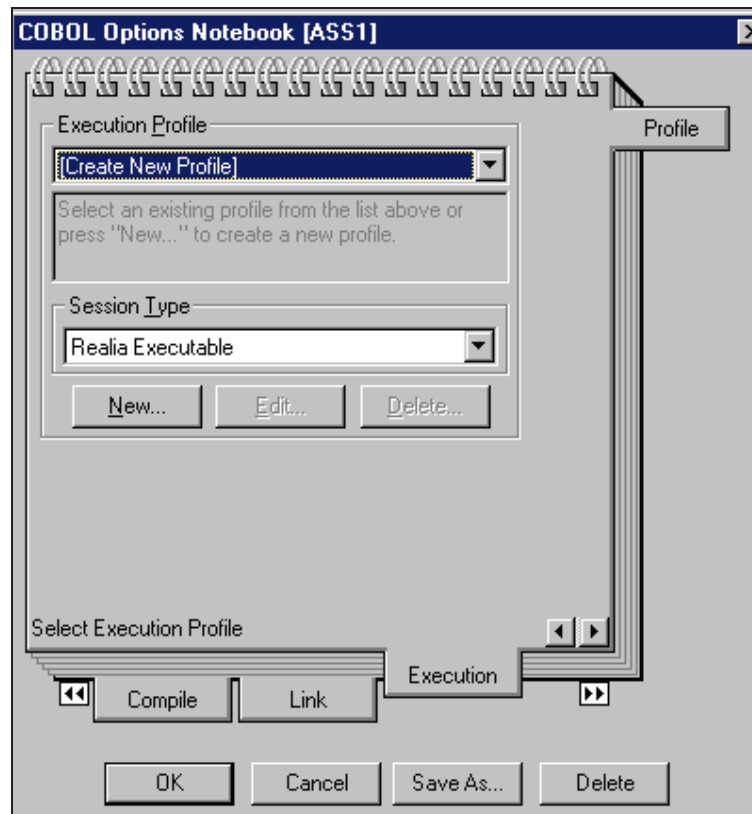


Figure 6: Execution tab

Step 3: Key in the executable filename and path.

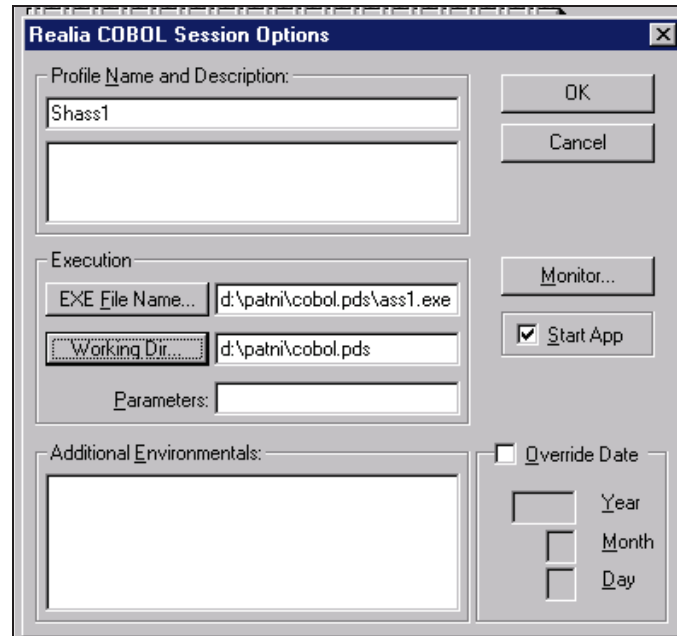


Figure 7: Realia COBOL Session Options

Step 4: After creation of the execution profile, run the .exe by selecting the **Build → Execute** option.

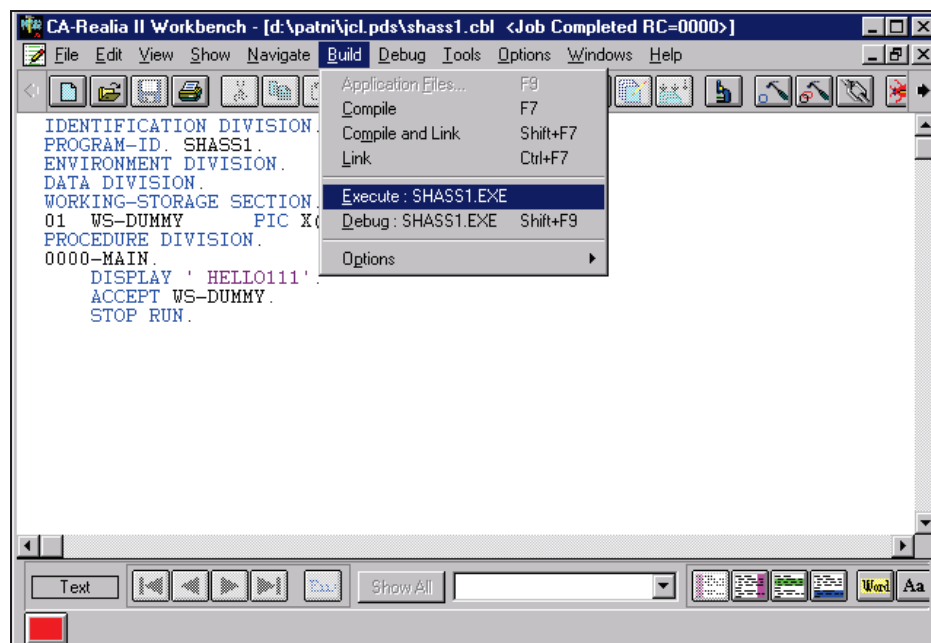


Figure 8: Run .exe

- 1. Debugging a COBOL program

Creating debug profile

After creation of the **Execution Profile**, we can also create a debug profile. This is mainly used for debugging.

Step 1: Select **Debug** option from the **Build** menu.

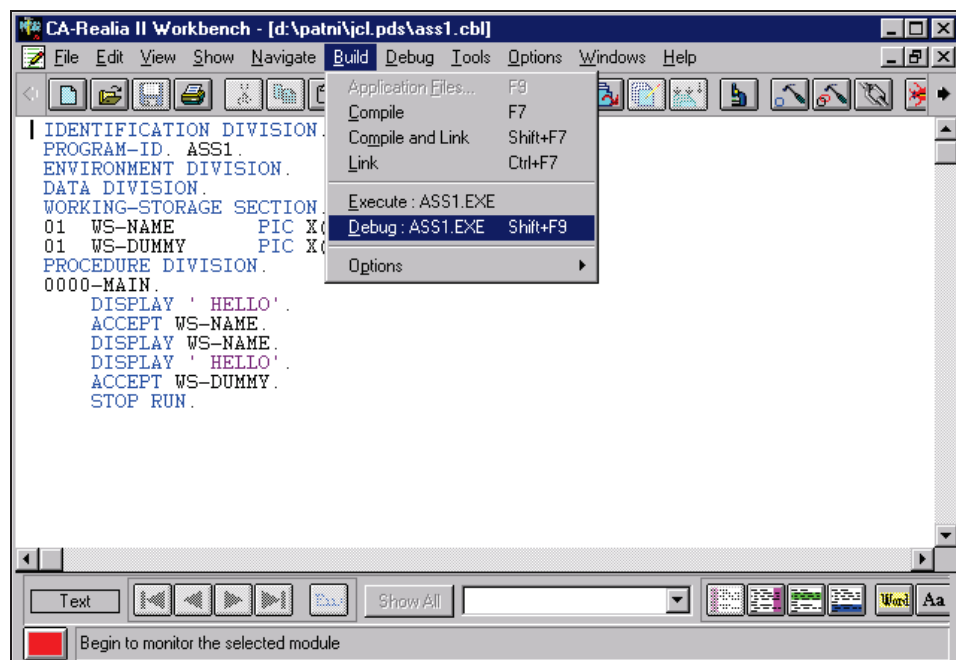


Figure 9: Build menu

Step 2: From the **Execution Options**, select the **Edit** option. From **CA-Realia COBOL session options**, select **Monitor**. Then select the cvx file to be debugged (monitor). The following screen will be displayed.

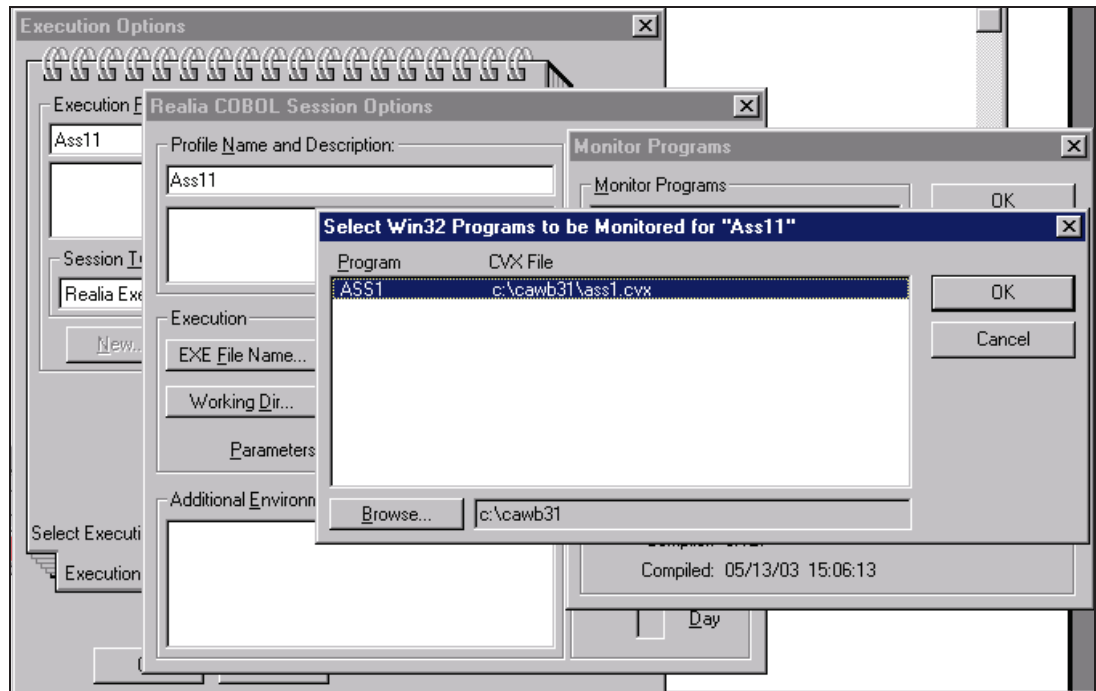


Figure 10: Execution Options

Step 3: Click **OK**. The following screen will be displayed. We can start debugging from the first statement from the procedure division.

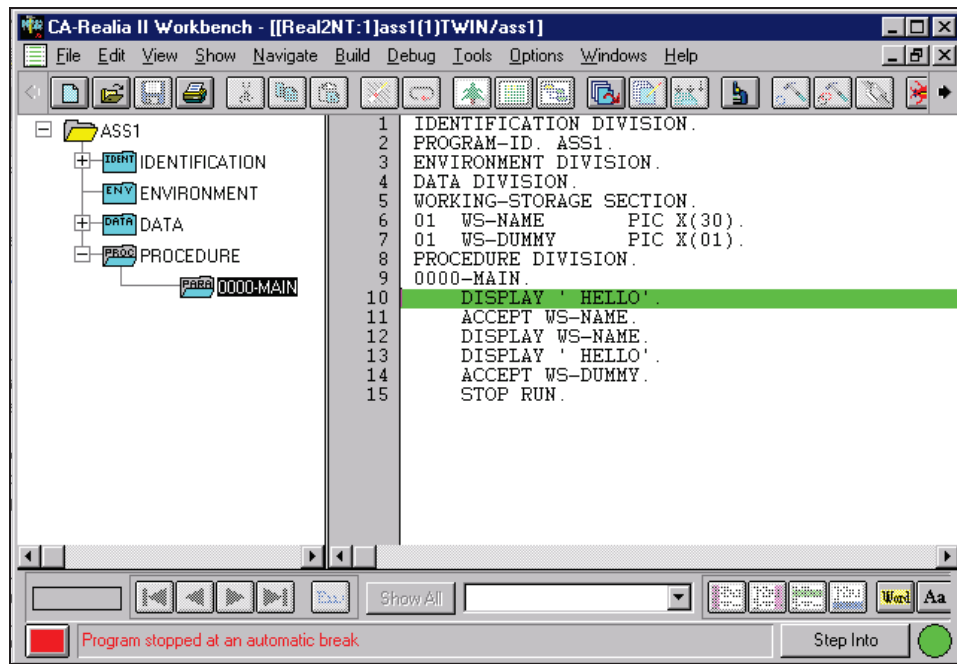


Figure 11: Debugging

Step 4: There are options for tracing in the debug menu. Some important options are as follows:

- Auto-Step:** In this animation style of execution, the program executes statement by statement. The execution path is visible and you can interrupt at any time by using the **Interrupt hot key (Ctrl+Q)**, the **Debug Session Interrupt** command, or the **Interrupt** auto-button.
- Step-into:** This command or auto-button is available when the debugger reaches a CALL statement. When you choose **Step Into**, the debugger steps into the called program and positions the highlighter on the first line.
- Step-out:** This command is available when the current line is within a called program. Choose **Step out** to stop stepping through the called program. The remaining lines of code are executed and the debugger highlights the first executable statement after the CALL statement.
- Step-Over:** This command is available when the debugger reaches a CALL statement. When you choose **Step Over**, the debugger executes the code of the called program. However, it never steps through the code line by line. The only exception is if there is a **breakpoint** in the called program. The debugger honors breakpoints in called programs, and will stop at the breakpoint instead of stepping out of the called program.
- Break points:** They can be set to start debugging from various points in the code. In debugging, breakpoints are predetermined statements in the program where execution of the program is interrupted, either unconditionally or under certain conditions.

- 3. Invoking Subroutines

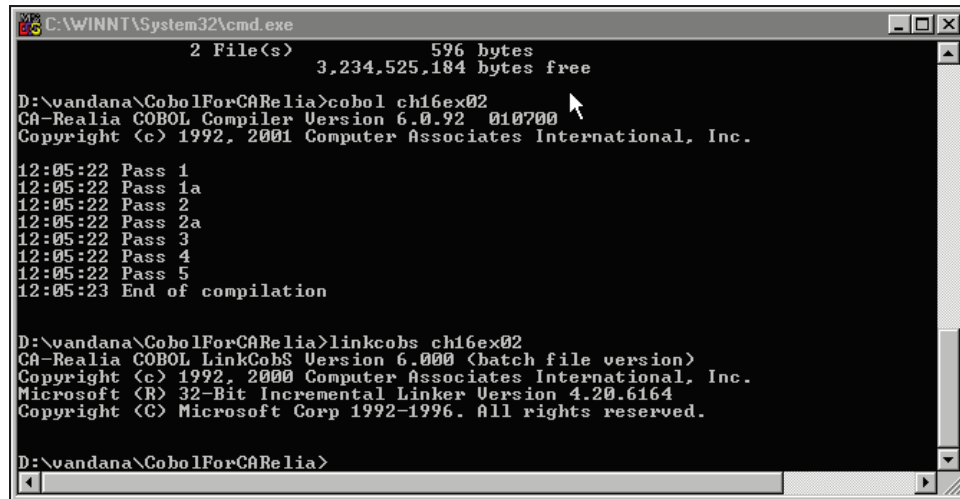
Creation of .DLL for invoking a Sub Program

To invoke a subprogram a .DLL has to be created. To create a .DLL, follow the steps given below:

Step 1: Set the environment settings using the following options:

- Run the **real2env.bat** in the command prompt, and ensure that the path contains **Path=D:\CAWB31**;
- Observe that other environments settings are done automatically and they can be seen by typing set on the command prompt.

Step 2: Compile and link the COBOL program. This creates the DLL.



```

C:\WINNT\System32\cmd.exe
2 File(s)          596 bytes
3,234,525,184 bytes free

D:\vandana\CobolForCARelia>cobol ch16ex02
CA-Realia COBOL Compiler Version 6.0.92 010700
Copyright (c) 1992, 2001 Computer Associates International, Inc.

12:05:22 Pass 1
12:05:22 Pass 1a
12:05:22 Pass 2
12:05:22 Pass 2a
12:05:22 Pass 3
12:05:22 Pass 4
12:05:22 Pass 5
12:05:23 End of compilation

D:\vandana\CobolForCARelia>linkcobs ch16ex02
CA-Realia COBOL LinkCobS Version 6.000 (batch file version)
Copyright (c) 1992, 2000 Computer Associates International, Inc.
Microsoft (R) 32-Bit Incremental Linker Version 4.20.6164
Copyright (C) Microsoft Corp 1992-1996. All rights reserved.

D:\vandana\CobolForCARelia>
  
```

Figure 12: DLL

Step 3: In the calling program, (for example: in CH16EX01.CBL), select the following options before compiling the program.

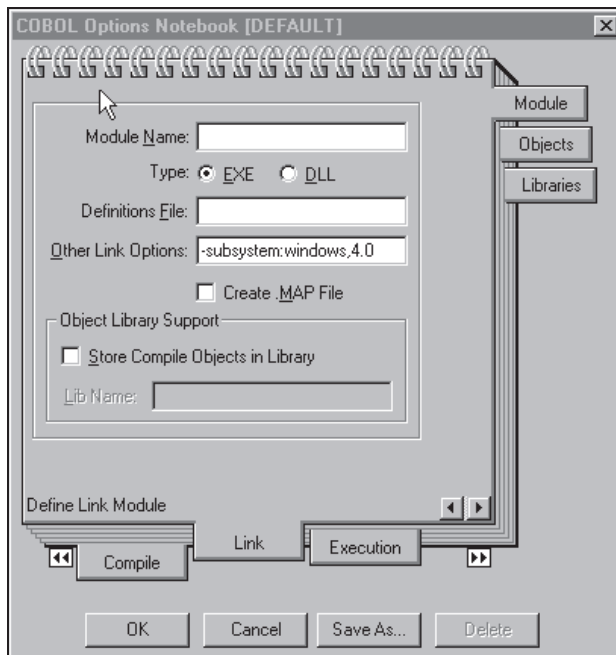


Figure 13: COBOL Options Notebook

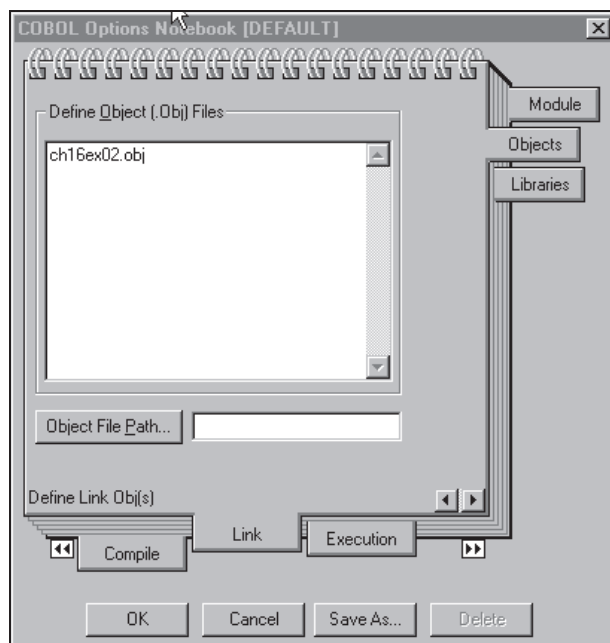


Figure 14: COBOL Options Notebook

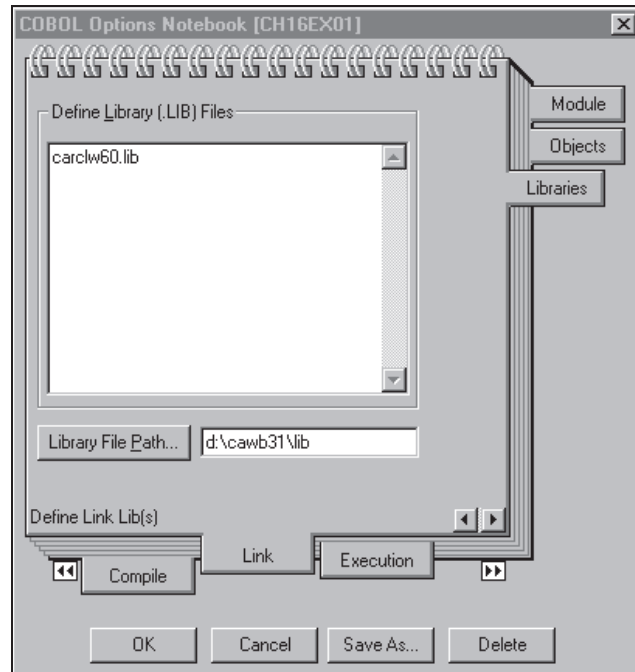


Figure 15: COBOL Options Notebook

Step 4: Compile, link, and execute the calling program.

Note: Creation of DLL is optional. You can compile the called program to create the .Obj file, which can be moved to the library path specified in library path option of the link tab.

COBOL STANDARDS

QMS/1001/STD
Revision No.1.2
(07-Nov-1996)

This document lays down the standards for programming in COBOL. These need to be adhered to in order to achieve uniformity throughout IGATE.

DOCUMENT HISTORY

<u>Date</u>	<u>Revision</u>	<u>Change</u>
04-Sep-1993	0.1D	Created Initial Draft
17-Dec-1993	1.0	Baselined
01-Dec-1994	1.1	Reorganized
04-Nov-1996	1.11D	Reviewed and Revised
07-Nov-1996	1.2	Baselined

DOCUMENT CONTROL INFORMATION**Prepared By**

Saurabhi Kulkarni

Kerman Balsara

Maintained By

All suggestions, comments, change requests and request for copies should be addressed to the CQA Group.

Approved By

Name : Satish M. Joshi

Designation : General Manager

Date :

Signature :

Released By

Name :

Designation : Manager - Quality Assurance

Date :

Signature :

CONTENTS

1.	INTRODUCTION	61
2.	IDENTIFICATION DIVISION	63
3.	ENVIRONMENT DIVISION	65
4.	DATA DIVISION	65
5.	PROCEDURE DIVISION	68
6.	READABILITY FACTORS	61
7.	CODING CONSIDERATIONS	75
8.	DESIGN CONSIDERATIONS	81

1 INTRODUCTION

1.1 The need for a Set of Standards

To ensure that IGATE delivers a quality product, there is a requirement within IGATE for a set of COBOL programming practices and procedures. Code written adhering to these standards will have better readability and maintenance will be easier as well as efficient. Also the time taken for a person to understand the purpose, structure and functionality of the code written by another person will be minimized. Overall the productivity will increase along with reduction in development time as well as maintenance costs.

In addition, the use of standards will improve the consistency of programming code, enhancing the appearance of programs to a great extent. When an applications product using these standards will be delivered to any client it will be a professional, consistent and well finished product.

No set of standards can however, ensure that programmers adhere to them. Thus it is essentially up to the individuals to follow strict discipline in using the standards when writing code. There are a number of guidelines and recommendations which have been included in this document which can be followed in order to design and code good COBOL programs.

1.2 Scope

The standards described in this document are to be followed when writing programs for any COBOL project or product being developed by IGATE.

For any other component of the program, use the respective standards. e.g. CICS standards, DB2 standards, IDMS standards etc.

It should be noted that any client specified standards will override these standards.

1.3 Overview

Taking the COBOL program structure into consideration, the standards have been laid down for coding each division.

Identification Division : Standard comment block which will give general and functional information about the program.

Environment Division : File naming conventions.

Data Division : Naming conventions for data items, logical grouping of data items and their usage.

Procedure Division : Naming conventions for paragraph names.

Readability Factors : Emphasizes the need and provides guidelines for alignment, comments. Use of these guidelines will improve readability of the program substantially.

Coding considerations : Gives standards for writing a structured program. Use of these will enable the programmer to understand and maintain the program in a better fashion.

Design considerations : Gives standards for creating FD's, copyfiles, designing screens, giving control report etc.

2 IDENTIFICATION DIVISION

- 2.1 Program-id must be coded and should be same as the external filename. For IBM users, the program-id should be equal to the TSO member name.
- 2.2 A detailed comment box with the following information should be placed just before the ENVIRONMENT DIVISION.

System Name

Author Name

Brief description of the function of the program

Brief outline of the main logic of the program

Tables and files used by the program along with the mode in which they are opened

All reports generated with their names and descriptions

Invocation method & parameters for the program

Subroutines called along with the parameters passed to them

Change log giving WRI(Work Request Id), modified/created by, modification/creation date, and details of the changes made

External procedures (JCL,etc.) used to run the program.

Screens used, if any like map names etc.

A standard block should be made as :

```

*                               *
* System name   :               *
*                               *
* Author name   :               *
*                               *
* Program Title:One line description of the program*
*                               *
* Purpose      : Explain the purpose of the      *
*               program                          *
*                               *
* Input        :               *
*                               *
* Process      : Brief logic of the program      *
*                               *
* Output       :               *
*                               *
* Subroutines called and parameters passed :      *
*                               *
* Screens used :               *
*                               *
* JCL to be used:               *
*                               *
*****
*               Work Request Log               *
*****
* Date   Programmer WRI   Description   *
*                               *
* 99/99/99 x.xxxxxxx  xx999  xxxxxxxxxxxxxxxxxxxxx*
*               xxxxxxxxxxxxxxxxxxxxx*
*                               *
*****

```

EJECT

3 ENVIRONMENT DIVISION

3.1 Configuration Section

- 3.1.1 If SOURCE-COMPUTER and OBJECT-COMPUTER clauses are to be coded, then use of a copyfile is recommended. Refer to Design considerations.

3.2 Input-output Section

- 3.2.1 Use of standard copyfiles for SELECT clause for the files used in the program is recommended. The copyfile will give the logical and physical file names to be used. Refer to Design considerations.

4 DATA DIVISION

4.1 File Section

- 4.1.1 Copyfile should be used for the FD declaration. Refer to Design considerations.

4.2 Working-storage Section

- 4.2.1 All the copyfiles that are used by this section should be copied at the end of this section.
- 4.2.2 Data definitions should start from 01 level. Further, as much as possible, all levels should be incremented in multiples of 5.
- 4.2.3 All the numeric data fields that are used for internal processing should be defined as COMP-3 with odd number of digits. This improves speed of arithmetic operation and conserves storage.
- 4.2.4 All the datanames (including subscripts) should be meaningful and at least 8 chars long and should not have names like WS-A, WS-B etc. Economy in length of data names should not be done at the cost of understanding and readability.

- 4.2.5 All 88-level entries should be prefixed with '88-'.
- 4.2.6 Data items used for similar purpose should be defined together as a logical group. In some cases the processing logic also needs to be considered. e.g. If monthly, quarterly and yearly totals handled in a program, these can be grouped in two ways :

(1) All totals for a field are grouped together :

```
01      W01-TOTALS.  
      05  W01-FIELD1-MNLY-TOT.  
      05  W01-FIELD1-QTLY-TOT.  
      05  W01-FIELD1-YRLY-TOT.  
  
      05  W01-FIELD2-MNLY-TOT.  
      05  W01-FIELD2-QTLY-TOT.  
      05  W01-FIELD2-YRLY-TOT.
```

(2) The period totals for all fields are grouped together :

```
01      W01-TOTALS.  
      05  W01-FIELD1-MNLY-TOT.  
      05  W01-FIELD2-MNLY-TOT.  
  
      05  W01-FIELD1-QTLY-TOT.  
      05  W01-FIELD2-QTLY-TOT.  
  
      05  W01-FIELD1-YRLY-TOT.  
      05  W01-FIELD2-YRLY-TOT.
```

If the program logic cumulates figures into monthly, quarterly and yearly totals in the same paragraph, then these datanames should be defined as in (1).

If the program logic is such that all monthly totals are done in one paragraph, all quarterly in another and all yearly in a third paragraph then the datanames should be defined as in (2).

- 4.2.7 WORKING-STORAGE groups should be named in ascending order.
eg. W01-, W02- etc.

All datanames in a group should be prefixed with the group prefix and suffixed with 2 or 3 characters indicating the use of the variable. e.g. -SW for Switches, -PRT for print lines, -CNT for counters, -ARR for Arrays, -IND for indexes, -LIT for literals, -SUB for subscripts, -FLAG for flags etc.

eg. 01 W01-COUNTERS.
05 W01-READ-CNT.
05 W01-WRITE-CNT.
05 W01-MODIFIED-CNT.

- 4.2.8 When defining picture, use 2-digit element length. eg. write PIC X(01) instead of PIC X and PIC X(05) instead of PIC X(5).

- 4.2.9 Do not use 77 levels since logical grouping of elements is not possible with 77 level.

- 4.2.10 Report layout should appear after all other variables are defined. The following convention should be used in the report layout :

Header : H01 - H99
Detail : D01 - D99
Total : T01 - T99
Footer : F01 - F99

Later, during modifications, if a new group logically falls between two groups, use alphabets for sequencing. eg. H01A-, H01B- etc.

- 4.2.11 One subscript should NOT be used for more than one table, even if the tables are of similar type. Use different subscripts for each table/array defined. The subscript name should include 3 characters of the table name.

4.2.12 All subscripts should be defined with 'USAGE IS INDEX' and should not be used for any other purpose than subscripting. The advantage of defining them in this manner is that they are automatically assigned PIC S9(04) COMP.

4.2.13 There should be no datanames declared by the programmer which are not used in the PROCEDURE DIVISION.

4.3 Linkage Section

4.3.1 Use prefix 'LN-' for all fields in the linkage section.

5 PROCEDURE DIVISION

5.1 Every paragraph name should be made up of a 4 digit number followed by a '-' and a meaningful name (at least 8 chars) describing the function performed by it. Thus the length of a paragraph name should be at least 13 characters long.

5.2 The 4 digit number in a paragraph name should be incremented by 100 when the program is initially coded so as to facilitate insertion of paragraphs at a later stage.

The exit paragraph name should have the corresponding paragraph number followed by the word '-EXIT'. e.g.

```
3000-PROCESS.  
    PERFORM          3100-CALCULATE THRU  
    3100-EXIT.  
  
    PERFORM          3200-CALCULATE-INTEREST THRU  
    3200-EXIT.  
  
    PERFORM          3300-CALCULATE-TAX THRU  
    3300-EXIT.
```

```
3000-EXIT.  
EXIT.
```

- 5.3 New paragraphs inserted during enhancement, should be defined with the 4 digit number which will divide the available range in exactly two halves. In the above example, if a paragraph is to be inserted between the 3100-CALCULATE and 3200-CALCULATE-INTEREST paragraphs, the number should be 3150-.
- 5.4 If all numbers are exhausted then add another set of 4 digits. e.g. paragraph 4155- can have additional paragraph names as 4155-1000-, 4155-2000- etc. If it is initially known that the program will have a huge size, start paragraph numbering with 5 or more digits e.g 10000-.
- 5.5 When coding DECLARATIVES SECTION, code a different paragraph for each file, include file name in paragraph name. The following are the standards for declaratives :

```
*=====
DECLARATIVES.
```

```
9000-DECL-FILE1 SECTION.
```

```
    USE AFTER STANDARD ERROR PROCEDURE ON FILE1.
```

```
    IF W03-ABORT-FLAG = "Y"
      GO TO 9000-EXIT.
```

```
    MOVE W02-FILE1-STAT TO W04-FILE-STATUS.
```

```
    IF 88-FILE-DOES-NOT-EXIST
```

```
      MOVE 'Y'      TO W03-ABORT-FLAG
```

```
      MOVE 099      TO W10-ERROR-CODE
```

```
    PERFORM 9900-DISPLAY-ERR-MSG THRU
      9900-EXIT
```

```
    GO TO 9000-EXIT.
```

```
    ***** HANDLE OTHER ERRORS AS DESIRED
```

```
9000-EXIT.
```

```
EXIT.
```

```
*=====
```

```
9100-DECL-FILE2 SECTION.
```

```
    USE AFTER STANDARD ERROR PROCEDURE ON FILE2.
```

```
    IF W03-ABORT-FLAG = "Y"
      GO TO 9100-EXIT.
```

```
    MOVE W02-FILE2-STAT TO W04-FILE-STATUS.
```

```
    IF 88-FILE-DOES-NOT-EXIST
```

```
      MOVE 'Y'      TO W03-ABORT-FLAG
```

```

MOVE 099      TO W10-ERROR-CODE
PERFORM 9900-DISPLAY-ERR-MSG THRU
              9900-EXIT
GO TO 9100-EXIT.
***** HANDLE OTHER ERRORS AS DESIRED

```

```

9100-EXIT.
EXIT.
*=====
***** DECLARATIVES FOR OTHER FILES
END DECLARATIVES.
*=====

```

6 READABILITY FACTORS

6.1 Alignment Rules

6.1.1 Align the 'PIC' clause, 'VALUE' clause and 'SPACES'/'ZEROS' for all variables defined. Throughout the program, use 'SPACES' and not 'SPACE'. Similarly, use 'ZEROS' and not 'ZEROES' or 'ZERO'.

6.1.2 The level number 05 should be aligned with the dataname of 01 level entry, level number 10 should be aligned with dataname in 05 level entry, and so on. Two spaces are required between level-number and dataname. Thus the 01 level should start from 8th column, 05 level from 12th column, 10 level from 16th column and so on. e.g.

```

W01-HEADING-LINES.
05      W01-REP-HDG1.
        10 FILLER PIC X(10) VALUE SPACES.
        10 FILLER PIC X(03) VALUE "IGATE".

```

6.1.3 Align 'MOVE's and 'TO's as much as possible in a paragraph. But if not possible, they should be aligned at least for a logical block of code.

e.g.

```

MOVE _____ TO _____.
      MOVE _____ TO _____.
      MOVE _____ TO _____.

```

If the 'MOVE TO ' cannot fit on the same line, align the source and destination variables on consecutive lines.

e.g.

```

MOVE _____ TO
_____

```

6.1.4 Every pair of IF and ELSE should be aligned. Condition/Statement after ELSE should be aligned with the condition after IF.

e.g.

```

IF 88-VALID-NAME
    IF 88-VALID-PHONE
        |
    ELSE
        |

ELSE
    IF 88-VALID-ID
        |
    ELSE
        |.
  
```

6.1.5 It is recommended that long lines of code should be split and placed on multiple physical lines and aligned properly.

6.2 All switches should be defined along with the VALUE clause. Comments should be added along with the definition about the various values it takes, their significance, paragraph names where they are set and paragraph names where they are tested.

Information should be provided in the following format :

```

*****
* Wnn-FIRST-SW                               *
* Used for (description in 2/3 lines)         *
* Set in paragraphs <para-name>....          *
* Tested in paragraphs <para-name>...        *
*                                           *
*****
  
```

6.3 The use of nested IF's should be restricted to 3 levels only.

6.4 Every ELSE should be the only statement on the line. Any statement that follows the ELSE should start on the next physical line.

6.5 Do not use NOT with AND and OR.
[As far as possible, positive conditions should be used.]

6.6 The use of complicated conditions like

IF NOT AND (OR NOT)
(AND) OR ... etc.

should be avoided. Such situations should be handled by coding multiple elementary IF's.

6.7 When more than one conditions are involved in an IF-ELSE statement, align them as follows :

e.g IF (condition-1) AND
((condition-2) OR (condition-3)) AND
(condition-4)
.....

ELSE

.....
.....

6.8 Paranthesis should be used when coding multiple conditions in single IF.

6.9 When assigning values to more than one dataname, multiple datanames should not be cluttered on one line. Instead they should be coded on separate lines with all the datanames well indented. Same rule should be followed when opening or closing multiple files in a single command.

6.10 A blank line should be inserted before and after every paragraph definition.

6.11 A blank line should be inserted after every logical group of code.

6.12 All the paragraphs should physically be placed in an ascending order.

6.13 Avoid coding paragraphs of length exceeding one printed page.

- 6.14 For every paragraph, comments must be written. These comments should be enclosed in the standard comment box, which should be placed before the respective paragraph definition. The comments should include the paragraph name, explain the function of the paragraph and the transfer of control from that paragraph.

```
*****
* <para-name>                                     *
* Brief explanation of function of the paragraph. *
* Transfer of control :                           *
* <called para-name> : reason for transfer      *
*      :                                         *
*      :                                         *
*      :                                         *
*****
```

- 6.15 Comments should always add value, not just restate what the code does. i.e. explain 'WHY' and not just the 'WHAT'.

Comments should be modified when the code is changed.

Generally comments should not be in-stream, except where the code needs explanation due to complexity or changes being introduced. Another exception to this is, when a message code is used in the program, the actual message should be coded as a comment just above that line. This improves readability and helps in understanding the program logic.

When maintaining existing code,

- * If a block of code needs to be added, place a comment line at the begin and at the end of the block stating the WRN for which the change is made.
- * If only one or two lines need to be added, add only one comment line before the new code stating the WRN and number of code lines being added.
- * Do NOT delete any existing lines. Only comment them out if they are not required. Use the above method for explaining the action. When commenting original lines, '*D' can be used so that these lines can be easily located if necessary.

- * When modifying code, do not modify the lines directly. Instead comment and add new lines.
- * All lines which have been added or commented for change should have the WRN in the columns 73-80.

7 CODING CONSIDERATIONS

- 7.1 Use TOP-DOWN approach when creating the layout of the program. The standard MAINLINE-PARA should contain only these three paragraphs.

0000-MAINLINE-PARA.

```
PERFORM 1000-INIT THRU --- Open files,
                        1000-EXIT.      init variables
PERFORM 2000-PROCESS THRU --- Main process
                        2000-EXIT.      logic
PERFORM 9000-WRAPUP THRU --- Close files,
                        9000-EXIT.      statistics.
```

0000-EXIT.
EXIT.

[Have common para at the end with a four digit prefix of 9999-, 9000 etc.]

- 7.2 There should be only ONE Open & Close statement per file. Project leader's permission should be sought before using more than one OPEN/CLOSE.

All open files should be closed before exiting from the program, whether normally or abnormally.

File status should be checked after every file handling statement.

- 7.3 In any IF condition 88-level datanames should be used instead of using the corresponding data item.

As a good programming practice,

- a> For a group of conditions connected by AND, the most unlikely conditions should be tested first.

b> For a group of conditions connected by OR, the most likely conditions should be tested first.

7.4 The scope of each paragraph should be what the paragraph name indicates.

e.g. The scope of a read paragraph should not extend beyond

- * Setting of key values
- * Reading the file
- * Checking Status
- * Building of control totals

7.5 Branching from a paragraph

Branching from a paragraph (Via a PERFORM) should only be to function related sub-paragraphs or to paragraphs performing common functions. For related paragraphs, a proper sequence of paragraph numbering should be followed and for the common paragraphs a certain range of numbers could be reserved, grouped by the function that these paragraphs perform. e.g. 8000-'s & 9000-'s could be for common routines.

- * The called paragraph number should always be higher than the calling paragraph number.
 - * The calling and called paragraph numbers should lie within the same thousands (ie. they should have the same first digit) if the called paragraph is not a common paragraph. This logic should also be carried down to the hundredths and tens position.
- e.g. 3100- can be called only from 3000-
3110- can be called only from 3100-
3111- can be called only from 3110-.

An exception to this is the 2000-PROCESS paragraph which controls the main logic of the program. Thus this can call 3000-, 4000- etc though they are not common paragraphs.

7.6 Every paragraph should be performed thru its exit.

7.7 'PERFORM..THRU' should be used instead of 'GO TO', in order to keep the programs more structured. However 'PERFORM' should also be used with care, so that there is no overlap in 'PERFORM' ranges and the number of levels in a program are not too many.

7.8 To avoid unstructured coding, the use of 'GO TO' should be restricted only to the exit of the current paragraph. 'GO TO' should NOT be used for any other transfer of control except in exceptional cases given below. Even in this structure 'GO TO' should be used only to transfer control to exit paragraph or to the continue paragraph.

e.g. 2000-process.
.....
.....
If condition
 go to 2000-01-continue.
.....
.....

2000-01-continue.
.....
.....

2000-exit.
 exit.

When there is more than one paragraph in this structure, the paragraphs in between should have a suffix of -nn along with the paragraph no. (example 2000-01-continue).

7.9 While deciding upon the structure of the program, it is necessary that functions which are common i.e. those which can be executed or called a number of times from various places, should be coded in separate paragraphs. Also these paragraphs should be placed towards the end of the program with numbers higher than all the paragraphs that perform these routines.

7.10 COMPUTE statement should be used only when complex expressions are involved. It should not be used when doing simple computations like incrementing value of a single data item, etc. ROUNDED clause must be coded for all COMPUTE statements.

7.11 The body of the procedure division should be as follows:

0000-MAIN.

```
PERFORM 1000-BEGIN      THRU
      1000-EXIT.
PERFORM 2000-PROCESS    THRU
      2000-EXIT.
PERFORM 9000-CLOSE      THRU
      9000-EXIT.
```

0000-EXIT.

EXIT.

1000-BEGIN.

```
PERFORM 1100-INITIALIZE-DATANAMES THRU
      1100-EXIT.
PERFORM 1200-OPEN-FILES    THRU
      1200-EXIT.
PERFORM 1300-ACCEPT-PARAMETERS THRU
      1300-EXIT.
```

|

1000-EXIT.
EXIT.

1100-INITIALIZE-DATANAMES.

|

1100-EXIT.
EXIT.

1200-OPEN-FILES.

|

1200-EXIT.
EXIT.

1300-ACCEPT-PARAMETERS.

|

1300-EXIT.
EXIT.

- 7.12 All the variables should be initialized before use eg. variables defined in tables which are used for computation.
- 7.13 Avoid reuse of variables for different purposes to conserve storage.
- 7.14 ALTER command should not be used.
GOTO depending on should not be used.
- 7.15 MOVE/ADD/SUBTRACT corresponding should not be used since same element names are required in two group items.
- 7.16 The number of switches used in the program should be kept to minimum without affecting the complexity of code.
- 7.17 'EJECT' statement should be appropriately placed in the code.
- 7.18 Avoid using SECTIONS except when using declarative section where it is mandatory.
- 7.19 In report programs, page EJECT should be given at the end of the report and not at the beginning of the report.
- 7.20 When using subroutines, if appropriate, use 'CANCEL' after returning to calling program.
- 7.21 When a program aborts, an information block giving at least the following should be displayed. The corresponding variable names and their usage should be standardised throughout the project by the Project Leader.

Date :
Time :
Program :
Paragraph :
Message :

- 7.22 Program termination statements should be coded.
- 7.23 When making changes to existing code, at times it is better to follow the existing program style even if it means deviation from the standards. But this should be done only after approval from the Project Leader. e.g. Consider a program in which datanames are defined as WS-TOTAL1,..., WS-TOTAL4. If you have to incorporate one more similar total, define it as WS-TOTAL5 instead of the standard Wnn-TOTAL5.

Consider the following piece of code :

```
1000-START-PARA.  
    Read next input-file.  
    ;
```

```
      If condition-1
        GO TO 1000-START-PARA.
      :
      If condition-2
        GO TO 1000-START-PARA.
```

```
1000-EXIT.
      EXIT.
```

There are non-standard GO TO's in this piece of code. But if we are required to add one more similar condition in the above paragraph, it is better to add it in the same style.

7.24 Wherever available :

- * INITIALIZE command should be used to initialize group items.
- * Use of in-stream 'PERFORM' could be done, with maximum of 10 lines in the 'PERFORM'.
- * Scope terminators should be used wherever available. e.g.
Every IF & IF-ELSE should be paired with END-IF

```
IF 88-VALID-NAME
  IF 88-VALID-PHONE
    |
  ELSE
    |
  END-IF
```

```
ELSE
  IF 88-VALID-ID
    |
  ELSE
    |
  END-IF
END-IF.
```

EVALUATE, END-EVALUATE should be used when more than three 'IF's are used at the same level.

8 DESIGN CONSIDERATIONS

8.1 General

- 8.1.1 Data definitions should start from 01 level. Further, all levels should be incremented in multiples of 5.

- 8.1.2 All the numeric data fields that are used for internal processing should be defined as COMP-3 with odd number of digits. This improves speed of arithmetic operation and conserves storage.
- 8.1.3 All the datanames should be meaningful and at least 8 chars long. Economy in length of data names should not be done at the cost of understanding and readability.
- 8.1.4 All 88-level entries should be prefixed with '88-'.
- 8.1.5 When defining picture, use 2-digit element length. eg. write PIC X(01) instead of PIC X and PIC X(05) instead of PIC X(5).
- 8.1.6 Do not use 77 levels since logical grouping of elements is not possible with 77 level.
- 8.1.7 For all error messages, use of either a VSAM file or ISAM file or a DB2 table or a relative file is recommended. A standard error handling routine which will take the error message code from the program and get the actual message associated for that code from the data store should be used.
- 8.1.8 A logging and recovery mechanism should be designed to avoid loss of data consistency due to system / program failures.
- 8.1.9 All files used in the system will have the following Copy files :

SOURCE-COMPUTER, OBJECT-COMPUTER clauses should be put in a file, say, 'CONFIG'.

File Name + suffix of 01 for SELECT Clause.

File Name + suffix of 02 for (FD) Clause and record layout.

File Name + suffix of 03 for File Status variable declarations.

File Name + suffix of 04 for Working-Storage record layout.

When using the Working-Storage copyfile, use a COPY statement as follows :

COPY file-name REPLACING WS by Wnn.
- 8.1.10 At the end of every program, a control report should give the statistics of the Number of Records Read, Processed, Written, Deleted, Modified etc.
- 8.1.11 Internal COBOL SORT should be avoided wherever possible. Instead, use external sorts.

8.2 Configuration Section

- 8.2.1 The SOURCE-COMPUTER and OBJECT-COMPUTER clauses should be kept in a copyfile which will be copied by every program.

8.3 Input-Output Section

- 8.3.1 The SELECT clause for all files to be used in the system should be kept in a copyfile which will be copied by every program using that file. The logical file name should be meaningful and should be at least 8 characters long. The physical filename must be suffixed by a 01 (either as part of the filename or as an extension).

e.g. The logical filename for a Cash Transaction File can be CASH-TRANS-FILE. The physical filename can be CASH-TRANS-FILE.01 (or CSHTRN01 if only 8 character filenames are supported)

- 8.3.2 The 'ASSIGN TO' name for all files should be the same as the file name.

e.g. The Select Clause for a Cash Transaction File can be written as

```
SELECT CASH-TRANSACTION-FILE  
ASSIGN TO CASH-TRANSACTION-FILE etc.
```

8.4 File Section

- 8.4.1 The file descriptions should be kept in a copy file which will be copied by every program using that file. The physical filename must be suffixed by a 02 (either as part of the filename or as an extension).

e.g. The physical filename for a Cash Transaction File can be CASH-TRANS-FILE.02 (or CSHTRN02 if only 8 character filenames are supported).

- 8.4.2 For all files, the clauses
'BLOCK CONTAINS ' and
'RECORDING MODE IS ' and
'RECORD CONTAINS ' and
'LABEL RECORDS ARE'
should be specified.

- 8.4.3 The record name for every file should be the file name suffixed with '-REC'.

e.g. If the file name is CASH-TRANS-FILE then the record name should be CASH-TRANS-REC.

- 8.4.4 Each field in the file record should be prefixed by a three character abbreviation of the file name.

Field names of the CASH-TRANS-FILE can be prefixed with 'CTF-'.

- 8.4.5 The Copy File must contain comments to indicate the program/s which create the data in this file and the total length of the record.

- 8.4.6 At the end of every record, a FILLER must be defined to allow addition of variables at a later date.

8.5 Working Storage Section

- 8.5.1 The File Status variable (defined in the SELECT clause for a file) should be kept in a copyfile which will be copied by every program using that file. The physical filename must be prefixed by a 03 (either as part of the filename or as an extension).

e.g. The physical filename for storing the File Status variables for a Cash Transaction File can be CASH-TRANS-FILE.03 (or CSHTRN03 if only 8 character filenames are supported)

8.6 Screen Section

For screen interface standards, IBM users should refer to the standards of the corresponding software used e.g. CICS, ADS/O.

The following standards should be followed when using the COBOL screen section :

- 8.6.1 All screens should be designed using a Screen Generation Utility (if available) for the first time. Subsequently they should be inserted in the program. This will facilitate quick generation of code where screens have a large number of items.

- 8.6.2 A screen must have the following attributes:

The company name should be displayed on top appropriately centered with the system date and time on top right hand corner. The program name could be displayed on the top left corner.

The function of the screen should also be appropriately centered on the next line.

Mode to be displayed on top right hand corner below date e.g. add, modify, view etc. for file maintenance programs. The last two lines of the screen should be kept for the function line and the third last line for the messages to be displayed. Screen layout attached herewith.

Screen headings should be in all upper case (capitals).

Literals in screens other than headings to be capitalized.

Screen should not be cluttered up and importance is to be given for alignment and enhanced read ability.

Fields should be bright as against literals.

All numeric fields should be right justified all alpha fields left justified.

8.6.3 Screen group level names should be prefixed with 'SSnn-' where nn is a number between 01 and 99.

8.6.4 All data names in screens should be meaningful. By meaningful it is implied that data name shall indicate the function or attribute for which it is used.

8.6.5 Use of the BLANK SCREEN clause should be substituted by bottom-up line by line blanking. This gives a better touch as far as screen blanking is considered.

While blanking a screen it is to be noted that the standard header occupying the first few lines should never be blanked by the program. Line 24 is for error messages and will be blanked automatically by the next accept statement.

The rest of the screen must be blanked bottom-up. For this as many sub-groups as may be necessary should be defined. e.g.

01 SS01-BLANK-SCREEN.

05 FILLER.

10 SS01-BLANK-23 LINE 23 COL 01 BLANK LINE.
10 SS01-BLANK-22 LINE 22 COL 01 BLANK LINE.
10 SS01-BLANK-21 LINE 21 COL 01 BLANK LINE.
10 SS01-BLANK-20 LINE 20 COL 01 BLANK LINE.
10 SS01-BLANK-19 LINE 19 COL 01 BLANK LINE.
10 SS01-BLANK-18 LINE 18 COL 01 BLANK LINE.
10 SS01-BLANK-17 LINE 17 COL 01 BLANK LINE.
10 SS01-BLANK-16 LINE 16 COL 01 BLANK LINE.

05 SS01-BLANK-SCR-2.

10 SS01-BLANK-15 LINE 15 COL 01 BLANK LINE.
10 SS01-BLANK-14 LINE 14 COL 01 BLANK LINE.
10 SS01-BLANK-13 LINE 13 COL 01 BLANK LINE.
10 SS01-BLANK-12 LINE 12 COL 01 BLANK LINE.
10 SS01-BLANK-11 LINE 11 COL 01 BLANK LINE.

05 FILLER.

10 SS01-BLANK-10 LINE 10 COL 01 BLANK LINE.
10 SS01-BLANK-09 LINE 09 COL 01 BLANK LINE.
10 SS01-BLANK-08 LINE 08 COL 01 BLANK LINE.

05 SS-BLANK-SCR-1.

10 SS01-BLANK-07 LINE 07 COL 01 BLANK LINE.
10 SS01-BLANK-06 LINE 06 COL 01 BLANK LINE.
10 SS01-BLANK-05 LINE 05 COL 01 BLANK LINE.
10 SS01-BLANK-04 LINE 04 COL 01 BLANK LINE.

05 FILLER.

10 SS01-BLANK-03 LINE 03 COL 01 BLANK LINE.

8.6.6 While prompting for confirmation with 'CONFIRM (Y/N)', the default value shall be "N" unless otherwise specified by the user.

8.6.7 Use of escape sequence shall have the following standard route unless otherwise specified by the user.

Escape from first field of the screen shall take control to the previous screen.

Escape from any other field of the same screen shall take control to the first field of the same screen.

8.6.8 While modifying data through screens, the following technique is to be adopted as a standard feature unless otherwise specified by the user :

The serial number of the field to be modified will be accepted and the cursor will be positioned directly on that field instead of having to skip through all the fields.

This applies only to screens having a large number of fields. Screens having a few fields need not adopt this method.

XXXXXXXX	IGATE PVT LTD COMPANY FUNCTION PERFORMED	99/99/99	HH- MM
MESSAGES TO BE DISPLAYED			
F1 :	F2 :	F3 :	F4 :
F5 :	F6 :	F7 :	F8 :