BACHELOR OF COMPUTER APPLICATIONS (BCA)

COBOL

CONTENTS

Syllabus

UNIT – I

LESSON – 1: INTRODUCTION TO COBOL

CONTENTS

1.0 AIMS AND OBJECTIVES

In this lesson, the learner will be introduced the History of Cobol , Structure of a COBOL program, Character set, words, data names and identifiers and Literals of COBOL. The objective here is to familiarize him the prerequisites to understand the language.

1.1 HISTORY OF COBOL

0In 1959, the new language named COBOL (**CO**mmon **B**usiness **O**riented **L**anguage) was introduced keeping in mind the business purpose applications.

The board of directorate which is known as CODASYL (Conference on DATA System Language) COBOL programming language committee established a COBOL maintenance committee to keep COBOL in step with the times. The first COBOL compiler became available in early 1962. The next version with some new additions was published in 1965. In August 1968 a standard version of the language was approved by the American National Standard Institute (ANSI). This standard version was again modified in 1974 and is known as ANSI-74 COBOL or COBOL -74. The revision process has been continuous and in 1985 a revised standard was introduced. This standard was known as COBOL-85.

1.2 FORMAT FOR COBOL PROGRAMS

COBOL is a high-level language. Hence, a COBOL program can be executed on a computer for which a COBOL compiler is available. The compiler translates a COBOL

source program into the machine language object program. This object program is really executed.

COBOL programs are written in coding sheets. There are 80 columns in a line of the coding sheet. The page number is coded in columns 1-3 and the line numbers are coded in columns 4-6. The page and line numbers together is called the sequence number. Depending on the type, the entries are coded both from column 8 or column 12 and in both cases it can be continued up to column 72. Columns 73-80 can be used to write some identification. The compiler ignores anything that is given in columns 73-80 except when a printed copy of the program is provided by the compiler in which case the entries in columns 73-80 are also listed. The use of the sequence number is also optional and can be omitted. However, when sequence numbers are provided they must appear in ascending order.

Column	**Field**
1-3	Page Number
4-6	Line Number (1-6 Sequence Number)
7	Continuation / Comment
8-11	A – Margin / Area A
12-72	B- Margin /Area B
73-80	Identification

In COBOL there are two types of entries known as margin A and margin B entries. Margin A entries start from column 8, 9, 10 or 11 and margin B entries start from column 12 or anywhere after 12.

An asterisk (*) in column 7 indicates a comment line and the entry is not compiled to produce object code. Comment lines are actually some notes revealing the intentions of the programmer. Since the compiler ignores them, anything can be included as comments. Comment lines can appear anywhere after the first line of the COBOL program. A comment line can also be indicated by using the character slash (/) in column 7. But in this case the comment line will be printed after causing a page ejection (i.e., after skipping to the top of the next page).

1.3 STRUCTURE OF A COBOL PROGRAM

Every COBOL program must have the following 4 divisions in the order in which they are specified below.

1. Identification division
2. Environment Division
3. Data Division
4. Procedure Division

In the Identification division the details about the author, date of writing the program etc will be specified.

In the Environment division, the details about the computer environment under which the program was written and compiled etc will be notified.

In the Data division, the variables that are used by the program will be defined and it is an important division for the program.

In the procedure division, all the programming statements (Executable Cobol statements) will be written and it is the most important division.

Under the divisions there are various sections intended for specific purposes. To name a few, working-storage section and File section come under Data division. Their purpose will be to allocate memory space for the variables and to notify the files that are to be used with the program.

A statement of a COBOL program can be written in one or more coding lines. To continue in the next line one has to use a hyphen (-) in column 7.

1.4 CHARACTER SET

To learn any language, first one must know the alphabets of the language and they are known as character set in general. There are 50 different characters in COBOL character set. They are listed below.

0-9	(10 numerals)
A-Z	(26 English alphabets-only capital letters)
–	(minus sign or hyphen)
+	(Plus sign)
*	(Asterisk)
/	(Slash)
0=	(Equal sign)
$	(Currency sign)
,	(Comma)
;	(Semi colon)
.	(Period or decimal point)
“	(Quotation mark)
(	(Left Parenthesis)
)	(Right Parenthesis)
>	(Greater than symbol)
<	(Less than symbol)

The characters **0-9** are called **numeric characters** or **digits**. The characters **A-Z** are called **letters** and the remaining characters are called **special characters**. The space or blank character in certain cases is treated as a letter.

1.5 COBOL WORDS

A COBOL word can be formed using the following characters:

0-9	
A-Z	(a-z)
-	(hyphen)

The following rules must be adhered in forming COBOL words.

(i) A word cannot begin or end with a hyphen.
(ii) A word can have at the maximum 30 characters.
(iii) One of the characters must be a letter. Some compilers put the additional restrictions that the first character must be a letter.
(iv) Except hyphen (-) no special character allowed.

Examples

Valid Word	Invalid Word & Reason
emp-sal	-pay (it starts with a hyphen)
	TOTAL MARK (blank space embedded)

There are 2 types of words in COBOL. A COBOL word can be either a user-defined word or reserved word. The reserved words are used in COBOL statements and entries for specific purposes by the COBOL compiler.

Some reserved words are given below:

ADD, SUBTRACT, DIVIDE, MULTIPLY, IF, PERFORM etc.

To know the complete set of reserved words of COBOL one can refer to the manual supplied with the compiler. Any attempt by the programmer to declare the reserved word will be indicated as an error during the compilation stage of the program.

1.6 DATA NAMES AND IDENTIFIERS

A **data name** gives reference to the storage space in the memory where the actual value is stored. This value takes part in the operation when that particular data name is used in the PROCEDURE DIVISION.

Identifier is a general term which means the single data name or a data name qualified, indexed or subscripted. Data names are only one form of identifiers. A data name must be a user-defined word and it cannot be a reserved word.

Examples

Valid Data Names	Invalid Data Names
NET-SALARY	COMPUTE (Reserved word)
TOT-MARK	MULTIPLY (Reserved word)
N100	23 (No letter)

1.7 LITERALS

The actual values can also appear in a program. Such values are known as literals.

For Example, the statement MOVE 0 TO TOTAL indicates that the value zero will be moved to the variable TOTAL. This constant 0 which used in the body of the statement is a literal.

A data name may have different values at different points of time whereas a literal means the specific value which remains unchanged throughout the execution of the program. For this reason **a literal is often called a constant**. Moreover the literal is not given a name; it represents itself and does not require to be defined in the DATA DIVISION.

There are 3 types of literals

a) numeric
b) nonnumeric.
c) figurative constants

(a) Numeric

A numeric literal can be formed with the help of digits only. It can have a sign (+ or -) and can have a decimal point also. If no sign is specified the literal will be taken as positive. Negative literals are indicated by – sign at the leftmost end. If no decimal point is used then the literal is obviously an Integer. If a decimal point is used, it must come in between the digits. The maximum number of digits allowed in a numeric literal is compiler dependent.

Examples

(i) Valid Numeric Literal	(ii) Invalid Numeric Literals
.123	‘’123’’(valid as nonnumeric literal but invalid as numeric literal)
12.5	- 23 (there is a blank space between the sign and the first digit 2)

(b) Nonnumeric

A nonnumeric literal is used in general to output messages or headings. Characters that are enclosed between “ “ constitute nonnumeric literal. The maximum number of characters that are allowed within two quotation marks is compiler dependent.

(iii) Valid Nonnumeric Literal

“BHARATHIAR”
“DATA DIVISION”
“100.50”
“HOUR/RATE”

(iv) Invalid Nonnumeric Literal

7	(valid as numeric literal but invalid as Nonnumeric literal)
“nine	(Invalid because there is no quotation mark on the right)
12.5”	(Invalid because there is no quotation mark on the left)

c) Figurative Constants

Figurative constants have some fixed names and the compiler recognizes these names and it sets up corresponding values in the object program.

Consider the statement given below :

MOVE ZERO TO COUNTER

Here value 0 will be moved to COUNTER by the compiler, as it recognizes ZERO and sets COUNTER with 0. Given below is the list of figurative constants.

Figurative Constant	Meaning
ZERO ZEROS ZEROES	value 0
SPACE SPACES	One or more blanks
HIGH-VALUE HIGH-VALUES	Highest value in the Collating sequence
LOW-VALUE LOW-VALUES	Lowest value in the Collating sequence
QUOTE QUOTES	one or more of "
ALL literal	one or more of the string characters comprising the literal

1.8 LANGUAGE DESCRIPTION NOTATION

The following notations will be used to describe the syntax of COBOL statements.

1. Words in CAPS & underlined → Key words (Compulsory in statements)
2. Words in CAPS & NOT underlined → Noise words (Optional in statements)
3. [entries] → can be included or dropped
4. { entries} → only one option to be selected
5. , ; → can be included or dropped
6. … → repetition of previous entry
7. Space character → used as a separator

1.9 LET US SUM UP

In the above lesson we have learnt the how the structure of a COBOL program will look like , character set , words, data names & literals of the COBOL language in detail. This will help you to learn the subsequent lessons comfortably.

1.10 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Describe the format of COBOL programs
2. Bring out the character set of COBOL
3. What do you mean by COBOL word? Give examples.
4. Explain about data names and identifiers

5. Discuss in detail the Literals of COBOL.

1.11 POINTS FOR DISCUSSION

1. Explain in brief about the structure of a COBOL program.
2. List out the character set of COBOL
3. What are the rules to be followed while forming a COBOL word? Give Examples.
4. Briefly explain about figurative constants.

1.12 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 2: DIVISIONS OF COBOL

CONTENTS

2.0 AIMS AND OBJECTIVES

In this lesson, the learner will be introduced the various divisions of COBOL and their contents. The objective here is to make the learner aware of the entries under each division and their importance.

2.1 IDENTIFICATION AND ENVIRONMENT DIVISION

The IDENTIFICATION AND ENVIRONMENT DIVISION are the first two leading divisions of any COBOL program. These divisions contain entries that are required to either identify the program or describe the computer system to be used for the compilation and execution of the program. Generally, the entries in these divisions are used only for documentation purposes.

2.2 IDENTIFICATION DIVISION

The IDENTIFICATION DIVISION is the first division of every COBOL source program. The paragraph PROGRAM-ID is essential in most of the machines. The other paragraphs are optional.

The structure of Identification division is given below.

IDENTIFICATION DIVISION.

PROGRAM-ID. Entry

[AUTHOR. entry.]
[INSTALLATION. entry.]
[DATE-WRITTEN. entry.]
[DATA-COMPILED. entry.]
[SECURITY. entry.]
[REMARKS. entry.]

} Optional

The division heading and paragraph names should be coded as margin A entries. The entries following the paragraph headings must be terminated by a period. The entry in the PROGRAM-ID paragraph contains the program name to be used to identify the object program.

The entries in the other paragraphs are normally treated as comments and the programmer is free to write anything for these entries. Only meaningful entries should be included in these places to provide better documentation and clarity.

The entry for the AUTHOR paragraph may include the name of the programmer. The entry of the DATE-COMPILED paragraph may contain the date of compilation. If this entry is left blank, the compiler inserts the actual date in the listing of the source program that may be printed during compilation.

2.3 ENVIRONMENT DIVISION

The ENVIRONMENT DIVISION is the second division in a COBOL source program. It is the most machine-dependent division. The computer and all peripheral devices required by the program are described in this division.

This division contains two sections

1) CONFIGURATION SECTION and
2) INPUT-OUTPUT SECTION.

Of these the CONFIGURATION SECTION appears first. The outline of the sections and paragraphs of this division is shown below.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. source-computer-entry.
OBJECT-COMPUTER. object-computer-entry.
[SPECIAL NAMES. special-computer-entry].
INPUT-OUTPUT SECTION.
FILE CONTROL. {file-control-entry}….
[I-O-CONTROL. input-output-control-entry].

For most compilers the COBOL source program must at least include the two section headings and the three paragraphs-SOURCE-COMPUTER, OBJECT-COMPUTER and FILE-CONTROL.

The division headings, section headings and the paragraph headings should be coded as Margin A entries. The paragraph headings must be followed by a period and then a space. The entries in the paragraphs are Margin B entries and can start in the same line with the paragraph heading.

2.4 CONFIGURATION SECTION

This section contains an overall specification of the computer used for the purpose of compilation and execution of the program. There are in all three paragraphs in this section, namely, source-computer, object-computer and special names.

2.4.1 SOURCE-COMPUTER

This paragraph specifies the name of the computer used to compile the COBOL program. The following is the form of this paragraph.

SOURCE-COMPUTER. computer-name.

For example, if ICL 1901 is to be used for compiling the COBOL source program, this paragraph should be as follows:

SOURCE-COMPUTER. ICL-1901

2.4.2 OBJECT-COMPUTER

The OBJECT-COMPUTER paragraph describes the computer on which the program is to be executed. The following shows the syntax for this paragraph.

OBJECT-COMPUTER. computer-name

[, MEMORY SIZE interger-1] { CHARACTERS / WORDS }

[, PROGRAM COLLATING SEQUENCE IS alphabet-name]

[, SEGMENT-LIMIT IS interger-2].

The computer name specifies a particular computer on which the object program is to be executed.

The MEMORY SIZE is used to indicate the amount of storage available to the object program. This clause is also used in conjunction with the SORT verb.

The PROGRAM COLLATING SEQUENCE clause specifies the collating sequence that is to be used to compare nonnumeric data items. The alphabet name in this clause should be defined in the SPECIAL-NAMES paragraph to specify a collating sequence. If this clause is absent, the machine's own collating sequence called NATIVE, is assumed.

The SEGMENT-LIMIT clause is used in most of the compilers to indicate that the sections having segment number less that the number specified in

integer-2 should be held in the permanent area of storage and should not be transferred to and from the virtual memory.

All the entries in this paragraph are terminated using period (a dot). The following is an example of the OBJECT-COMPUTER paragraph.

OBJECT-COMPUTER.

MEMORY SIZE 8000 WORDS.

2.4.3 SPECIAL-NAMES

This paragraph is used to relate some hardware names to user-specified mnemonic names. This paragraph is optional in all compilers. The following is the format of this paragraph.

SPECIAL-NAMES, [, CURRENCY SIGN IS literal-1]

[, DECIMAL-POINT IS COMMA]

[, CHANNEL integer IS mnemonic-names]…

[, ALPHABET alphabet-name IS { STANDARD-1 / NATIVE / implementor-name }]…

[, implementor-name IS mnemonic-name].

The CHANNEL clause is used to control the line spacing of line printers. This clause is used to associate a user-defined name called the mnemonic name with a channel in the printer carriage control. The range of integer depends on the particular line printer to be used. This mnemonic name can only be used in a WRITE statement.

The ALPHABET clause specifies a user-defined alphabet name that can be used to indicate a collating sequence in the PROGRAM COLLATING SEQUENCE clause discussed above or in the collating sequence clause in the sort verb. The word NATIVE stands for the computer's own collating sequence and STANDARD-1 stands for the ASCII collating sequence. The alphabet name is also used to define the external character set in which the data is recorded on a file.

The SPECIAL-NAMES paragraph can have other entries which are implementor-dependent. These entries are not discussed here as they are not of general interest. In fact, the CHANNEL entry is also an implementor-defined clause. The entries can appear in the paragraph in any order.

Consider the example : Let the SPECIAL-NAMES paragraph be as follows:

SPECIAL-NAMES. CHANNEL 1 IS PAGE-TOP.

We know that conventionally channel is associated with the top of the page. While instructing the computer to write a line on the line printer the programmer may like to specify that the line must be printed as the first line on a new page. This may be done by including the ADVANCE TO CHANNEL 1 clause in the write statement. The purpose of the special name clause illustrated above is to enable the programmer to replace CHANNEL 1 by PAGE-TOP in the ADVANCING clause. The idea is to provide better documentation. If one prefers to use CHANNEL 1 in the write statement, the special name entry is not required.

2.5 INPUT-OUTPUT SECTION

This section contains information regarding files to be used in the program. There are two paragraphs in this section- FILE-CONTROL and I-O-CONTROL.

FILE-CONTROL is used almost in every program. In the following some of the entries of the FILE-CONTROL paragraph will be discussed.

The INPUT-OUTPUT SECTION is optional in many computers.

2.5.1 FILE-CONTROL

The FILE-CONTROL paragraph names each file and identifies the first medium through file control entries. The simplified format of a file control entry is given below.

SELECT [OPTIONAL] file-name ASSIGN TO hardware-name.

In general, a COBOL source program uses some files. For each of these files, there must be a FILE-CONTROL entry. This entry names the file and assigns a peripheral device which holds that particular file. The file names that appear in the SELECT clauses must be unique and all these files must be described in DATA DIVISION. The file name should be formed according to the rules of data names.

The word OPTIONAL may be used only for input files. When the object program is executed, the optional files need not be present on every occasion. If the optional clause is omitted for a particular file, the file must be present during the execution of the program. If the file is absent, an execution error will occur. On the other hand, if an optional file is absent, any attempt to open the file for reading will not result in an error. But the absent file will be considered to be an empty file which means that the file does not contain any record.

The assign clause assigns a particular physical peripheral device name to a file. The physical peripheral device names are machine-dependent. We shall use the device names READER, PRINTER, TAPE and DISK to mean card reader, line printer, magnetic tape and magnetic disk device respectively.

An example of the FILE-CONTROL paragraph is given below.

```
FILE-CONTROL.
        SELECT CARD-DESIGN ASSIGN TO READER
        SELECT PRINTER-FILE ASSIGN TO PRINTER.
```

This paragraph indicates that there are two files - CARD-DESIGN and PRINTER-FILE. The file named CARD-DESIGN is a card file while the other is a report file to be printed on a line printer.

2.6 DATA DIVISION

The DATA DIVISION is that part of a COBOL program where every data item processed by the program is described.

It is important to note that unless a data item is described in the DATA DIVISION, it cannot be used in the procedure division.

The DATA DIVISION is divided into a number of sections such as File Section, Working-storage section, Screen section, Linkage section and Report Section. Depending on the use of a data item, it should be defined in the appropriate section.

Let us consider now 2 sections, namely, FILE SECTION and WORKING-STORAGE SECTION :

a) FILE SECTION

The FILE SECTION includes the description of all data items that should be read from or written onto some external file.

b) WORKING-STORAGE SECTION

The data items which are developed internally as intermediate result as well as the constants are described in this section of the DATA DIVISION.

The format of the DATA DIVISION is as follows:

```
DATA DIVISION.
[FILE SECTION.
File section entries.
……
……]
[WORKING-STORAGE SECTION.
 WORKING-STORAGE ENTRIES.
 ……
 ……]
```

All the section names as well as the division name must be coded as margin A entries, Each section of the DATA DIVISION is optional which means that a section may be omitted if there is no data that may be described in a particular section. It is important to note that the sections to be included must appear in the order shown above.

2.7 LET US SUM UP

The above lesson gives the learner clear ideas about the Identification, Environment and Data division. Having learnt this, the learner will be in a position to use them in the programs for better documentation.

2.8 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. What are the divisions of COBOL?
2. Sketch out the structure of Identification division.
3. What are the entries under Environment Division?
4. Write notes on Input-output section.
5. Describe the format of Data Division.

2.9 POINTS FOR DISCUSSION

1. Explain in detail about
   a. Identification Division
   b. Environment Division
2. Write notes on Data Division

2.10 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 3: PICTURE CLAUSE CHARACTERISTICS

CONTENTS

3.0 AIMS & OBJECTIVES

The aim of this lesson is to introduce the learner the picture clause and the various concepts associated with it. This knowledge will help the user to describe appropriate sizes for the variables he/she intend to use in COBOL programs.

3.1 PICTURE CLAUSE

The picture clause describes the general characteristics of an elementary data item. These characteristics are described below.

3.1.1. Class

In COBOL a data item may be one of the three classes - **numeric, alphabetic or alphanumeric**.

The numeric items consist only of digits 0 to 9.

Alphabetic items consist only of the letters A to Z (a to z) and the space (blank) character.

The alphanumeric items may consist of digits, alphabets as well as special characters.

3.1.2 Sign

A numeric data item can be signed or unsigned.

If a numeric data is considered as unsigned then during execution such unsigned data items are treated as positive quantities.

To describe a signed data item one should use the code character S at the leftmost end of the picture clause of the corresponding variable.

It is important to note that internally, the operational sign (S) is not stored as a separate character. The operational sign is stored at the zone bits of the rightmost digit position of the data item. While preparing data for such an input-signed item, care should be taken to ensure that the data appears on the input medium in the same form.

3.1.3 Point Location

The position of the decimal point is another characteristic that can be specified in the case of numeric data items. If the said position is not specified, the item is considered to be an integer which means that the decimal point is positioned immediately after the rightmost digit. It may be noted that in COBOL the decimal point is not explicitly included in the data. **The position of the decimal point is merely an assumed position**.

The compiler at the time of compilation only makes a note of this assumed decimal point. It generates the object code in such a way that the data items before taking part in the operations are aligned according to their assumed decimal points.

3.1.4 Size

The number of characters or digits required to store the data item in the memory in known as Size of the data item.

All the four general characteristics described above can be specified through a PICTURE clause.

The PICTURE clause is to be followed by a picture character string as shown below.

{ PICTURE / PIC } IS character-string

The character string can consist of 1 to 30 code characters that define the above mentioned attributes of the elementary item. The code characters and their meaning are given below.

Code character	Meaning
9	[Digit] Each occurrence of this code represents a digit
X	[Any character of Cobol] Each occurrence of this code indicates any allowable character from the COBOL character set.
A	[Alphabet] Each occurrence of this code indicates a letter or space character.
V	[Assumed Decimal Point] The occurrence of this in a picture string indicates the position of the assumed decimal point.
P	[Assumed Decimal Point lying outside] The occurrence of this indicates the position of the assumed decimal point when the point lies outside the data item.
S	[Signed Data Item] The occurrence of this indicates that the data item is signed.

There is no special code to indicate the size.

The total number of occurrence of 9, X or A in the picture string indicates the size

The occurrence of V, P and S are not counted in determining the size of an item.

The allowable combinations are governed by the following rules:

(i) In the case of an **alphabetic item** the picture may contain only the symbol **A.**

(ii) In the case of a **numeric item** the picture may contain only the symbols **9, V, P** and **S.** These are called operational characters. It must contain at least one 9.

The symbols V and S can appear only once and S, if it is included, must be the leftmost character of the picture string. The symbol P can be repeated on the right or on left (but not on the left of S) as many times as is required to indicate the position of the assumed decimal point.

(iii) In the case of an **alphanumeric item**, the picture may contain all **X**s or a *combination of 9, A and X* (except all 9 or all A). In the latter case the item is considered as if the string consists of all Xs.

The picture clause is only to be specified for elementary items; it cannot be used for a group item.

The size of a group item is equal to the total of the sizes of all subordinate elementary items. The class of a group item is alphanumeric.

The following examples illustrate the PICTURE specification.

Example 1:

PICTURE IS S999V99

represents a signed data item with a size of 5 characters and the positions of the assumed point is before 2 places from the rightmost end. Note that S and V are not counted.

Example 2:

PIC IS PPP9999

means that the numeric data is of 4 characters in size and there are 7 positions after the assumed decimal point. Thus if the data in the memory is 123, the value will be taken as .0000123. If, on the other hand, the picture were defined as 999PP, the value would have been 12300.

Example 3:

PIC XXXXXX

represents the alphanumeric item with size of 6 characters.

Instead of repeating 9, X, A or P in the picture string, it is possible to write the number of occurrences of a character enclosed within parenthesis immediately after the said character.

Thus

S9(3)V9(2)	is equivalent to S999V99.
X(7)	is equivalent to XXXXXX.
P(4)9(3)	is equivalent to PPPP999.

3.2 LET US SUM UP

In this lesson we have learnt the importance of Picture clause and the different code characters (9,A,X,V,P,S). We have also learnt the size of the data item. This will help you in prescribing the correct declarations for the variables which you will be using in COBOL programs.

3.3 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. What do you mean by Picture Clause?
2. What are the three classes of data item?
3. Highlight the code characters with their meaning.
4. How will you find the size of data items? Give examples.

3.4 POINTS FOR DISCUSSION

1) Explain with example all types of PICTURE clauses used in a COBOL program.

3.5 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 4: EDITING

CONTENTS

4.0 AIMS AND OBJECTIVES

In this lesson, the learner will be introduced the various editing characters of COBOL and their role in editing values The objective here is to make the learner aware of the edit characters and their importance, as they constitute critical phase in getting the results in the edited and required style.

4.1 EDITING

Editing is normally performed by moving a numeric data item to a field containing special editing characters in its PICTURE clause.

4.2 EDIT CHARACTERS FOR NUMERIC DATA

The following characters can be used in the PICTURE clause to indicate editing.

Z * $ - + CR DB . , B O /

The use of these edit characters are explained below.

Z (Zero Suppression)

The edit character Z is used to suppress the leading zeroes in the numeric data item. The leading zeroes are replaced with blanks.

Z character has no effect of trailing zeroes.

Examples

The following examples illustrate the use of Z editing characters. The character ƀ is used to indicate a space character and the character ^ is used to indicate the position of the decimal point.

Picture of the Field	Numeric Value Moved to the Field	Edited Value
ZZ999	01234	~~b~~1234
ZZ999	00052	~~bb~~052
ZZ999	1^68	~~bb~~001
ZZZV99	0^65	~~bbb~~65
ZZZZVZZ	0^05	~~bbbb~~05
ZZZZVZZ	0	~~bbbbbb~~

*** (Asterisk)**

The edit character * (asterisk) is identical to Z except that the leading zeros are replaced by asterisks instead of space characters.

Examples

Picture of the Field	Numeric Value Moved to the Field	Edited Value
**999	01234	*1234
**999	00012	**012
**999	1^23	**001

$ (Currency Sign)

A single currency sign can appear at the leftmost position of a picture. In that case the $ character is inserted.

Examples:

Picture of the Field	Numeric Value Moved to the Field	Edited Value
$99999	123	$00123
$99999	12345	$12345
$ZZ999	123	$~~bb~~123
$ZZ999	12345	$12345
$**999	123	$**123

- (Minus)

A minus sign can appear either at the **leftmost** or **rightmost** position of a picture. If an item is negative, a minus sign will be inserted in the said position. On the other hand, if the item is positive, a space character will be inserted.

Examples:

Picture of the Field	Numeric Value Moved to the Field	Edited Value
-9999	-123	-0123
-9999	382	~~b~~0382
9999-	-123	0123-

9999-	382	0382b
-ZZZV99	-46^52	-b4652
-ZZZV99	46^52	bb4652

+ (Plus Sign)

A plus sign is similar to minus sign except that when the item is positive, +sign will be inserted instead of the space character. If the item happens to be negative, a minus sign will be inserted although there is a plus sign in the picture.

Examples:

Picture of the Field	Numeric Value Moved to the Field	Edited Value
+9999	-123	-0123
+9999	123	+0123
9999+	-123	0123-
9999+	123	0123+
+ZZZV99	-12^34	-b1234
+ZZZV99	12^34	+b1234

CR and DB (Credit and Debit Sign)

The two characters CR or DB symbol may appear **only at the rightmost position** of the picture. They are identical to the minus sign edit character. In other words, the symbols CR or DB will appear in the rightmost position only if the item is negative, otherwise they will be replaced by two space characters.

Examples:

Picture of the Field	Numeric Value Moved to the Field	Edited Value
9999CR	-1234	1234CR
9999CR	1234	1234bb
ZZZCR	-12	b12CR
ZZZ9V99DB	-123^45	b12345DB
ZZZ9V99DB	123^45	b12345bb

. (Period or Decimal Point)

A period may be used to insert a decimal point and may **not appear more than once. Both the period and V cannot appear in the same picture**. A period must not also appear as the rightmost character in the picture.

Examples:

Picture of the Field	Numeric Value Moved to the Field	Edited Value
9999.99	324^52	0324.52
ZZ99.99	12^34	bb12.34

, (Comma)

A comma, when used in a picture, is treated as an insertion character and inserted wherever it appears. There can be more than one comma in a picture.

Examples:

Picture of the Field	Numeric Value Moved to the Field	Edited Value
99,999	1234	01,234
99,999	37	00,037
ZZ, Z99	1234	~~b~~1,234

B (Blank Insertion)

The appearance of a B anywhere in the picture will insert a space character in the edited data. There can be more than one B in a picture.

Examples:

Picture of the Field	Numeric Value Moved to the Field	Edited Value
99B99B99	171062	17~~b~~10~~b~~62
99B99B99	12	00~~b~~00~~b~~12

0 (Zero Insertion)

A zero appearing in a picture will be treated in the same way as a B except that 0 will be inserted instead of a space character.

Examples:

Picture of the Field	Numeric Value Moved to the Field	Edited Value
9900	12	1200
09990	456	04560

/ (Slash Insertion)

The edit character slash (/) also called virgule or stroke, may appear anywhere in the picture. If used, it will be inserted. There can be more than one slash in the picture.

Examples:

Picture of the Field	Numeric Value Moved to the Field	Edited Value
99/99/99	150681	15/06/81
999/999/99	3245	000/032/45

BLANK WHEN ZERO

BLANK WHEN ZERO is an editing clause which may be used along with a picture. This will set the entire data item to blanks of its value is equal to zero. However, the edit character asterisk (*) may not be used if BLANK WHEN ZERO is specified. When this clause is used to describe a field whose picture contains an asterisk, it is ignored by the compiler. The syntax of this clause is as follows:

BLANK WHEN ZERO

Examples

Picture of the Field	Numeric Value Moved to the Field	Edited Value
ZZZ.99 BLANK WHEN ZERO	2^5	~~bb~~2.50
ZZZ.99 BLANK WHEN ZERO	0	~~bbbbbb~~
999.99 BLANK WHEN ZERO	0	~~bbbbbb~~

Floating Insertion

The currency symbol ($) can appear in multiples on the left-hand side of a picture. In this case the character will be treated in the same way as the Z character and only one currency symbol will be inserted immediately to the left of the first non-zero digit of the data. Such a floating insertion is also possible in the case of minus (-) and plus (+) signs. In the case of the minus character, no sign will be inserted unless the data is negative. The appearance of a period halts the floating insertion.

Examples

Picture of the Field	Numeric Value Moved to the Field	Edited Value
$$$$9.99	235^25	~~b~~$235.25
++++.99	-475^25	-475.25
----.99	-3^5	~~bb~~-3.50

4.3 LET US SUM UP

In this lesson all the editing characters are introduced and their role can be understood with the help of the examples presented. The Edit characters are very important in programs to get correct and required form of output.

4.4 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. What is meant by Editing?
2. List out the edit characters for numeric data.
3. Explain Z * $ edit characters with examples
4. Explain + - CR DB edit characters with examples
5. Explain B 0 / edit characters with examples

4.4 POINTS FOR DISCUSSION

1) Discuss in detail about the editing picture clause of COBOL.

4.5 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 5: LEVEL STRUCTURE

CONTENTS

5.0 AIMS AND OBJECTIVES

In the present lesson the learner gets an idea about level numbers , Value clause and Filler clause. A sample program is presented in the lesson to demonstrate the concept of value clause and level numbers.

5.1 LEVEL NUMBERS

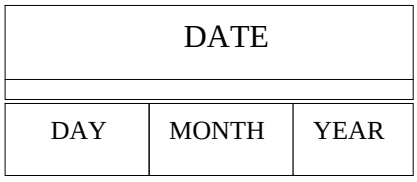
COBOL makes use of level number concept to describe the hierarchical data structure. A level number is a 2 digit number. The allowed level numbers are 01 – 49, 66, 77 and 88.

All variables are declared in the working-storage section using appropriate level numbers.

An elementary data item can take either 01 or 77 as its level number.

In COBOL a distinction is made between elementary and group data items. A few elementary data may be combined to form a group.

For example, DAY, MONTH and YEAR may be three elementary data items. These may be combined to form a group data named DATE. The organization may be shown pictorially as follows:



It may be noted that the memory space referred to by DATE is the combined memory space for DAY, MONTH and YEAR. The advantage of such a grouping is obvious. The programmer can now refer to the individual elementary items DAY, MONTH, YEAR or to the group item DATE. An elementary data item is thus the one which the programmer would always like to refer to as a whole and not in parts.

To describe the hierarchical structure introduced above, the concept of level number is employed in COBOL.

The most inclusive group must have the level number 01. The first subdivisions can have any level number between 02 and 49. Further subdivisions should follow the same range with the restriction that an item cannot have a level number less than or equal to the level numbers of the group that may include it. Thus a group includes all elementary data or smaller groups beneath it until a level number equal to or less than the level number of the said group is encountered. The following examples reveal the concept of the level numbers.

Example 1

```
01    DATE
05    DAY
05    MONTH
05    YEAR
```

Example 2

```
01    PAY
02    GROSS-PAY
03    BASIC
03    DEARNESS
03    HOUSE-RENT
02    DEDUCTIONS
03    PF-DEDUCT
03    IT-DEDUCT
02    NET-PAY
```

The group DATE which was shown earlier in the pictorial form is illustrated in the first example. In the second example PAY is the most inclusive group which has three subdivisions, namely, GROSS-PAY, DEDUCTIONS and NET-PAY. GROSS-PAY is again subdivided into BASIC, DEARNESS and HOUSE-RENT. In a similar way deductions are further subdivided into PF-DEDUCT and IT-DEDUCT. It may also be noted that the elementary data items are BASIC, DEARNESS, HOUSE-RENT, PF-DEDUCT, IT-DEDUCT and NET-PAY. The structure can be pictorially shown as follows:

PAY

GROSS – PAY			DEDUCTIONS		NET - PAY
BASIC	DEARNESS	HOUSE - RENT	PF - DEDUCT	IT - DEDUCT	

Sometimes, in a hierarchical data structure such as this, the programmer may not require a data item to be referred to in the PROCEDURE DIVISION. Such a situation usually arises when a group and only some of its subdivisions are to be used in the program. The remaining subdivisions need not be used explicitly. In such situations the word FILLER may be used to

name data to which the programmer does not wish to assign a specific name. FILLER can be used as many times as required.

5.2 VALUE CLAUSE

The value clause defines the initial value of the data item.

Generally initialization will be done just before the first statement in the procedure division is executed.

The syntax is

VALUE is literal

The literal can be any numeric value, a nonnumeric string of characters included within quote(“) or any figurative constant.

Examples :

1) 01 a pic value is 100
2) 01 compname pic x(15) value is “ABC Company”
3) 01 n pic 9(2) value is ZERO
4) 01 ans pic x value is space
5) 01 result pic x(4) value spaces.
6) For Group Data value specification

```
01 test-entry  value is “123456”.
02  t1      pic 9(2).
02  t2      pic 9(2).
02  t3      pic 9(2).
```

Here t1=12,t2=34 and t3=56.

5.3 SAMPLE PROGRAM

Write a program to demonstrate value clause.

```
Identification division.
Program-id. Valcls.
Environment division.
Data division.
Working-storage section.
01       name pic x(4) value “ABCD”.
02       mark  pic 9(3) value 100.
Procedure division.
p-1.

display(1 1) erase.
Display(3 5) “Given Name = “ name.
Display(5 5) “ Mark =” mark.
Stop run.
```

Explanation:

Note that in the above program, initial values are given by the programmer for name and mark. They will be displayed as such as a result of execution.

In the program, the user is at liberty to change the initial values by using statements like MOVE, ACCEPT etc.

5.4 FILLER CLAUSE:

Consider the statements given below:

1) 01 f pic x(80) value all "-".
This statement causes a line of 80 characters filled with "-"
2) 01 f pic x(60) value all "*".
This statement causes a line of 60 characters filled with "*"
3) 01 filler pic x(10) value "TESTING".
Note that we can either use simply "f" or "filler" in the statements.

Generally fillers are used to improve the clarity of the output and form designs utilize the potential of filler clauses to the maximum.

5.5 LET US SUM UP

In the above lesson the concept of level numbers, value clause and Filler clause are discussed with syntaxes and examples and a sample program is written to emphasize these concepts.

5.6 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. What do you mean by Level Numbers?
2. List the permitted level numbers in COBOL
3. Specify the level numbers for elementary data.
4. Specify the level numbers for group data.
5. Explain the role of value clause in COBOL.

5.7 POINTS FOR DISCUSSION

1) What are level numbers? Explain their usage with examples.

2) Write notes on

a) Elementary Data Item.

b) Group Data Item.

5.8 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

UNIT II

LESSON – 6: DATA MOVEMENT VERB: MOVE

CONTENTS

6.0 AIMS AND OBJECTIVES

The aim of this lesson is to introduce the learner how to move data from one place to another place in memory. This is done with the help of MOVE verb. The syntax, rules and the examples given will make the learner to understand the said verb.

6.1 SYNTAX OF MOVE VERB

The general form of the MOVE verb is as follows:

MOVE { identifier - 1 / literal - 1 } TO identifier – 2 [, identifier – 3] …

6.2 RULES OF MOVE VERB

Data movement is governed by the following rules.

(a) The contents of identifier – 1 or the value of literal – 1 is moved to identifier – 2, identifier – 3, etc. Note that there may be more than one receiving field whereas there must be only one sending field, the contents of all the receiving fields will be replaced by the value of the sending field. The contents of identifier – 1 remain unaltered.

(b) When the sending field is numeric and the receiving field is numeric or numeric edited (i.e., picture contains edit symbols) the data movement is called numeric data transfer. In such cases the dominant factor in the movement is the alignment of the decimal points of the two fields. For the purpose of this alignment, the numeric fields for which the position of the decimal point is not explicitly indicated, the decimal point is assumed to be at the right of the rightmost digit. If the receiving field is not large enough to hold the data received, truncation can take place at either and depending on whether the integral part, fractional part or both can or cannot be accommodated (see examples given in this section for further clarification). However, if significant integral positions are likely to be lost, a warning to that effect is issued by the compiler. On the other hand, if the receiving field is larger than the sending

field, zero-fill will take place in the unused positions to keep the numeric value unaltered.

(c) When both the sending and receiving fields are alphabetic, alphanumeric or alphanumeric edited, the data movement is called alphanumeric data transfer. In such cases the receiving area is filled from left to right and space fill occurs to the right if the receiving area is larger than the sending field. When the receiving area is smaller, truncation occurs from the right and the compiler gives a warning to that effect.

Ideally, both the sending and receiving fields should belong to the same category. However, quite often it becomes necessary to transfer a data to a field having a different category. Identifier – 1, identifier – 2, identifier – 3, etc., can be group items. In such cases, the move is very frequently used. This is when we wish to initialize a record area by spaces. For example, the statement MOVE SPACES TO REC-AREA will space-fill the entire area denoted by the group name REC-AREA.

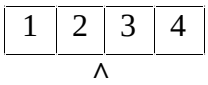
6.3 EXAMPLES OF MOVE USAGE

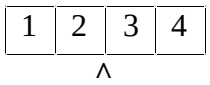
(a) MOVE A TO B.

	Contents of A		Contents of B	
	Before execution	After execution	Before execution	After execution
(i) PIC 9999	5 6 7 8	5 6 7 8	1 2 3 4	5 6 7 8
(ii) PIC 999	5 6 7	5 6 7	PIC 9999 1 2 3 4	0 5 6 7 Zero fill on the the left
(iii) PIC 99V9	1 2 3 ^	1 2 3 ^	PIC 999V99 5 6 7 8 2 ^	0 1 2 3 0 ^ Zero fill on left and right

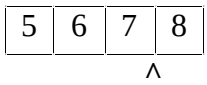
	Contents of A		Contents of B	
	Before execution	After execution	Before execution	After execution

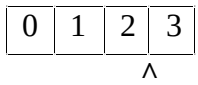
(iv) PIC 99V99





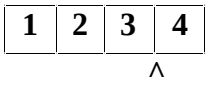
PIC 999V9

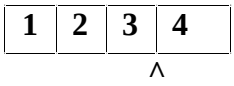




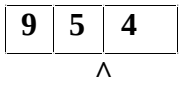
Zero fill on Left and truncation on right

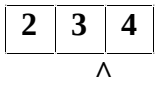
(v) PIC 999V9





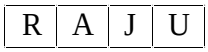
PIC 99V9

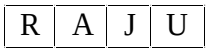




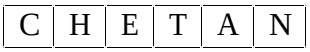
Truncation on left

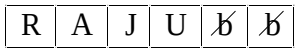
(vi) PIC X(4)





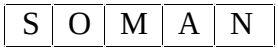
PIC X(6)

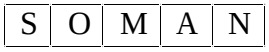




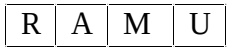
Space fill on right

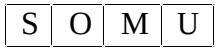
(vii) PIC X(5)





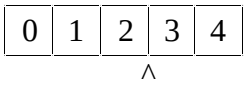
PIC X(4)

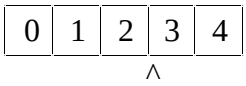




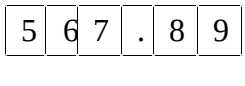
Truncation on right

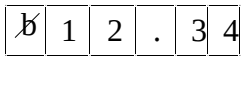
(viii) PIC 999V99





PIC ZZ9.99





Decimal point alignment and editing

(b) MOVE 15 TO A.

In this case the number 15 will be moved to A and if the PICTURE of A is 999, then after the execution of the above statement A will contain 015.

(c) MOVE “THERE IS AN ERROR” TO A.

From quotes to quotes the total number of characters including space is 17. since this is a nonnumeric literal, all the 17 characters will be moved to A from left to right if the PICTURE of A is X(17).

(d) MOVE A TO B, C, D.

If the contents of A is 22 and the contents of B, C and D are 452, 3892 and 46 respectively, then after the execution of the above instruction the contents of B, C and D will be 022, 0022 and 22 respectively.

The above mentioned rules for data movement are also used elsewhere in COBOL. For example, when the value of a data item is initialized by using the VALUE clause, the same rules apply. In this case the data name should be considered as the receiving field and the value of the literal should be taken as that of the sending field. Thus

77 NEW-DATA PIC X(10) VALUE "NEWDATA"

will initialize NEW-DATA by the value NEWDATA bbb.

6.4 LET US SUM UP

In this lesson clear ideas about the MOVE verb are highlighted. One getsstrong background of the concept by understanding the how the values are placed while moving takes place from one variable to other. This verb will be used in majority of the programs.

6.5 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Specify the syntax of Move Verb
2. State the rules of Move Verb
3. Give examples to show how Move statement works.

6.6 POINTS FOR DISCUSSION

1) Discuss the usage of move verb with exempt.
2) Discuss the rules for Data movement.

6.7 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 7: ARITHMETIC VERBS

CONTENTS

7.0 AIMS & OBJECTIVES

Most of the problems require some computations to be performed on the input or intermediate data which are numeric in nature. Arithmetic verbs are used to perform these computations. All these verbs can contain either identifiers or numeric literals or both. In the case of identifiers, they must be elementary numeric fields, and identifiers used after GIVING option must be edited or unedited numeric fields. Some arithmetic verbs in their most elementary forms are discussed below.

7.1 ADD VERB

This verb can be used to find the sum of two or more numbers and to store the sum. The ADD verb takes any one of the following two forms.

```
ADD { identifier-1 }                 ( identifier-2 )
    { literal - 1  }                 (, literal-2   )   …

      TO    identifier-3   [, identifier-4]  …
```

```
ADD { identifier-1 }  { identifier-2 }  ( identifier-3 )
    { literal-1    }  { literal-2    }  (, literal-3   )

      GIVING   identifier-4   [, identifier-5]  …
```

Examples

(a) ADD A TO B.

This example shows that the value of A will be added to the value of B and the result will be stored in B. The alignment of the decimal point is done automatically.

(b) ADD A B C TO D.

In this case the values of A, B and C will be added to the old value of D and the resultant sum will be the new value of D.

(c) ADD 30 A TO B.

This example shows that the number 30, the value of A and the value of B will be added and the resultant sum will be stored in B.

(d) ADD A, B GIVING C.

Here only the values of A and B will be added and the sum will be stored in C. The old value of C will be lost and that value will not take part in the summation.

(e) ADD A, B GIVING C, D, E.

In this case the value of A, B will be added and the sum will be stored in C, D and E. Hence after the execution of this statement, C, D and E will have the same value.

The above examples indicate that in the case of the TO option the previous value of the last named operand takes part in the summation and then this value is replaced by the result. However, this is not the case when the GIVING option is used. It should be mentioned here that the last named operand in both the cases can never be a literal as the resultant sum is always stored there.

It is important to note that TO and GIVING cannot be used simultaneously. Thus ADD A TO B GIVING C would be wrong. The purpose is served by specifying as ADD A B GIVING C. With GIVING option identifier-2/numeric-literal-2 is a must.

7.2 SUBTRACT VERB

This verb is used to subtract one, or the sum of two or more numbers from one or more numbers and to store the result.

The form of the SUBTRACT verb is as follows:

```
SUBTRACT { identifier-1 }          [ , { indentifier-2 } ] ...  FROM
         { numeric-     }          [   { numeric-     } ]
         { literal-1    }          [   { literal-2    } ]

    identifier-3          [, identifier-6] ...

[ , GIVING identifier-5   [ , identifier-6 ] ... ]
```

Examples

(a) SUBTRACT A FROM B.

This statement means that the value of A will be subtracted from the value of B and the subtracted result will be stored in B. The decimal point alignment will be done automatically.

(b) SUBTRACT A, B FROM C.

This one shows that the value of B and A will be added and the resultant sum will be subtracted from the value of C. After subtraction, the final result will be stored in C. The old value of C will be lost.

(c) SUBTRACT A, B FROM C GIVING D.

This indicates that the summation of the value of A and B will be subtracted from the value of C and the final result will be stored in D. The old value of D will be lost whereas in this case C retains the old value.

(d) SUBTRACT 15 FROM A B.

Here the number 15 will be subtracted from the values of A and B. A and B will receive these new values.

As in the case of the ADD statement, here also the last-named operand must not be a literal as the final result will be stored there. If the GIVING option is used, identifier-3, identifier-4 etc. can also be numeric literals. For example, SUBTRACT A B FROM 50 GIVING C.

7.3 MULTIPLY VERB

This statement causes one or more multiplicands to be multiplied by a multiplier and to store the products. The form of the MULTIPLY verb is as follows:

```
MULTIPLY { identifier-1       } BY identifier-2 [ , identifier-3 ] ...
         { numeric-literal-1  }

, [ GIVING identifier-4 [ , identifier-5 ] ... ]
```

Examples

(a) MULTIPLY A BY B.

In this case the value of A and B will be multiplied and the product will be stored in B. The decimal point position will automatically be taken care of. The old value of B will be lost.

(b) MULTIPLY A BY B GIVING C.

Here the value of A and B will be multiplied and the product will be stored in C. The old value of C will be lost but B will contain its old value.

(c) MULTIPLY A BY B C D.

Here B will be multiplied by A and the result will be stored in B. similarly, C will be multiplied by A and the product will be stored in C and the result of the multiplication of D and A will be stored in D.

(d) MULTIPLY A BY B C GIVING D E.

In this case the product of B and A will be stored in D, whereas the product of C and A will be stored in E.

In the case of the MULTIPLY statement also, literals cannot be used for identifier-2, identifier-3, etc. However, if the GIVING option is used, numeric literals are also permitted in place of identifier-2, identifier-3, etc. For example,

```
MULTIPLY TAX BY .05 GIVING TAX-BASE.
```

7.4 DIVIDE VERB

The purpose of this verb is to divide one number by another and to store the result. There are several forms of this verb. One of its forms is as follows:

DIVIDE { identifier-1 / numeric-literal-1 } INTO identifier-2 , [identifier-3 …]

[, GIVING identifier-4 [, identifier-5] …]

Examples

(a) DIVIDE 5 INTO A.

If the value of A is 20, then after the execution of this statement the value of A will be 4. The old value of A will be lost.

(b) DIVIDE 5 INTO A GIVING B.

If the value of A is 20, then after the execution of this statement the value of B will be 4. Here A will retain its old value.

(c) DIVIDE 3 INTO A GIVING B C.

Here the result of the division of A by 3 will be stored both in B and C.

(d) DIVIDE 2.5 INTO A B GIVING C D.

In this case A will be divided by 2.5 and the result will be stored in C, whereas the result of the division of B by 2.5 will be stored in D.

As in the case of the MULTIPLY statement, literals cannot be used for identifier-2, identifier-3, etc. Only when the GIVING option is used the numeric literals permitted in place of identifier-2, identifier-3, etc. For example, DIVIDE A INTO 25 GIVING V.

The second form of this verb is as follows:

DIVIDE { identifier-1 / numeric-literal-1 } BY { identifier-2 / numeric-literal-2 }

GIVING identifier-3 [, identifier-4] …

In this case identifier-1 or numeric-literal-1 will be divided by identifier-2 or numeric-literal-2, whatever may be the case. The result is stored in identifier-3, identifier-4, etc.

Examples

DIVIDE A BY 3 GIVING C.

If the value of A is 21 then after the execution of this statement C will contain 7.

There is another form of DIVIDE verb where there is a provision to store the remainder. Its form is

DIVIDE { identifier-1 / numeric-literal-1 } { INTO / BY } { identifier-2 / numeric-literal-2 }

GIVING identifier-3 [REMAINDER identifier-4.]

Example

DIVIDE A INTO B GIVING C REMAINDER D.

If the identifier A, B, C and D are all two-digited numbers and if they contain 05, 37, 18 and 20 respectively before the execution of the statement, then after the execution of the statement, they will contain 05, 37, 07 and 02 respectively.

7.5 COMPUTE VERB

COMPUTE verb is so powerful. All the computations performed by the other four verbs can also be done easily by using the COMPUTE verb. Its general format is

COMPUTE identifier-1[ROUNDED],[identifier-2 ROUNDED] …

= arithmetic-expression[; ON SIZE ERROR imperative-statement]

The COMPUTE statement has the following meaning. During execution the arithmetic expression on the right of the equal sign is evaluated and the value is then moved to the identifiers(s) on the left-hand side. If any identifier on the left of the equal sign is a numeric-edited item, editing takes place when the value of the expression is moved to the said identifier. The identifiers on the left of the equal sign (=) must be numeric or numeric-edited elementary items. The right-hand side must be an arithmetic expression. An arithmetic expression can be an identifier (numeric elementary items only), a numeric literal or can specify a computation involving two or more such identifiers and/or literals. An arithmetic expression has always a numeric value. The following are the rules for constructing arithmetic expression.

(i) When an arithmetic expression specifies a computation, it may consist of two or more numeric literals and/or data names joined by arithmetic operators. The following table lists the operations and their meaning.

Operator	Meaning
**	Exponentiation
/	Division
*	Multiplication
-	Subtraction
+	Addition

There must be at least one space preceding and following the operator in an arithmetic expression. No two arithmetic operators can appear together in an expression. In this respect ** is considered to be a single operator.

(i) Parentheses may be used to specify the order of operations in an arithmetic expression. Where parentheses are absent, the order is taken to be left to right as follows:

**	Exponentiation
/ *	Division and Multiplication
- +	Subtraction and Addition

When parentheses are used, the portion of the expression enclosed within parentheses is evaluated first.

(ii) An arithmetic expression may be proceeded by a + or – sign. Such operations are called unary + or unary – operators.

Examples of valid arithmetic expressions are

```
3          *      I
RATE             *      QUANTITY   -    DISCOUNT
```

Note that when the right-hand side of a COMPUTE verbs is a single identifier or literal, the effect is that of a MOVE statement.

Example 1

```
COMPUTE   A = B + C
```

has the same effect as that as that of ADD B C GIVING A

Example 2

```
COMPUTE    F =  1.8 * C + 32
```

The value of the expression on the right-hand side is evaluated and this value is then moved to F. Suppose C and F are defined with pictures 99 and ZZ9.9 respectively and the current value of C is 3. Then after the execution of the statement, F will have the value 37.4.

7.6 LET US SUM UP

This lesson has taught the learner to understand the arithmetic verbs like Add, Subtract, Multiply, Divide and Compute. All these verbs have been presented with syntaxes and adequate examples are also provided.

7.7 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Explain with syntax ADD verb.
2. Explain with syntax SUBTRACT verb.
3. Explain with syntax MULTIPLY verb.
4. Explain with syntax DIVIDE verb.
5. Explain with syntax COMPUTE verb.

7.9 POINTS FOR DISCUSSION

1) Explain the usage of the followings.

a) ADD verb b) SUBTRACT verb

2) Write notes on

a) MULTIPLY verb b) DIVIDE verb.

3) Discuss the usage of COMPUTER verb.

7.8 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 8: INPUT AND OUTPUT VERBS

CONTENTS

8.0 AIM AND OBJECTIVES

Reading the data into the memory from some input medium (such as punched cards) and writing the results from the memory onto some output medium (such as continuous stationary) are of basic importance. The verbs OPEN,READ,WRITE and CLOSE are available for such input-output operations.

8.1 OPEN

When a READ or a WRITE operation is performed on a file, it must be open. The opening of a file may be done with the help of the OPEN verb. With the OPEN verb it must be also indicated whether the file should be opened as an input file or output file. If it is an input file, only reading is possible, whereas in the case of an output file, only writing is possible. A file once opened remains open until it is closed by a CLOSE statement.

The OPEN statement in its simple form is as follows:

```
        { INPUT  file-name-1 [ , file-name-2 ] ... }
OPEN    {                                          }
        { OUTPUT file-name-3 [ , file-name-4 ]     }
```

Example 1

OPEN INPUT TRANSACTION, OLD-MASTER OUTPUT NEW-MASTER.

The example shows that there are two input files named TRANSACTION and OLD-MASTER and one output file called NEW-MASTER. All these files are opened and these are ready for reading or writing.

Example 2

```
OPEN INPUT MARK-FILE.
OPEN OUTPUT RESULT-FILE.
```

The first OPEN statement opens the MARK-FILE in input mode and the file is ready for reading. The next statement makes the RESULT-FILE ready for writing. There may be several OPEN statements in a program.

8.2 READ

The purpose of this verb is to make available the next logical record from an input file. It is important to note the meaning of "next" logical record in the above statement. The first time the READ statement is executed, the first record of the file will be read into the record area described in the FILE SECTION of the DATA DIVISION. The next time the READ statement is executed, the second record will be read in the same area. In this way each time a READ statement is executed the successive records will be read in the same area. Thus a time will come when there will be no more records in the file. In that case the statements following the AT END clause will be executed. The format of the READ statement is

```
READ  file-name RECORD  [INTO identifier-1]
      AT END  imperative-statement
```

Example 1

```
READ OLD-MASTER AT END MOVE ZERO TO END-OF-RECORDS.
```

As a result of this statement, normally the next record from the OLD-MASTER file will be read. If there is no more record in OLD-MASTER, the value zero will be moved to the field named END-OF-RECORDS.

Example 2

```
READ TRANSACTION RECORD AT END GO TO PARA-END.
```

This example is similar to the earlier example. The next record from the TRANSACTION file will be read if it is available. If the file does not contain any more records, the control will be transferred to the paragraph named PARA-END.

Example 3

```
READ KARD-FILE INTO IN-REC AT END
                    GO TO JOB-END.
```

This statement not only reads the next record into the record area of KARD-FILE but also moves the record into the area name IN-REC. When there is no more record in the KARD-FILE, the control is transferred to the paragraph named JOB-END. If the record area of the KARD-FILE has been named KARD-REC, the above statement is equivalent to

```
READ KARD-FILE AT END GO TO JOB-END.
MOVE KARD-REC TO IN-REC.
```

It may be noted that if the record has been successfully read, it is now available in KARD-REC as well as IN-REC.

8.3 WRITE

The WRITE verb releases a record onto an output file. The syntax of the WRITE statement can be different depending on the output device and medium used. The verb as

described here can be used only to print results on a continuous stationery through a line printer. The form of the WRITE statement in such a case is

WRITE record-name [FROM identifier-1]

({ BEFORE / AFTER } ADVANCING { { integer-1 / identifier-2 } { LINES / LINES } / { mnemonic-name / hardware-name } })

The first point to be noted is that in the case of the READ statement the file name is to be specified, whereas in the case of the WRITE statement it is required to mention the record name an not the file name. The ADVANCING phrase is used to control the vertical positioning of each record at the time of printing on the stationery placed on the printer. When the BEFORE phrase is used, the record is printed before the stationary is advanced, whereas the AFTER phrase may be used when the intention is to advance the stationary first and then to print the record. If integer-1 or identifier-1 is mentioned, the stationary is advanced by the number of lines equal to the value of integers-1 or to the current value of identifier-1.

If the mnemonic-name is specified, the printer will be advanced to the carriage control channel declared for the mnemonic-name in the SPECIAL-NAMES paragraph. This option is provided so that the hardware names which may be peculiar to a particular computer need not appear in the PROCEDURE DIVISION.

If the FROM option is used, the operation is identical to that of MOVE identifier-1 TO record-name followed by a WRITE record-name without the FROM clause. It is illegal to use the same storage area for both record-name and identifier-1.

It should be noted that after WRITE is executed the record is no longer available.

Examples

(i) WRITE TRANS-RECORD AFTER ADVANCING 3 LINES.

This WRITE statement indicated that TRANS-RECORD is a record name of a file that has been assigned to PRINTER. The current position of the stationery will be advanced by 3 lines, i.e., there will be 2 blank lines and the present record will be written on the third line.

(ii) WRITE TRANS-RECORD BEFORE ADVANCING 3 LINES.

The record will be written first and then the page will be advanced by 3 lines.

8.4 CLOSE

When the processing of a file is over, the file may be closed. This is done with the help of the CLOSE-verb. The form of the CLOSE statement is

CLOSE file-name-1 [, file-name-2] …

The file must be open when a close statement can be executed. Once a file is closed, it is no longer available to the program. It should be opened again if the file is required subsequently. It may be noted that unlike the OPEN statement, the nature of the use of the file (input and output) should not be mentioned in the CLOSE statement.

Example

CLOSE TRANSACTION, OLD-MASTER, NEW-MASTER, PRINT-FILE.

This statement will close all the four files – TRANSACTION, OLD-MASTER, NEW-MASTER and PRINT-FILE.

8.5 ACCEPT

The ACCEPT statement is used to read low-volume data from the operator's console, some other hardware device or from the operating system. The general format of the ACCEPT statement is as follows:

When the FROM option is omitted, the data is read into the identifier from the operator's console. At the time of execution, a message is displayed on the console (e.g., AWAITING COBOL INPUT) and the program is suspended until the operator enters the data through the console keyboard. Data entered by the operator will be left justified in the identifier. For example,

ACCEPT FLAG-A

can be specified to read the value of FLAG_A fro the console. It may be noted that no file defi0nition is necessary.

The mnemonic-name option is implementer-dependent. The hardware device from which the data is to be read is to be equated to a mnemonic name in the SPECIAL-NAME APARAGRAPH. For example, the following entry in the SPECIAL-NAMES paragraph

TYPEWRITER-1 IS CONTROL-DATA

may equate the mnemonic name CONTROL-DATA with the assumed implementor-name TYPEWRITER-1. The

ACCEPT FLAG-1 FROM CONTROL-DATA

will read the value of FLAG-A fro the hardware device indicated by TYPEWRITER-1.

The DATE, DAY and TIME options are new features introduced in ANSI 74 COBOL. The DAY option returns the six-digit current date in the form YYMMDD where YY, MM and DD stand for year, month and day respectively. The DAY option returns a five-digit current date in the form YYDDD where YY stands for the year and DDD stands for the day of the year(001 to 365). The TIME option returns an eight-digit time in the form HHMMSSTT where HH, MM, SS, TT represent hour, minute, second and hundreds of a second respectively. For all the three options, the returned value is transferred to the identifier (in the ACCEPT statement) according to the rules of the MOVE statement. For example,

ACCEPT THIS-DAY FROM DATE

will transfer the value of the current date to THIS-DAY.

8.6 DISPLAY

The function of the DISPLAY statement is opposite to that of the ACCEPT statement. It is used to display low-volume results on the operator's console or some other hardware device. The general format of the DISPLAY statement is

DISPLAY { identifier-1 / literal-1 } [, identifier-2 / , literal-2] [… UPON mnemonic-name]

If the UPON option is omitted, the values of the identifier-1/literal-1, identifier-2/literal-2, etc. are displayed on the console. The UPON option with the mnemonic-name is implementor-dependent. The mnemonic name, as in the case of the ACCEPT verb, should be defined in the SPECIAL-NAMES paragraph. When more than one operand is specified, the values of the operands are displayed in the sequence they are specified. There will be no space between these values. The operands must be of the usage DISPLAY. The literals may be any literal or figurative constant except the ALL literal. If a figurative constant is specified, only one occurrence of the constant is displayed.

Example

Consider the following statement.

DISPLAY " SUM IS", THE-SUM

and suppose that the picture and current value of THE-SUM are 9(3) and 15 respectively. Upon execution of the statement, the following will be displayed on the console

SUM IS 015

8.7 GO TO , STOP RUN

GO TO

GO TO verb is used to unconditionally transfer the control to elsewhere in the program. Its form is as follows:

GO TO procedure-name

As a result of the execution of this statement, the control is transferred to the first statement of the paragraph or section mentioned in the procedure name.

Example

GO TO ERROR-ROUTINE

Suppose ERROR-ROUTINE is a paragraph name. The execution of this statement will transfer the control to the first statement in ERROR-ROUTINE. On the other hand, suppose ERROR-ROUTINE is a section name and FIRST-PARA is the name of the first paragraph in this section. In the case control will be transferred to the first statement in FIRST-PARA. It may be noted that GO TO FIRST-PARA is identical to GO TO ERROR-ROUTINE.

STOP RUN

This verb causes the termination of the execution of the object program. Its form is

STOP RUN

8.8 LET US SUM UP

In the lesson the learner is exposed to the Input-output Verbs like Open, Close, Read, Write, Accept, Display etc. All of these will be useful in majority of the programs. They are self-explanatory in nature by their names.

8.9 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Explain with syntax OPEN statement.
2. Explain with syntax READ statement.
3. Explain with syntax WRITE statement.
4. Explain with syntax CLOSE statement.
5. Explain with syntax ACCEPT & DISPLAY statements.

8.10 POINTS FOR DISCUSSION

1) Explain the usage of OPEN verb with syntax and examples.

2) Write short notes on READ verb.

3) Briefly explain about WRITE verb.

4) Write notes on ACCEPT verb.

8.11 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 9: CORRESPONDING OPTIONS

CONTENTS

9.0 AIMS AND OBJECTIVES

The aim of this lesson is to introduce the learner to get ideas about various corresponding options , rounded option and size error option.

9.1 MOVE CORRESPONDING

Quite often it is required to move some of the data items of one group to some other data items in another group. If the names of the corresponding data items of the two groups are distinct, then for each data item, a separate MOVE verb should be used. However, if the corresponding data items of both the records have identical names, then instead of using separate MOVE statement, just one MOVE statement with the CORRESPONDING option can be used. The following example illustrates the use of the MOVE verb with the CORRESPONDING option.

Example

Consider the following DATA DIVISION entries.

```
02      PAY-REC.
          02      ID-NUMBER          PIC     9[5].
          02      NAME               PIC     X[25].
          02      DEPARTMENT         PIC     X[20].
          02      BASIC-PAY          PIC     9999V99.
          02      FILLER             PIC     X[24].

01      PRINT-REC.
          02      FILLER             PIC     X[5].
          02      ID-NUMBER          PIC     Z[5].
          02      FILLER             PIC     X[5].
          02      NAME               PIC     X[25].
          02      FILLER             PIC     X[5].
          02      DEPARTMENT         PIC     X[20].
          02      FILLER             PIC     X[5].
          02      BASIC-PAY          PIC     ZZZZ .99.
```

```
02      FILLER          PIC     X[5].
02      DEDUCTIONS      PIC     ZZZZ .99.
02      FILLER          PIC     X[5].
02      ALLOWANCES      PIC     ZZZZ .99.
02      FILLER          PIC     X[5].
02      NET-PAY         PIC     ZZZZ. .99.
```

Suppose it is required that the data stored in the four fields of PAY_REC should be moved to those fields of PRINT_REC that are given the same data names. The following four MOVE statements can serve the purpose.

```
MOVE ID-NUMBER OF PAY-REC TO ID-NUMBER OF PRINT-REC.
MOVE NAME OF PAY-REC TO NAME OF PRINT-REC.
MOVE DEPARTMENT OF PAY-REC T0 DEPARTMENT OF PRINT-REC.
MOVE BASIC-PAY OF PAY-REC TO BASIC-PAY OF PRINT-REC.
```

However, since both the records have same names for the concerned data items, the following statement

```
MOVE CORRESPODING PAY-REC TO PRINT-REC.
```

will have the same effect. It is not necessary that the corresponding data names in the two records should appear in the same order. The general format of the MOVE CORRESPONDING statement is

```
MOVE { CORRESPONDING } identifier-1  TO  identifier-2
     { CORR         }
```

where dentifier-1 and identifier-2 should be group names. Note that MOVE CORRESPONDING is not a group move, it is merely a means for specifying a number of elementary moves through a single MOVE statement. As such any editing, if specified, will be performed. Source and destination groups can include data names that are not common. Only those fields having identical names in the two records will take part in the data movement. The remaining data items in the destination group will remain unchanged.

9.2 ADD and SUBTRACT CORRESPONDING

The CORRESPONDING option can also be used with the ADD and SUBTRACT verbs. The following are the formats of these verbs with the CORRESPONDING option.

```
ADD { CORRESPONDING } identifier-1  TO  identifier-2
    { CORR         }
```

```
SUBTRACT { CORRESPONDING } identifier-1  TO  identifier-2
         { CORR         }
```

In the case of the ADD statement numeric elementary items in the group referred to by identifier-1 are added to and stored in the corresponding elementary items of the group named in identifier-2. In the case of the SUBTRACT statement, the corresponding numeric elementary items of the group referred to by identifier-1 are subtracted from and are stored in the corresponding numeric elementary items of the group referred to by identifier-2.

9.3 GENERAL RULES CONCERNING CORRESPONDING OPTION

The following rules should be observed when the CORRESPONDING option is used.

(i) Identifier-1 and identifier-2 in all cases must refer to group items i.e., these identifiers must not be data items with level numbers 66, 77 or 88.

(ii) Data items in identifier-1 and identifier-2 take part in the specified operation (MOVE, ADD or SUBTRACT) only when they have the same data name and same qualifiers up to but not including identifier-1 and identifier-2.

(iii) In the case of ADD or SUBTRACT CORRESPONDING only numeric data items are considered for addition or subtraction respectively. This means that data items other than numeric are not considered for the arithmetic operations even though they may have identical names in the two groups named in identifier-1 and identifier-2.

(iv) All data items subordinate to identifier-1 and identifier-2 with level numbers 66 or 88 or containing a REDEFINES or OCCURS clause, are ignored for the purpose of the operation. Identifier-1 and identifier-2 may, however, have a REDEFINES or OCCURS clause or may be subordinate to data items having a REDEFINES or OCCURS clause.

(v) FILLER data items are ignored.

(vi) CORRESPONDING items can have different locations within the group and the field sizes can also be different.

Examples

Let us consider the following DATA DIVISION entries.

```
01      OLD-REC.
        02      FIRST-PART.
                03      ITEM-1          PIC         999.
                03      ITEM-2          PIC         999.
                03      ITEM-3          PIC         9[5].

        02      SECOND-PART.
                03      SEC-1           PIC         9[4].
                03      SEC-2.
                        04  SEC-21      PIC         X[5].

        02      THIRD-PART.
                 03     THIRD-1         PIC         XXX.
                 03     THIRD-2         PIC         999.

01      NEW-REC.
        02      FIRST-PART.
                03      ITEM-1          PIC         9999.
                03      ITEM-2          PIC         9[3].
                03      ITEM-3          PIC         X[5].
                03      ITEM-4          PIC         X[21].
```

```
02      SECOND-PART.
        03      SEC-1       PIC     X[4].
        03      SEC-21      PIC     X[5].

02      FOURTH-PART.
        03      THIRD-1     PIC     XXX.
        03      THIRD-2     PIC     999.
```

Now, let us see which data items will be moved if the PROCEDURE DIVISION contains the statement MOVE CORRESPONDING OLD-REC TO NEW-REC. The said data items are ITEM-1, ITEM-2, ITEM-3 and SEC-1. Note that SEC-21, THIRD-1 and THIRD-2 cannot take part in the operation. This is because although those names are common to both the groups, their qualifiers are different. If, on the other hand, the PROCEDURE DIVISION statement is ADD CORRESPONDING OLD-REC TO NEW-REC, only ITEM-1 and ITEM-2 will take part in the add operation. This is because ITEM-3 and SEC-1 in NEW-REC are not numeric data items.

9.4 ROUNDED OPTION

Let us consider the following DATA DIVISION entries.

```
77      A       PIC     99V999      VALUE       IS      23.412.
77      B       PIC     99V999      VALUE       IS      35.273.
77      C       PIC     99V9        VALUE       IS      41.5.
```

Now, after the execution of the statement ADD A B GIVING C, C will contain 58.6 instead of 58.685 as C can retain only one digit after the decimal point. Instead of this usual truncation, rounding can be specified through the ROUNDED option. The ROUNDED option can be specified as follows:

```
ADD     A       B       GIVING      C       ROUNDED
```

Now, the content of C will be 58.7 instead of 58.6.

It may be noted from the above example that whenever as arithmetic operation is executed, if the number of places in the fractional part of the result happens to be greater than the number of placed provided for the fractional part in the receiving field, truncation will occur. However, if the ROUNDED option is specified, 1 is added to the last digit whenever the most significant digit being thrown out is greater than or equal to 5. In the example shown here the most significant digit of the excess is 8 which is greater than 5. Therefore, 1 has been added to 6 which is the last digit of the receiving field. On the other hand, if A and B contains 23.412 and 35.213 respectively, both the statements

```
ADD     A       B       GIVING      C
```

and

```
ADD     A       B       GIVING      C       ROUNDED
```

will give the same result and in both the cases C will have the value 58.6.

The ROUNDED option can be specified in the case of any arithmetic verb by writing the word ROUNDED after the identifier, denoting the field that receives the result of the

operation. The ROUNDED phrases cannot be specified for the identifier that receives the remainder in the DIVIDE operation.

9.5 ON SIZE ERROR OPTION

If after an arithmetic, the result exceeds the largest value that can be accommodated in the result, the error is called a size error. To take an example, let A and B be two elementary items with pictures 99 and 999 respectively. Suppose also that the current values of the two fields are 35 and 980 respectively. Now, the execution of the statement ADD A To B causes a size error. This is because the result field B is not large enough to hold the result of the addition, namely, 1015.

When a size error occurs, the contents of the result field after the operation is unpredictable. However, the processing is not terminated and the computer will proceed with the execution of the next statement regardless of the fact that a size error occurred. Therefore, it is the responsibility of the programmer to monitor the arithmetic operation by specifying the ON SIZE ERROR phrase at the end of the arithmetic statement. It has the following syntax:

; ON SIZE ERROR imperative - statement

When this phrase is specified the imperative statement gets executed, if an ON SIZE ERROR occurs. Thus a statement

ADD A TO B ON SIZE ERROR GO TO ERROR-PARA.

will cause the control to be transferred to ERROR-PARA in the case of a size error. Otherwise, the effect will be the same as that of ADD A TO B. When the ON SIZE ERROR phrase is specified, the arithmetic statement must be terminated by a period.

The ON SIZE ERROR phrase enables a programmer to take measures in case a size-error condition arises. However, specifying the ON SIZE ERROR phrase with each and every arithmetic operation can increase the execution time of the program. Thus when the programmer is sure that there is no possibility of a size error, the phrase may not be specified. In this connection it is recommended that the programmer should give the result fields enough room so that size error does not occur.

It may be worthwhile to note the differences between the ROUNDED and SIZE ERROR options. The ROUNDED option is concerned with the case when a loss of digits occurs at the right end. This loss merely makes the result approximate, but the result is not altogether wrong. The ROUNDED option only affects the nature of approximation. If specified, the result is approximately by rounding. Otherwise, it is approximated be truncation. On the other hand, SIZE ERROR is concerned with the case when a loss of digits occurs in the most significant part (left end). The result in such a case is totally wrong.

9.6 LET US SUM UP

In the present lesson the learner is exposed to the corresponding options with MOVE/ ADD/SUBTRACT. Points about the rounded option and size error option are also discussed.

9.7 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Explain with syntax MOVE Corresponding
2. Explain with syntax ADD Corresponding
3. Explain with syntax SUBTRACT Corresponding
4. Explain with syntax ROUNDED option
5. Explain with syntax ON SIZE ERROR option

9.8 POINTS FOR DISCUSSION

1) Explain the usage MOVE CORRESPONDING verb with example.

2) Explain the usage of CORRESPONDING option with ADD and SUBTRACT verbs.

3) List out the rules for using CORRESPONDING option.

9.9 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 10: PROGRAMS USING ARITHMETIC VERBS

CONTENTS

10.0 AIMS AND OBJECTIVES

Having learnt the arithmetic verbs with their syntaxes and examples, the learner will now be exposed to simple programs to demonstrate the arithmetic verbs.

10.1 PROGRAM FOR ADD VERB

Write a simple program to demonstrate ADD verb. Use edit characters also in the program.

```
Identification division.
Program-id. Addverb.
Environment division.
Data division.
        Working-storage section.
77 a  pic    s9(3)v9(2) value 0.
77 b  pic     s9(3)v9(2) value 0.
77 c  pic     s9(4)v9(2) value 0.
77 e-c pic   +z(4).z(2).
Procedure division.
Para-1.
        Identification division
        program-id. Add verb.
        environment division
        data division
        working – storage section.
                77      A       PIC     S9(3)   V9(2)
                77      B       PIC     S9(3)   V9(2)
                77      C       PIC     S9(4)   V9(2)
                77      e-a     PIC     +z(4)  Value 0
        PROCEDURE DIVISION
```

PARA - 1

Explanation : This program gets 2 inputs from user.
It adds them using ADD verb.
c=a+b
The unedited result is available in c.
We move c to e-c, where edit characters are available.
Note that the variable e-c makes use of the + edit character.
Result is displayed with e-c.

10.2 PROGRAM FOR SUBTRACT VERB

Write a simple program to demonstrate SUBTRACT verb. Use edit characters also in the program.

```
Identification division.
Program-id. Subverb.
Environment division.
Data division.
Working-storage section.
    77  a  pic   s9(3)v9(2) value 0.
    77   b  pic    s9(3)v9(2) value 0.
    77   e-b pic  +z(3).z(2).
Procedure division.
Para-1.
        Display(1 1) erase.
        Display(3 5) “Enter first number :”.
        Accept a.
        Display(5 5) “Enter second number :”.
        Accept b.
        Subtract a  from b.
        Move b to e-b.
        Display(15 5) “b-a = “ e-b.
        Stop run.
```

Explanation : This program gets 2 inputs from user.
It subtracts them using SUBTRACT verb.
b-a is calculated and stored in b.
After execution : OLD value of a is same
OLD value of b is lost and
b-a is stored in b.
The unedited result is available in b.
We move b to e-b, where edit characters are available.
Note that the variable e-b makes use of the + edit character.
Result is displayed with e-b.

10.3 PROGRAM FOR MULTIPLY VERB

Write a simple program to demonstrate MULTIPLY verb. Use edit characters also in the program.

```
Identification division.
Program-id. Mulverb.
Environment division.
```

```
Data division.
Working-storage section.
    77  a  pic  s9(3)v9(2) value 0.
    77   b  pic   s9(3)v9(2) value 0.
    77   c pic    s9(4)v9(2) value 0.
    77   e-c pic  -z(4).z(2).
Procedure division.
Para-1.
        Display(1 1) erase.
        Display(3 5) “Enter first number :”.
        Accept a.
        Display(5 5) “Enter second number :”.
        Accept b.
        Multiply a by b giving  c.
        Move c to e-c.
        Display(15 5) “Product = “ e-c.
        Stop run.
```

Explanation : This program gets 2 inputs from user.
It multiplies them using MULTIPLY verb.
a*b is found and stored in c.
The unedited result is available in c.
We move c to e-c, where edit characters are available.
Note that the variable e-c makes use of the - edit character.
Result is displayed with e-c.

10.4 PROGRAM FOR DIVIDE VERB

Write a simple program to demonstrate DIVIDE verb. Use edit characters also in the program.

```
Identification division.
Program-id. Divverb.
Environment division.
Data division.
Working-storage section.
    77  a  pic    s9(3)v9(2) value 0.
    77   b  pic     s9(3)v9(2) value 0.
    77   c pic      s9(4)v9(2) value 0.
    77   e-c pic   -z(4).z(2).
Procedure division.
Para-1.
        Display(1 1) erase.
        Display(3 5) “Enter first number :”.
        Accept a.
        Display(5 5) “Enter second number :”.
        Accept b.
        Divide a by b giving c.
        Move c to e-c.
        Display(15 5) “Answer = “ e-c.
        Stop run.
```

Explanation : This program gets 2 inputs from user.
It divides them using DIVIDE verb.
a/b is calculated and stored in c.
The unedited result is available in c.
We move c to e-c, where edit characters are available.
Note that the variable e-c makes use of the - edit character.
Result is displayed with e-c.

10.5 PROGRAM FOR SIZE ERROR

Write a simple program to demonstrate on size error option.

```
Identification division.
Program-id. sizeerr.
Environment division.
Data division.
Working-storage section.
    77  a     pic    s9(3) value 0.
    77  b      pic    s9(3) value 0.
    7    c      pic    s9(3) value 0.
    77  e-c   pic   +z(3).
Procedure division.
Para-1.
        Display(1 1) erase.
        Display(3 5) “Enter first number :”.
        Accept a.
        Display(5 5) “Enter second number :”.
        Accept b.
        Add a b to c on size error
              display (10 5) “Size error on c ---Please increase size”
               go to end-para.
        Move c to e-c.
        Display(15 5) “Sum = “ e-c.
End-para.
        Stop run.
```

Explanation : This program gets 2 inputs from user.
It adds them using ADD verb.
The unedited result is available in c.
We move c to e-c, where edit characters are available.
Note that the variable e-c makes use of the + edit character.
Result is displayed with e-c.

Note : If the user gives
a=955 and b=288 then c will become 1243
and the size error will be present on c,
as c can store at the maximum 3 digits.
To clear out this problem the size of c can be declared like this.
01 c pic 9(4) value 0.
01 e-c pic +z(4).

10.6 PROGRAM FOR COMPUTE VERB

Write a simple program to demonstrate COMPUTE verb.

```
Identification division.
Program-id. compverb.
Environment division.
Data division.
Working-storage section.
    77  a  pic s9(3)v9(2) value 0.
    77  b  pic s9(3)v9(2) value 0.
    77  c  pic s9(4)v9(2) value 0.
    7   e-c pic  +z(4).z(2).
Procedure division.
Para-1.
        Display(1 1) erase.
        Display(3 5) “Enter first number :”.
        Accept a.
        Display(5 5) “Enter second number :”.
        Accept b.
        Compute c= a+b.
        Move c to e-c.
        Display(15 5) “Answer = “ e-c.
        Stop run.
```

10.7 LET US SUM UP

With the help of the above programs one can understand the working nature of arithmetic verbs like Add, Subtract, Multiply, Divide and Compute. Also one can acquire knowledge about size error options.

10.8 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Write a Program to add 4 values using ADD verb.
2. Write a Program to subtract the 4^{th} value from the first 3 values using SUBTRACT verb.
3. Write a Program to multiply 5 values and store the result in 6^{th} variable..
4. Write a Program to convert the temperature given in Centigrade to Fahrenheit using COMPUTE verb. (Hint : F= 1.8C+32)

10.9 POINTS FOR DISCUSSION

1) Write a simple COBOL program to illustrate ADD verb.

2) Write a simple program in COBOL to illustrate SUBTRACT verb.

3) Write a simple program in COBOL to illustrate ON-SIZE error option.

10.10 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2^{nd} Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4^{th} Edition.

UNIT – III

LESSON – 11: CONDITIONS

CONTENTS

11.0 AIMS AND OBJECTIVES

The main aim of this lesson is to introduce what is meant by condition and the different types of them. This will facilitate the learner to use whenever the need arise to include them in programs.

11.1 CONDITION

A condition is an entity that at one point of time can have only one of the two values – true or false. As already pointed out, the IF verb makes use of conditions. In COBOL a condition can be any one of the following:

(i) Relational condition
(ii) Sign condition
(iii) Class condition
(iv) Condition-name condition
(v) Negated simple condition and
(vi) Compound condition.

11.2 RELATIONAL CONDITION

We know that a relational condition indicates a comparison between two operands and has the form.

Operand-1 relational-operator operand-2

where the relational-operator can be any one of the following:

IS [NOT] GREATER THAN
IS [NOT] > THAN

IS [NOT] LESS THAN
IS [NOT] < THAN
IS [NOT] EQUAL TO
IS [NOT] =

It was stated earlier that the operands can be an identifier or a literal. However, either operand can also be an arithmetic expression but must contain at least one reference to an identifier. Sometimes, operand-1 and operand-2 are respectively referred to as the subject and object of the relational condition.

Comparison of Numeric Operands

We are familiar with the kind of relational condition where both the operands are numeric. The comparison in this case is algebraic and the two operands can be compared regardless of the size and USAGE of the fields.

Comparison of Nonnumeric Operands

A nonnumeric operand (identifier/literal other than numeric) can be compared to another nonnumeric operand according to the following rules.

(i) Fields of Equal Sizes

Characters in the corresponding positions are compared to determine the value of the relational condition. Comparison starts with the leftmost character in both the fields and proceeds in a left to right manner. If the two characters being compared are found to be unequal at any stage, the field containing the greater (according to the collating sequence of the computer (NATIVE) or that specified by the PROGRAM COLLATING SEQURNCE clause in the OBJECT-COMPUTER paragraph) characters is considered to be greater. Only when the characters are found to be identical does the comparison proceed to the next position on the right. Two fields are taken to be equal only when all such pairs of characters have been found to be identical and the rightmost end has been reached.

(ii) Fields of Unequal Sizes

If the two operands are not of equal size, the shorter field is considered to be extended on the right by spaces to make its size equal to the longer field and the rules for comparing fields of equal sizes are used.

Comparison of a Numeric Operand with a Nonnumeric Operand

A numeric operand can be compared to a nonnumeric operand subject to the following restrictions.

(i) The numeric operand must be an integer data item or integer literal.
(ii) Both the operands must have the same USAGE (DISPLAY or some form of DISPLAY).

At the time of comparison, the numeric operand is treated as if its value were moved to an alphanumeric item were then compared to the nonnumeric field.

Group Item as an Operand in Relational Condition

When an operand of a relational condition is a group item, the said item is considered to be an alphanumeric field.

The following examples illustrate the results of different comparisons. Usage DISPLAY is assumed in all cases.

A (Operand-1)		B (Operand-2)		Result of Comparison
Picture	Value	Picture	Value	
X(3)	001	X(4)	0001	A>B
X(4)	3254	X(3)	325	A>B
X(3)	BOY	X(4)	GIRL	A<B
X(3)	BOY	X(4)	BIRD	A>B
X(3)	BOY	X(4)	BOYb	A=B
9(3)	354	X(2)	46	A<B
S9 (2) LEADING SEPARATE	-46	X(2)	46	A=B
9(3)	354	9(2)	46	A>B

11.3 SIGN CONDITION

The sign condition determines whether or not the algebraic value of an operand is positive, negative or zero. The operand can be either a numeric identifier or an arithmetic expression. The format of this condition is as follows:

{ identifier / arithmetic-expression } IS [NOT] { POSITIVE / NEGATIVE / ZERO }

When arithmetic expression is used, it must contain at least one identifier. The POSITIVE option determines the value of the condition to be true only if the value of the operand strictly positive. This means that the value zero is not treated as positive.

The following examples illustrate the use of the sign condition.

Example 1

```
77      BALANCE   PIC     S9 (6) V99.
.
.
IF      BALANCE   IS      ZERO GO TO NIL-BALANCE.
```

It may be noted that the above IF statement is equivalent to the following statement that makes use of a relational condition.

```
IF      BALANCE   =       0       GO TO NIL-BALANCE.
```

Example 2

```
02      DEPOSIT      PIC    9(4) V99.
02      WITHDRAWAL   PIC    9(4) V99.
        .
        .
        .
IF      DEPOSIT -WITHDRAWAL IS POSITIVE GO TO
        CALCULATION.
```

The control is transferred to the paragraph named CALCULATION if the current value of DEPOSIT is greater than that of WITHDRAWAL.

In general, any sign condition can be replaced by an equivalent relational condition. The use of the sign condition may perhaps be convenient in certain cases and its use may also increase the readability of the statement that uses it.

11.4 CLASS CONDITION

The class condition determines whether or not the value of an operand is numeric or alphabetic. An operand is numeric if it contains only the digits 0 to 9 with or without an operational sign. An operand is alphabetic if it contains only the letters A to Z and space. The format of the class condition is as follows:

```
Identifier   IS   [NOT]   { NUMERIC    }
                          { ALPHABETIC }
```

The following rules apply in the case of a class condition.

(i) The usage of the identifier must be DISPLAY or some forms of DISPLAY.

(ii) For the NUMERIC option the identifier must be either numeric or alphanumeric. If the data item is defined with an operational sign (picture contains S or a SIGN clause has been specified), then the appearance of sign (zoned in the units position or a leading or trailing sign, as the case may be) is considered to be normal.

(iii) For the alphabetic option, the identifier must be either alphabetic or alphanumeric.

(iv) The identifier may be a group item. However, for the NUMERIC option, the group item must not contain elementary items described with an operational sign.

The class condition is very useful for the validation of the input data. In COBOL, the data is read into the record area in the same form as recorded on the external medium regardless of the specified class of the individual fields in the record. For example, if we are reading the value of a numeric field from a card and the corresponding position in the card contains non-numeric characters, the system will not detect it to be an error. Instead, the nonnumeric characters will be stored in the character positions of the numeric field. This error may even pass unnoticed because during any subsequent numeric operation (such as numeric MOVE or arithmetic operation), only the numeric part of the characters in the field (except for the position that may indicate the operational sign) will be used. Thus the possible

punching mistake in the data card can go and detected unless proper care is taken. One may avoid some of these blunders (though not all) through the use of class condition.

Let BASIC-PAY be a data name in a card record defined with picture

9(5) V99. Having read the card we can test the value of BASIC-PAY to ensure that the data on the card is actually numeric. This can be done as follows

```
IF BASIC-PAY IS NOT NUMERIC GO TO PARA-ERROR.
```

If the data contains any character other than digits, control will be transferred to PARA-ERROR. Otherwise, control will go to the next sentence in sequence. It may be noted that the data must be punched with leading zeros, if any, and not with leading spaces. The space character is considered to be an alphabetic character.

11.5 CONDITIONS-NAME CONDITION

A condition name is an entity which itself is a condition and as such can have either a true or false value. However, a condition name cannot be defined independently. It must always be associated to a data name called the conditional variable. The condition name may be defined in any section of the DATA DIVISION and must be placed immediately after the entry that defines the conditional variable. There can be more than one condition names associated to a conditional variable. In that case all the condition name entries must follow the entry defining the conditional variable.

A condition name entry specifies either a single value or a set of values and/or a range of values for the conditional variable. The condition name becomes true whenever the conditional variable assumes any of these values. Otherwise, the condition name is set to false. It must be noted that it is not possible to set the value of a condition name explicitly. The value of a condition name is always set implicitly depending on the current value of the conditional variable. The format of the condition name entry is given below.

```
88  condition-name  { VALUE  }  IS   }  Literal-1  [ { THRU    }  literal-2 ]
                    { VALUES }  ARE  }              [ { THROUGH }            ]

[ , literal-3 [ { THRU    }  literal-4 ] … ]
              { THROUGH }
```

The following rules apply for a condition name.

(i) Condition names must be described at level 88. The level number begins in margin A or any position after it. The condition name must begin from margin B or any position after it. There must be at least one space between the level number and condition name.

(ii) The normal rules for naming a data item also apply in the case of a condition name.

(iii) If the same condition name is used in more than one place, the condition name must be qualified by the name of its conditional variable.

(iv) The name of the conditional variable can be used as a qualifier for any of its condition names. If the reference to a conditional variable requires

qualification or subscripting, the same combination of qualification or subscripting must also be used for the associated condition name.

(v) The values specified through the VALUE clause in the condition name entry must not conflict with the data description of the conditional variable. A literal in the VALUE clause can either a numeric literal, non numeric literal or figurative constant.

(vi) When the THRU/THROUGH phrase is used, literal – 1 must be less than literal – 2 and literal – 3 must be less than literal- 4.

(vii) A conditional variable can be an elementary item or a group item. However, it cannot be another condition name, or a 66-level item (RENAMES clause) or a group containing the JUSTIFY clause, or the SYNCHRONIZED clause or the USAGE clause other than DISPLAY.

The following is an example of the use of condition names

```
77    MARITAL-STATUS              PIC      9
88    SINGLE          VALUE       IS       ZERO
88    MARRIED         VALUE       IS       1.
88    WIDOWED         VALUE       IS       2.
88    DIVORCED        VALUE       IS       3.
88    ONCE-MARRIED    VALUE       ARE      1, 2, 3.
88    VALID-STATUS    VALUE       ARE      0 THRU 3.
```

It may be noted that six condition names have been defined here. All of them are associated with the conditional variable MARITAL – STATUS. If at a point of time, MARITAL STATUS gets the value of 2, then the condition names WIDOWED, ONCE-MARRIED and VALID-STATUS will become true and others will become false.

The condition names can be used as conditions. Thus in PROCEDURE DIVISION we may have statements, such as:

(a) IF SINGLE SUBTRACT 125 FROM DEDUCTIONS.
(b) IF ONCE - MARRIED ADD 32 TO SPECIAL-PAY.
(c) IF NOT VALID – STATUS GO TO ERROR – IN – STATUS.

In (a) above the statement SUBTRACT 125 FROM DEDUCTIONS will be executed if MARITAL-STATUS is equal to zero. Similarly, in (b) the ADD statement will be executed only when MARITAL – STATUS is equal to 1, 2 or 3 and in (c) the control goes to the procedure ERROR-IN – STATUS only when MARITAL-STATUS has a value other than the 0, 1, 2 or 3 . As in (c) a condition name can be preceded by NOT to indicate the negation of the condition.

The format for a condition – name condition is

[NOT] condition – name

It is important to note the usefulness of a condition name. When a condition name specifies a single value for the conditional variable, the condition name is equivalent to a relational condition. For example, in (a) above the condition name SINGLE is equivalent to the relational condition MARITAL-STATUS =0. Even when the condition name specifies more than one value for the conditional variable, the condition name can be replaced by an equivalent compound condition. Thus it may not be absolutely necessary to make use of the condition names. Thus it may not be absolutely necessary to make use of the condition

names. The main advantage of a condition name is that it increases the readability of the statement that uses it. Certainly, the use of the condition name WIDOWED conveys more information to a reader of the program than the use of the relational condition MARITAL-STATUS =2. Precisely for this reason, it is recommended that whenever possible, meaningful condition names should be used in a program.

11.6 NEGATED SIMPLE CONDITION

Any of the simple condition described above can be preceded by the logical operator NOT. The effect of placing the operator NOT before a simple condition is to reverse the value of the condition. It may be seen that the operator NOT can be used in two ways. In simple conditions it can be used as a part of the condition. It can also be used to precede a simple condition to make it a negated simple condition. An example of the first use may be DEPOSIT NOT LESS THAN 500.00 while an example of the second use is Not DEPOSIT LESS THAN 500.00. Of course, in this case, both the conditions mean the same thing and can be used in either form. What matters is the role of the operator NOT. In the former case NOT is part of a relational operator and in the latter case it is a logical operator. However, NOT must not precede a simple condition that includes NOT as a part if it.

11.7 COMPOUND CONDITION

Two simple conditions can be connected by the logical operators AND or OR to form a compound condition (also known as combined condition). When two conditions are combined by AND, the compound condition becomes true only when both the constituent conditions are true. In all other cases the compound condition is false. On the other hand, if OR is used to combine two conditions, the compound condition is true if either or both the constituent conditions are true. It is false only when both the conditions are false.

For example, the compound condition AMOUNT GREATER THAN 499 AND AMOUNT LESS THAN 1000 is a compound condition which will be true only when the value of AMOUNT is in the range 500 to 999(inclusive of both). This is because both the simple conditions are true for these values of AMOUNT. For other values of AMOUNT, only one of them is true. Similarly, the compound condition AMOUNT LESS THAN 500 OR AMOUNT GREATER THAN 999 will be false only when the value of AMOUNT is in the range 500 to 999.

A compound condition can consist of any number of simple or negated simple conditions joined either by AND or OR. Compound conditions in such cases are resolved as follows. Negated simple conditions are evaluated first. This is followed by the evaluation of pairs of resulting conditions around each AND in a left-to-right order. After this the resulting conditions around each OR are evaluated in a left-to-right manner. If required, parentheses can be used in compound conditions. In such cases all the conditions within the parentheses are evaluated first in accordance with the above rules. When parentheses are used within parentheses, evaluation proceeds from the least inclusive pair of the parentheses to the most inclusive pair.

In general, a compound condition has the following form:

Condition-1 { AND / OR } Condition-2 …….

Where condition-1 and condition-2 can be any one of the following:

- a simple condition
- a negated simple condition
- a compound condition optionally enclosed in parentheses
- a negated compound condition where a compound condition enclosed in parentheses is preceded by NOT

It may be noted that no two logical operators can appear side by side except that the operators AND or OR may be immediately followed by NOT.

Using the above mentioned rules, fairly complicates compound conditions can be constructed. However, in actual practice, the need for a complicated compound condition hardly arises. For the sake of readability, it is recommended that the use of complex compound conditions should be avoided.

The following is an example of the use of a compound condition. Consider the sentence

IF AGE IS LESS THAN 30 AND (HIGHLY-EDUCATED OR HIGHLY-EXPERIENCED) MOVE 3 TO BONUS-CODE.

Here, HIGHLY-EDUCATED and HIGHLY-EXPERIENCED are condition names. If either of them is true and if AGE is less than 30, 3 will be moved to BONUS-CODE. Notice the importance of parentheses. If these are removed, the compound condition can become true if HIGHLY-EXPERIENCED is true regardless of the value of AGE and that of the condition name HIGHLY-EDUCATED.

Abbreviation

Consecutive relational conditions in a compound condition can be abbreviated in certain cases as follows:

(i) When the subjects in the consecutive relational conditions are identical, the subject May be omitted from the one where it appears first.

(ii) when the subjects and relational operators in the consecutive relational conditions are Identical, the subject as well as the relational condition may be omitted, the subject as well as the except from the one where they appear.

Some examples of abbreviation are given below.

Example 1

The compound condition

AMOUNT GREATER THAN 499 AND AMOUNT LESS THAN 1000

can be abbreviated to

AMOUNT GREATER THAN 499 AND LESS THAN 1000

Here, the second appearance of the common subject AMOUNT has been omitted.

Example 2

The compound condition

CARD-CODE = 3 OR CARD-CODE = 5 OR CARD-CODE = 7

may be abbreviated to

CARD-CODE = 3 OR 5 OR 7

Here, the subjects as well as the relational conditions in the given compound condition are identical. Consequently, the second and third appearances of the subject and the relational condition have been omitted.

The consecutives relational conditions that are being considered for abbreviation may

also contain the word NOT. In this case the interpretation of the abbreviated condition can become ambiguous. To resolve the ambiguity the following rule has been recommended in the ANS I standard. The word NOT preceding a relational operator in an abbreviated condition is considered part of the relational operator. Otherwise, NOT is considered to be a logical operator negating the condition preceded by it. The following examples can help to understand this rule.

Example 3

The condition

AGE LESS THAN 30 AND NOT LESS THAN 20 OR 40

is interpreted to be an abbreviation of the compound condition

AGE LESS THAN 30 AND AGE NOT LESS THAN 20 OR AGE NOT LESS THAN 40

This is because NOT precedes the relational operator LESS THAN in the abbreviated condition and as such it is interpreted to be a part of the relational operator NOT LESS THAN.

Example 4

The condition

NOT AGE LESS THAN 20 AND 30

will be interpreted to be an abbreviation of

NOT AGE LESS THAN 20 AND AGE LESS THA 30.

Here, NOT precedes the data name AGE. It is therefore not considered to be part of the relational operator LESS THAN which has been abbreviated.

11.8 LET US SUM UP

In the present lesson we have focused our attention in learning what is meant by condition and the different forms of them with sufficient examples. The learner with this knowledge will certainly be in a position to include them in programs whenever the need arises.

11.9 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. What do you mean by conditions?
2. Explain about Relational condition
3. Explain about Sign condition
4. Explain about Condition Names
5. What do you mean by compound conditions? Explain.

11.10 POINTS FOR DISCUSSION

1) Write notes on SIGN condition.
2) What do you mean by condition-Name Condition? Explain.
3) Explain in detail about compound condition

11.11 REFERENCES

1) COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2) COBOL Programming , V. RAJARAMAN, PHI Pub
3) Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4) Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 12: CONDITIONAL STATEMENTS

CONTENTS

12.0 AIMS AND OBJECTIVES

Program writing requires checking of many conditions. Hence the knowledge about conditional statements becomes vital, without which one can not write most of the real world programs. Keeping this in view, this lesson introduces the learner the popular and effective IF statement with its different forms.

12.1 IF STATEMENT

The simple form of IF statement is

IF condition-1 statement-1

where the condition-1 may be any one of the following.

{ Identifier-1 / Literal -1 } { IS [NOT] GREATER THAN / IS [NOT] > / IS [NOT] LESSER THAN / IS [NOT] < / IS [NOT] EQUAL TO / IS [NOT] = } { Identifier-2 / Litreal-2 }

If the condition-1 becomes true then statement-1 will be executed. Statement-1 can be a simple statement or a compound statement. If the condition fails then control goes out to the next statement.

Example :

1) IF A > B DISPLAY " A IS BIG"

2) IF BASIC-PAY NOT > 5000 GO TO TEST-PARA.

12.2 IF … ELSE STATEMENT

We are already familiar with the simple form of the IF statement. The general form of the IF statement is as follows:

IF condition ; { statement-1 / NEXT SENTENCE } [; ELSE { statement-2 / NEXT SENTENCE }]

The condition can be any one of the conditions discussed above. Each of statement-1 and statement-2 represents one or more COBOL statement. When more then one statement is specified they must be separated by one or more spaces or by an optional semicolon (;) or comma (,). During execution, if the condition is found to be true, the statements represented by statement-1 are executed. On the other hand, if the condition is found to be false, the statements represented by statement-2 are executed. For ease of reference, we shall call the statements represented by statement-1 and statement-2 as then part and else part respectively.

It may be noted that either the then Part or else part is executed depending on the value of the specified condition. After that the control implicitly goes to the statement that immediately follows the IF sentence.

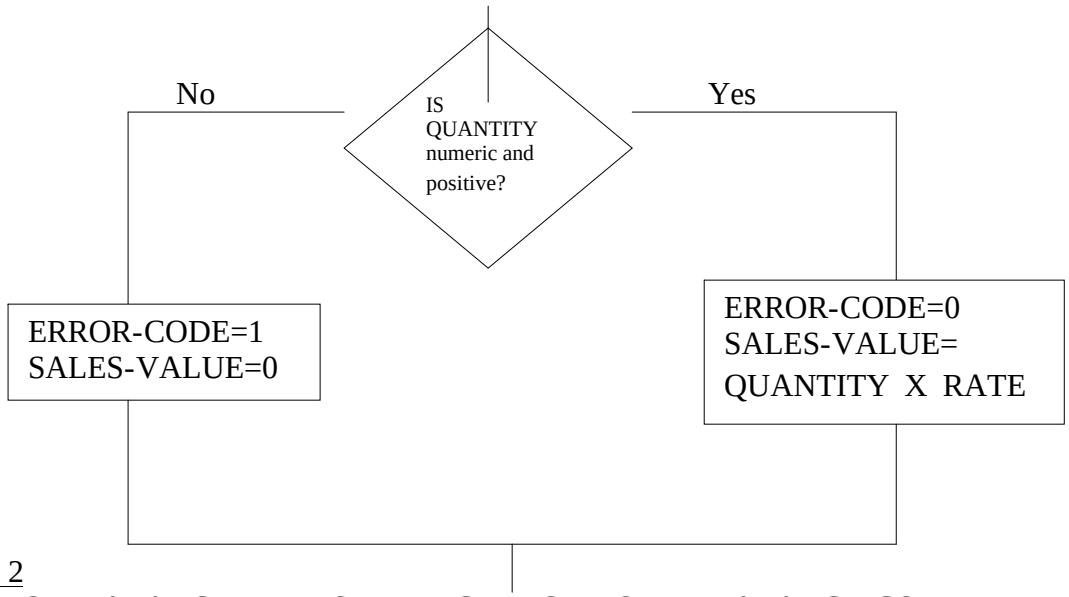
Normally, an If statement should be terminated by a period (.) followed by a blank (see next section for exception). For this reason an IF statement is often referred to as an IF sentence. Sometimes, we encounter situations where no action needs to be specified if the condition is true, but some actions are necessary if the condition is false. In that case, the NEXT SENTENCE phrase can be used for the then part and the else part can be written to the indicate the actions required. Similarly, the NEXT SENTENCE phrase can replace the else part if no action is required when the condition is false. The NEXT SENTENCE phrase indicates that the control should pass to the statement that follows the IF sentence. Note that if no action needs to be specified for the else part, the phrase ELSE NEXT SENTENCE, being optional, can be omitted. It is in this form that we have used the IF statement so for. However, the phrase ELSE NEXT SENTENCE may not be omitted in certain cases.

The following examples illustrate the use of IF statement.

Example 1

```
IF   QUANTITY IS NUMERIC AND QUANTITY IS POSITIVE
     MOVE ZERO TO ERROR-CODE. COMPUTE SALES VALUE=QUANTITY*RATE
ELSE MOVE 1 TO ERROR –CODE
     MOVE ZERO TO SALES-VALUE
```

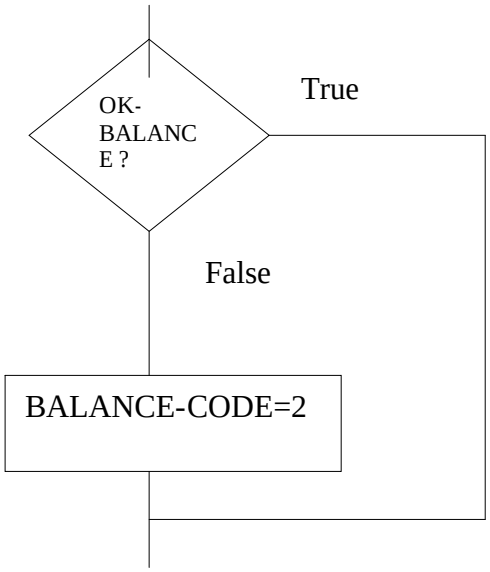
The specified condition tests whether or not the current value of the data name QUANTITY is numeric as well as positive. If the condition is true ERROR-CODE is set to zero and SALES-VALUE is computed by multiplying QUANTITY by RATE. On the other hand, if the condition is FALSE, ERROR CODE is set to 1 and SALES-VALUE is set to zero. In either case the control goes implicitly to the next statement after this IF sentence. The above sentence is equivalent to the following flowchart.



Example 2

IF OK-BALANCE NEXT SENTENCE ELSE MOVE 2 BALANCE-CODE

Here, OK-BALANCE is a condition name. No action is specified if this condition is true. If the condition is false, BALANCE-CODE should be set to 2. The sentence is equivalent to the following flow chart.



It may be noted that any If sentence can be alternatively written by just negating the condition. In order to retain the meaning of the original sentence, the then and else parts are to be interchanged. Thus the following sentence has the same meaning as the one illustrated above.

IF NOT OK-BALANCE MOVE 2 TO BALANCE-CODE ELSE NEXT-SENTENCE

If desired, the ELSE NEXT SENTENCE phrase may be dropped to get the following equivalent form.

IF NOT OK-BALANCE MOVE -2 TO BALANCE-CODE.

12.3 NESTED IF STATEMENT

The then and else statement of an IF statement can contain other IF statements. The included IF statements in their turn may also contain other IF statements. Such inclusion of one or more IF statements within the scope of an IF statement is called nesting. Note that the most inclusive IF statement must have a terminating period and thus this statement along with all the included statements is often called or NESTED IF STATEMENT.

Since the else phrase in an IF statement is optional, a nested If sentence may have fewer ELSEs than Ifs. This makes the interpretation of a nested IF sentence rather difficult. The first step in interpreting such a sentence would be to find out which ELSE belongs to which IF and which are the IFs donot have the corresponding ELSEs. Once this is done, the actions specified for the different cases can be recognized easily. To avoid any ambiguity in interpretation, the COBOL rule in the matter is as follows.

The nested IF sentence should be examined in a left- to-right manner to encounter each ELSE in the order of its appearance. As soon as an ELSE is encountered, it must be paired with the immediately preceding IF that has not yet been paired with another ELSE.

Note that the above rule also helps in detecting those Ifs for which the ELSE phrase may be absent.

The above rule states how the COBOL compiler will interpret a nested IF sentence. Therefore, while writing such a sentence this rule must be applied to verify that the interpretation of the compiler will not be different from what is intended. The following are examples to show how the meaning of a nested IF sentence can be obtained by applying the above rule.

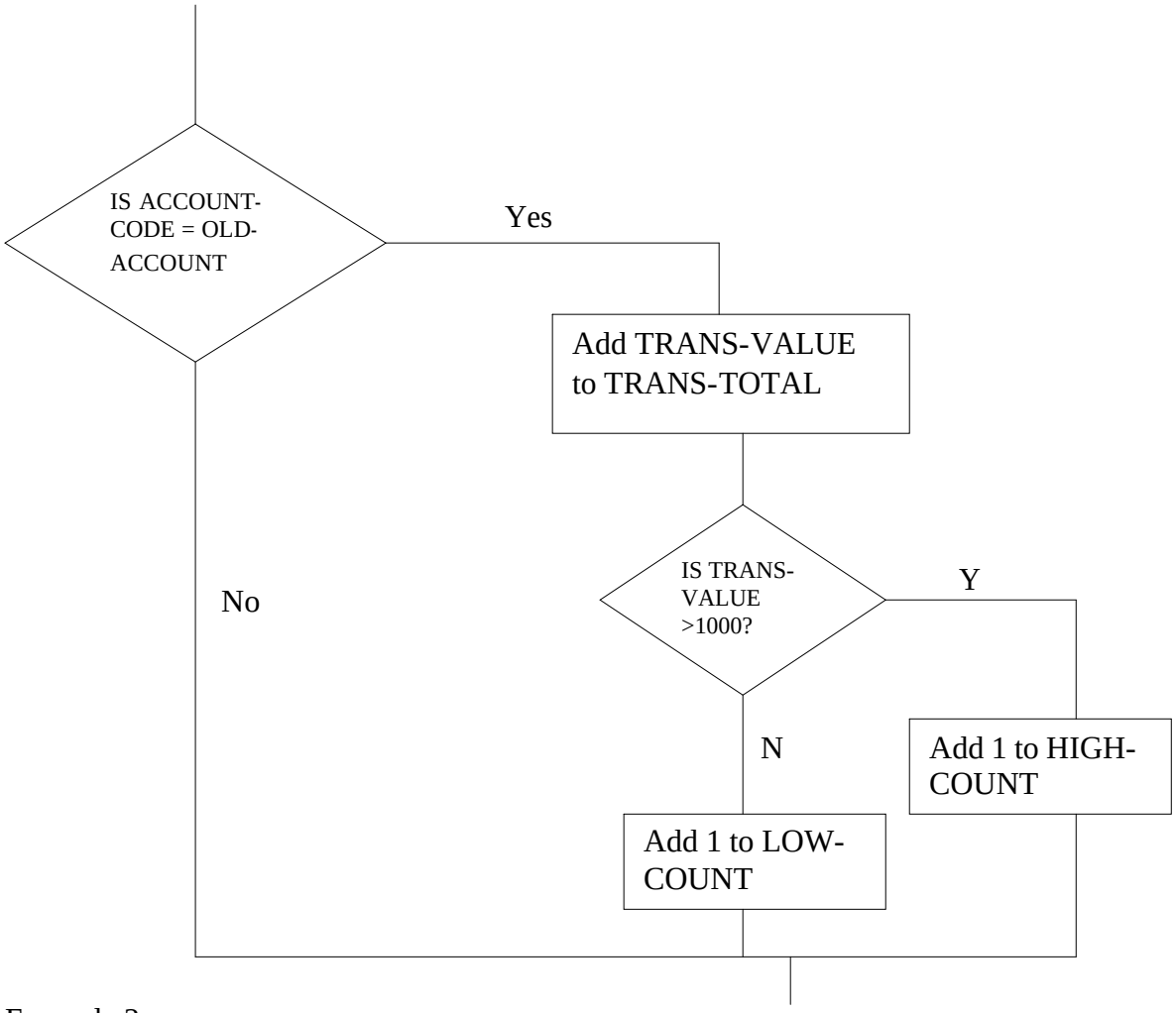
Example 1

Consider the following sentence

```
If account-code is equal to old-code
②     add trans-value to trans-total
if trans-value >1000
①     add 1 to high-count
else
①     add 1 to low-count
```

This nested IF sentence contains two IFs and one ELSE. The IF-ELSE pair has been marked by the number 1 within a circle. The IF marked with 2 within a circle is a lone IF without any ELSE.

Now it can be easily seen that the above sentence is equivalent to the following flow chart.

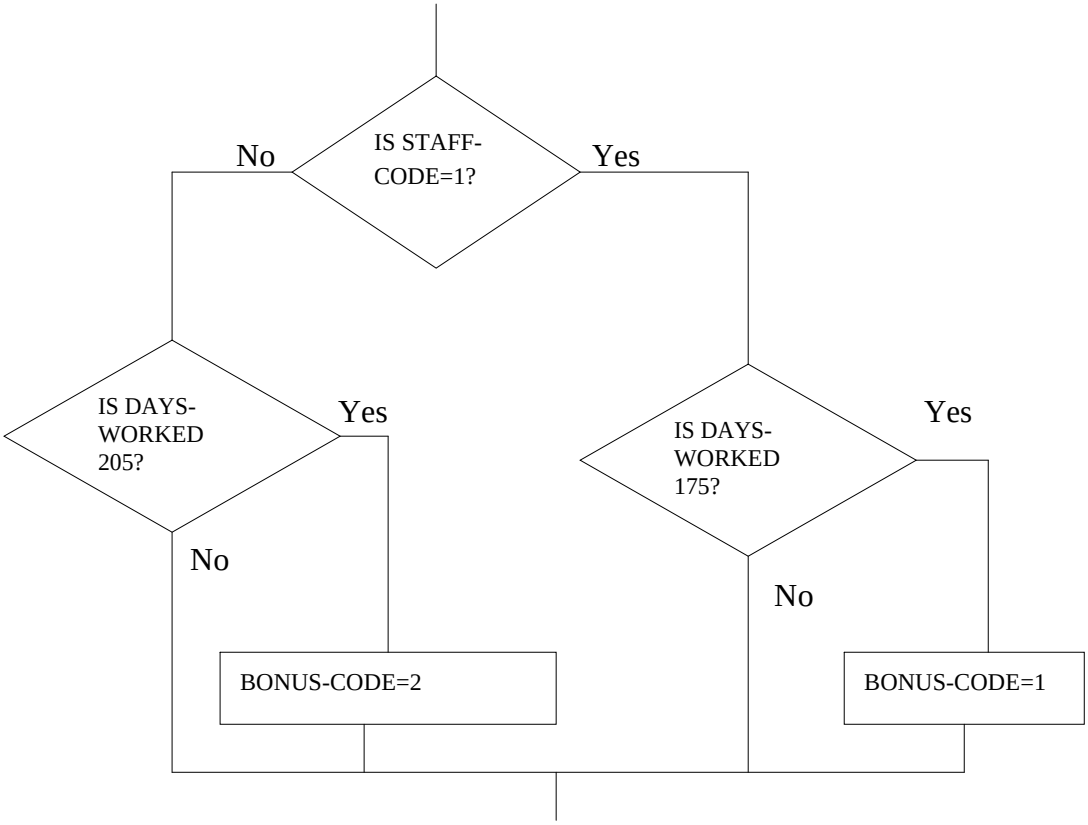


Example 2

Consider the following nested sentence:

```
IF STAFF-CODE =1
②     IF DAYS-WORKED IS GREATER THAN 175
          ①     MOVE1 TO BONUS-CODE
      ELSE NEXT SENTENCE
          ①
ELSE IF DAYS-WORKED IS GREATER THAN 205
②     ③     MOVE 2 BONUS-CODE
      ELSE NEXT SENTENCE.
          ③
```

The IF-ELSE pairs in this sentence can be detected by applying the rule stated above. We indicate the IF-ELSE associations by marking each pair with identical numbers within circle. The sentence can now be easily recognized as an implementation of the following flow chart.



It can also be seen that while the last ELSE NEXT SENTENCE phrase can be dropped, it is not possible to drop the first ELSE NEXT SENTENCE phrase. If this first phrase is omitted, the pairing of IF's and ELSE's will be disturbed. It is left as an exercise to verify that the BONUS-CODE will then remain unchanged if the STAFF-CODE is other than the 1. Although the same action will be taken when STAFF-CODE is equal to 1, it is worth while to observe that this is because the condition (DAYS-WORKED IS GREATER THAN 205) in the last IF statement can never be true.

Therefore, we observe the ELSE NEXT SENTENCE phrase is useful in certain cases. The phrase also increases the readability. Therefore, the use of NEXT SENTENCE either in the then or else parts of an IF statement is recommended. The only exception can be when the ELSE NEXT SENTENCE phrase is immediately followed by the terminating period of the IF sentence. This is because in this case the said ELSE must apply to the IF of the most inclusive IF statement of the nesting. Therefore, the control will be transferred to the next sentence regardless of whether or not the ELSE NEXT SENTENCE phrase is explicitly included.

12.4 LET US SUM UP

The above lesson has introduced the learner the IF statement and its variants with examples. With this knowledge the learner can make use of these statements whenever the need arises to use them. Most of the programs will be making use of IF statement. Nested IF statement has to be used carefully.

12.5 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Explain with syntax IF statement .
2. Explain with syntax IF .. ELSE statement .
3. Explain with syntax NESTED .. IF statement .
4. Write a few IF statements and explain them.

12.6 POINTS FOR DISCUSSION

1) Explain in detail about the different types of IF statements with examples.

12.7 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 13: PERFORM STATEMENTS

CONTENTS

13.0 AIM AND OBJECTIVES

Perform statements play an important role in COBOL programs. Repeated executions of paragraphs based on conditions make use of Perform statements. The learner will therefore be introduced the different forms of the Perform statements. Based on the requirement, the appropriate one can be employed in the programs.

13.1 PERFORM STATEMENT

The PERFORM statement can be used to execute a group of consecutive statements written elsewhere in the program. We shall refer to this group of statements as the range of the PERFORM statement. During execution, when a PERFORM statement is encountered, a temporary departure from the normal sequential execution takes place and the statements contained in the specified range are executed. Upon execution of the said statements, the control implicitly returns to the next statement following the PERFORM statement. It is also possible to get the statements in the said range executed repetitively for a specified number of times or until a condition is satisfied. PERFORM is a powerful COBOL verb and has five different forms.

The format of a simple PERFORM statement is as follows:

PERFORM procedure-name-1 [{ THRU / THROUGH } procedure-name-2]

The statement group beginning with the first statement of the procedure named in procedure-name-1 and ending with the last statement of the procedure names in procedure-name-2, constitutes the range consists of the statements contained in the procedure referred to by procedure-name-1. when the simple PERFORM statement is executed, this range is executed only once.

Some examples of the PERFORM statement are given below.

Example 1

PERFORM CALCUALTE-TAX.

In this example, CALCULATE-TAX is either a section name or paragraph name. Suppose it is a section name. All the statements contained in this section will be executed as a result of the execution of the PERFORM statement and after the execution of these statements, the control will come back to the statement following the PERFORM statement.

Example 2

PERFORM BEGIN-CALCULATION THRU END-CALCULATION.

Suppose, BEGIN-CALCULATION and END-CALCULTION are paragraph names. The execution of the above PERFORM statement will cause the execution of the group of the statements starting with the first statement of BEGIN-CALCULATION and ending with the last statement of end-calculation. It may be noted that there may be other paragraphs in between these two paragraphs. All these paragraphs area also included in the range. Upon the execution of the range, the control returns to the statement following the PERFORM statement.

It may be noted that the return of control after the execution of the statements in the specified range takes place implicitly. This means that at the end of the range, the programmer should not put any statement (such as GO TO) to transfer the control explicitly to the statement following the PERFORM statement. The compiler establishes a return mechanism at the end of the range and it is this mechanism which is responsible for the return of the control.

The following points may be noted in connection with the range of a PERFORM statement.

(i) A GO TO statement is allowed within the range of a PERFORM statement. However, it is the responsibility of the programmer to ensure that the control Ultimately reaches the last statement of the range.

(ii) There is no restriction as to what can be the last statement of a range except that it cannot be a GO TO statement. When an IF sentence is used at the end of a range, he next sentence (specified implicitly or explicitly) for that IF sentence refers to the return mechanism.

(iii) The use of a PERFORM statement within the range of another PERFORM Statement is allowed, Some compilers allow unrestricted use of such nesting of PERFORM statements (except that there may be limitations on the depth of Nesting depending on the operating system and hardware capabilities). Some Compilers require that the range of the included PERFORM statement must be either completely within or completely outside the range of the invoking PERFORM statement. In other words, the sequence of ranges specified in the Nested PERFORM statements should neither overlap nor share the same Terminal statement. It is better to observe these restrictions for the sake of Portability.

(iv) The range of statements that should be performed gets linked up with the PERFORM STATEMNT ONLY WHE NTHE LATTER IS EXECUTED. If the control reaches the first statement of the range through normal sequence or through explicit transfer of the control, then also the range gets executed in the

normal way. After the execution of the last statement of the range, the control falls through the next statement following the range.

13.2 PERFORM WITH TIMES OPTION

The format of PERFORM TIMES statement is as follows:

PERFORM procedure-name-1 [{ THRU / THROUGH } procedure-name-2]
{ Identifier / integer } TIMES

Example : 1) Perform para-2 5 times.

In this case the specified para-2 will be repeatedly executed 5 times.

2) Perform para-2 thru para-5 n times.

In this case the specified range of all paragraphs from para-2 to para-5 will be repeatedly executed n times. Note that the value of n should be available before this statement gets executed.

13.3 PERFORM WITH UNTIL OPTION

The format is as follows:

PERFORM procedure-name-1 [{ THRU / THROUGH } procedure-name-2]
UNTIL condition

Examples : 1) Perform p-3 until i > 5.
Here p-3 will be executed whenever i <= 5 . When the condition fails it executes p-3 and if the condition turns out to be true it skips p-3.

2) Perform p-2 thru p-4 until k > n.
It is similar to the previous example. All the paragraphs from p-2 to p-4 will be executed till the specified condition is satisfied.

13.4 PERFORM WITH VARYING OPTION

The format is as follows:

PERFORM procedure-name-1 [{ THRU / THROUGH } procedure-name-2]

VARYING { identifier-1 / Index-name-1 } FROM { identifier-2 / index-name-2 / Literal-1 }

BY { identifier-3 / Literal-2 } UNTIL condition

Examples : 1) Perform para-3 varying k from 1 by 1 until k > 10.

Here para-3 will be repeatedly executed 10 times (For k=1,2,3, …..10)

2) Perform p-2 thru p-4 varying j from 1 by 2 until j > 100.
Here paragraphs from p-2 through p-4 will be repeatedly executed for j=1,3,5 …,99.

3) Perform para-3 varying I from 1 by 1 until I > 50
after J from 1 by 1 until J > 10.
Here para-3 will be executed 500 times. Keeping I=1 (J=1,2,3,…10),I=2 (J=1,2,3…10) …. And I=50 (J=1,2,3,…10). In this example 2 loops are used.

13.5 LET US SUM UP

The learner has just now been introduced the fascinating forms of PERFORM statements with their syntaxes and examples. The learner has been completely explained about Simple PERFORM, PERFROM…THRU, PERFORM …TIMES, PERFORM … UNTIL and PERFORM… VARYING options. This lesson will make the learner to use the most appropriate form of PERFORM that suits based on the purpose of work.

13.6 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Why we need PERFORM statements?
2. Explain with syntax PERFORM … THRU
3. Explain with syntax PERFORM … TIMES
4. Explain with syntax PERFORM … UNTIL
5. Explain with syntax PERFORM … VARYING

13.7 POINTS FOR DICUSSION

1) Explain in detail about the different types of PERFORM statements with examples.

13.8 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 14 : RENAMES & REDEFINES CLAUSES

CONTENTS

14.0 AIMS AND OBJECTIVES

This lesson introduces the Renames Clause and Redefines Clause to the learner. Many COBOL applications require renaming of existing groups and redefining them for specific purpose.

14.1 RENAMES CLAUSE

Sometimes a re-grouping of elementary data items in a record may be necessary so that they can belong to the original as well as to the new group. This is possible in COBOL by the use of the RENAMES clause. The following example illustrates the use of the RENAMES clause.

```
01      PAY-REC.
        03      FIXED – PAY.
                05      BASIC PAY                   PIC         9(6) V99
                05      DEARNESS-ALLOWANCE          PIC         9(6) V99
        03      ADDITIONAL – PAY
                05      HOUSE -  RENT               PIC         9(4)V99
                05      MTHLY - INCENTIVE           PIC         9(3)V99
        03      DEDUCTIONS.
                05       PF – DEDUCT                PIC         9(3)V99
                05       IT – DEDUCT                PIC         9(4)V99
                05       OTHER - DEDUCT             PIC         9(3)V99
66      PAY-OTHER-THAN-BASIC  RENAMES  DEARNESS-ALLOWANCE THRU
        MTHLY-INCENTIVE
66      IT-AND-PF-DEDUCTIONS RENAMES PF-DEDUCT THRU IT-DEDUCT
```

In the example, PAT-OTHER-THAN-BASIC will become a new group consisting of DEARNESS-ALLOWANCE, HOUSE-RENT and MTHLY-INCENTIVE. Note that the new group overlaps on two original groups, namely, part of FIXED-PAY and the entire ADDITIONAL-PAY. Such overlapping is allowed provided the elementary items are all contiguous. In a similar way IT-AN-DEDUCTIONS has two elementary items PF-DEDUCT

and IT-DEDUCT. This new group is formed out of the original group deductions. Alternatively, the same thing can also be done in the original description itself by placing the IT-AND-PF-DEDUCTIONS at level 04 under DEDUCTIONS. The exact syntax of the RENAMES clause is as follows:

66 data-name-1 RENAMES daa-name-2 THRU data-nmae-3

14.2 RULES OF RENAMES CLAUSE

The following rules must be observed while using the RENAMES clause:

(i) All RENAMES entries must be written only after the last record description entry.

(ii) The RENAMES clause must be used only with the special level number 66. the level number begins in margin A or any position after it. Data-name-1 must begin from margin B or any position after it. There must be at least one space between the level number and data-name-1.

(iii) Data-name-2 and data-name-3 can be the names of elementary items or group items. They, however, cannot be items of level 01, 66, 77 or 88.

(iv) Data-name-1 may not be used as a qualifier. It can only be qualified by the name of the record within which it is defined.

(v) Neither data-name-2 nor data-name-3 can have an OCCURS clause in its neither description entry, nor can they be subordinate to an item that has an OCCURS clause in its data description entry.

(vi) Data-name-3, if mentioned, must follow data-name-2, in the record and must not be one of its subfields.

14.3 REDEFINES CLAUSE

Sometimes it may be found that two or more storage areas defined in the DATA DIVISION are not in use simultaneously. In such cases only one storage area can serve the purpose of two or more areas if the area is redefined. The REDEFINES clause used for the purpose allows the said area to be referred to by more than one data name with different sizes and pictures. Let us consider the following example.

```
01   SALES-RECORD
     02   SALES-TYPE          PIC          9.
     02   SALES-BY-UNTI.
          03   QTY            PIC          9(4).
          03   UNIT-PRICE     PIC          9(8) V99.
     02   TOTAL-SALES         REDEFINES    SALES-BY-UNIT
          03   AMOUNT         PIC          9(10) V99.
          03   FILLER         PIC          X (2).
```

This example describes a sales record which may either contain the total amount of sale (AMOUNT) or the quantity (QTY) and UNIT-PRICE. The purpose of such description may be to have two types of records and their types may be determined from the data item named SALES-TYPE. Depending on some predetermined values of SALES-TYPE the record will be interpreted in one of the two forms, Note that SALES-BY-UNIT and TOTAL-SALES refer to the same storage space. They really represent two different mappings of the same storage area.

The REDEFINES clause as illustrated above are quite common in use. However, the clause may be simply used for the purpose of conservation of storage space possibly in the working-storage section. In such cases two records having no meaningful connection between them can also be used to share same storage space provided both of them are not used in the program simultaneously. The syntax of the REDEFINES clause is as follows:

Level-number daa-name-1 REDEFINES data-name-2

14.4 RULES OF REDEFINES CLAUSE

The following rules govern the use of the REDEFINES clause:

(i) The level-number of data-name-1 and data-name-2 must be identical.

(ii) Except when the REDEFINES clause is used to 01 level, data-name-1 and data-name-2 must be of same size. In the case of 01 level, the size of data-name-2 must not exceed
that of data-name-2 (originally defined area).

(i) Multiple redefinition is allowed. The entries giving the new descriptions must Immediately follow the REDEFINES entry. In the case of multiple redefinitions the data-name-2 must be the data-name of the entry that originally defined the area.

(ii) The REDEFINES clause must immediately follow data-name-1.

(iii) Entries giving new descriptions cannot have VALUE clauses(except in the case of condition-names, i.e., 88-level). This means that data-name-1 or any of this subordinate
Must not have any VALUE clause.

(iv) The REDEFINES clause must not be used for records (01 level) described in the FILE SECTION. The appearance of multiple 01 entries in the record description is implicitly assumed to be the redefinition of the first 01-level record.

(v) This clause must not be used for level-number 66 or 88 items.

14.5 LET US SUM UP

This lesson has taught the learner

- Renames Clause and
- Redefines Clause

As many applications of COBOL require renaming and redefining the existing groups, the knowledge about these two clauses is of critical importance.

14.6 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Explain the role of RENAMES clause.
2. Specify the rules of RENAMES clause.
3. What is meant by Redefines clause?
4. Bring out the rules of REDEFINES clause.
5. What level numbers are used for Renames and Redefines clauses?

14.7 POINTS FOR DISCUSSION

1) Discuss the usage of RENAME and REDEFINES clauses.

14.8 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 15: PROGRAMS

CONTENTS

15.0 AIMS AND OBJECTIVES

The learner will now be exposed to the complete programs that make use of IF,IF-ELSE,PERFORM statements, REDEFINES and RENAMES clauses.

15.1 PROGRAM FOR IF

Write a program to check whether the given number is ODD or EVEN.

```
Identification division.
Program-id. Ifst.
Environment division.
Data division.
Working-storage section.
01 n pic 9(3) value 0.
01 q pic 9(3) value 0.
01 r pic 9(3) value 0.
Procedure division.
Para-1.
        Display(1 1) erase.
        Display( 3 5) " Enter a Number :".
        Accept n.
        Divide n by 2 giving q remainder r.
        If  (r = 0)
            Display(10 5) " EVEN NUMBER"
            Go to end-para.
        Display(10 5) " ODD NUMBER"
End-para.
        Stop run.
```

15.2 PROGRAM FOR IF –ELSE

Write a program to check whether the given number is ODD or EVEN. Use IF..ELSE.

```
Identification division.
Program-id. IfElse.
Environment division.
Data division.
Working-storage section.

01 n pic 9(3) value 0.
01 q pic 9(3) value 0.
01 r pic 9(3) value 0.

Procedure division.
Para-1.
        Display(1 1) erase.
        Display( 3 5) " Enter a Number :".
        Accept n.
        Divide n by 2 giving q remainder r.
        If  (r = 0)
            Display(10 5) " EVEN NUMBER"
        else
            Display(10 5) " ODD NUMBER".
         Stop run.
```

15.3 PROGRAM FOR SIMPLE PERFORM

Write a program to demonstrate simple PERFORM statement

```
Identification division.
Program-id.  Perf1.
Environment division.
Data division.
Procedure division.
Para-1.
        Display(1 1) erase.
        Display "ABC".
        Perform  para-2.
        Display "XYZ".
        Stop run.
Para-2.
         Display "DEF".
```

Explanation : The output of the program will be

```
ABC
DEF
XYZ
```

15.4 PROGRAM FOR PERFORM…THRU

Write a program to demonstrate PERFORM …THRU statement

```
Identification division.
Program-id.  Perfthru.
Environment division.
```

```
Data division.
Procedure division.
Para-1.
        Display(1 1) erase.
        Display "ABC".
        Perform  para-2 thru para-4.
        Display "XYZ".
        Stop run.
Para-2.
         Display "DEF".
Para-3.
        Display "GHI".
        Display " Gandhi".
Para-4.
        Display "Bharathiar".
```

The output of the above program will be

```
ABC
DEF
GHI
Gandhi
Bharathiar
XYZ
```

15.5 PROGRAM FOR PERFORM …UNTIL

Write a program to find the sum of "n" natural numbers.

```
Identification division.
Program-id. PerfUntil.
Environment division.
Data division.
Working-storage section.
01 n pic 9(2) value 0.
01  i  pic 9(2) value 1.
01 sum pic 9(4) value 0.

Procedure division.
Para-1.
        Display(1 1) erase.
        Display(5 5) "Enter a Number ".
        Accept n.

         Perform calc-para until i > n.
         Display(10 5) " Sum = " sum.
         Stop run.

Calc-para.
        Compute sum = sum + i.
         Add 1 to i.
```

15.6 PROGRAM FOR PERFORM … VARYING

Write a program to demonstrate PERFORM …VARYING

```
Identification division.
Program-id. PerfVary.
Environment division.
Data division.
Working-storage section.
01 n pic 9(2) value 0.
01  i  pic 9(2) value 1.
01 sum pic 9(4) value 0.

Procedure division.
Para-1.
        Display(1 1) erase.
        Display(5 5) “Enter a Number “.
        Accept n.
         Perform calc-para varying i from 1 by 1 until  i > n.
         Display(10 5) “ Sum = “ sum.
         Stop run.

Calc-para.
        Compute sum = sum + i.
```

15.7 PROGRAMS FOR REDEFINES CLAUSE

Write a simple program to explain REDEFINES clause at 01 level.

```
identification division.
program-id. Redef.
environment division.
data division.
working-storage section.
01  a  pic 9(3) value 125.
01  r  pic 9(2) redefines a.

procedure division.
p-1.
        display(1 1) erase.
        display(5 5) "Value of r = " r.
        stop run.
```

Note : The output of the program will be

Value of r = 12.

This output is arrived using Redefining the variable a.

Another Program for Redefines

Write a program to explain REDEFINES clause at level other than 01.

```
identification division.
program-id.
environment division.
data division.
working-storage section.
01  emp-details.
```

```
02  emp-rec.
        03 emp-name pic x(20).
        03 emp-sal  pic 9(2).
02  pay-rec redefines emp-rec.
        03 name   pic x(20).
        03 sal    pic 9(2).

procedure division.
p-1.
        display(1 1) erase.
        display(3 5) "Name : " accept emp-name.
        display(5 5) "Salary : " accept emp-sal.
        display(7 5) "Name : " name.
        display(9 5) "Salary : " sal.
        stop run.
```

15.8 PROGRAM FOR RENAMES CLAUSE

```
identification division.
program-id.
environment division.
data division.
working-storage section.
01 emp.
   02 empdet.
        03 name pic x(25).
        03 age  pic 9(3).
        03 sal  pic 9(6).

   66 paydet renames name thru sal.

procedure division.
p1.
        display(1 1) erase.
        display(5 5) "Name : ".
        accept name.
        display(7 5) "Age : ".
        accept age.
        display(9 5) "Sal : ".
        accept sal.

        display(1 1) erase.

        display(5 5) "Details (Renaming used) : " paydet.
        stop run.
```

15.9 LET US SUM UP

Having tried the above programs the learner gets clear ideas about the working principles of IF, IF –ELSE,PERFORM,REDEFINES and RENAMES clauses.

The Learner can make simple modifications in the above programs to see the consequences on the screen which will make him/her confident to use these important verbs in his/her programs without any ambiguity.

15.10 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Write a program to find whether the given number is +ve or not.
2. Write a program to display your name 5 times using Perform Times.
3. Write a program to find the factorial of a given number.
4. Write a program to check whether the given number is prime or not.

15.11 POINTS FOR DISCUSSION

1) Write a simple COBOL program to illustrate PERFORM . . . . . THRU verb.

2) Explain PERFORM . . . . . UNTIL verb with an example.

15.12 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

UNIT – IV

LESSON – 16: SEQUENTIAL FILES

CONTENTS

16.0 AIMS AND OBJECTIVES

The aim of the proposed lesson is to introduce the concepts like File characteristics, File-Control Entries For Sequential Files, File Description for Fixed-Length Records, Statements For Sequential Files and Sequential Files With Variable-Length Records. The objective in this lesson is to explain how to create and read a tape or disk file. A magnetic tape file, such as a card or printer file, can only have a sequential organization.

16.1 FILE CHARACTERISTICS

The task of file handling is the responsibility of the system software known as IOCS (Input-Output control system).

Record Size:

We know that the records of a card file must consist of 80 characters(this can vary with the model and make of the printer). The size of the records in a tape or disk file, on the other hand, may be chosen by the programmer.

Block Size:

The usual practice is to group a number of consecutive records to form what is known as a block or physical record. The number of records in a block is often called “blocking factors”.

The IOCS reserves a memory space equal to the size of a block of the file. This memory space is known as the buffer.

Buffers:

Modern computers are capable of handling I-O operations independent of the CPU by means of the hardware known as data channel.

For example, if two buffers are allocated for an input file, the IOCS can fill-in one buffer while the program processes the records already read and available in another buffer.

Label Records / Disk Directory:

The most important of the information stored in the header label is what is known as the file title. In the case of magnetic-disk files the labels usually do not exist(there are many exceptions). Since, more than one file is stored on a disk pack, the IOCS also maintains a disk directory for all the files.

16.2 FILE-CONTROL ENTRIES FOR SEQUENTIAL FILES

The characteristics of each of the files handled in a program are specified in the ENVIRONMENT DIVISION and DATA DIVISION.

```
SELECT   [ OPTIONAL ]   file-name   ASSIGN   TO   hardware-name

        ( ; RESERVE   integer-1  { AREA  }  )
                                 { AREAS }

        [ ; ORGANIZATION   IS   SEQUENTIAL ]

        [ ; ACCESS  MODE  IS  SEQUENTIAL ]
       [ ; FILE   STATUS   IS   data-name-1 ]
```

RESERVE clause:

This clause specifies the number of buffers to be used for the file.

Integer-1 indicates this number.

ORGANIZATION/ACCESS clause:

These two clauses indicate that the said file is organized as a sequential file and will be accessed sequentially.

FILE STATUS clause:

This clause has been included in the above syntax for completeness. The REVERSE, ORGANIZATION, ACCESS and STATUS clause can be specified in any order.

16.3 FILE DESCRIPTION- FIXED-LENGTH RECORDS:

The general characteristics of a file are described in the file description (FD) entry of the DATA DIVISION.

```
FD    file-name
      (                                                          )
      ; BLOCK CONTAINS   integer-1  { RECORDS    }
                                    { CHARACTERS }

      ( ; RECORD   CONTAINS   integer-2   CHARACTERS )
```

```
[ ; LABEL { RECORD IS / RECORDS ARE } { STANDARD / OMITTED } ]
```

BLOCK CONTAINS CLAUSE:

Integer –1 of the BLOCK CONTAINS clause specifies the block size either in terms of records or in terms of characters. For example, BLOCK CONTAINS 50 RECORDS means that there are 50 records in the block.

RECORDS CONTAINS CLAUSE:

This clause specifies the records size. Integer-2 specifies the number of characters in a record. The RECORD CONTAINS clause is used for documentary purposes only.

LABEL RECORD CLAUSE:

This clause specifies whether or not the standard header and trailer labels should be present in the magnetic-tape files.

VALUE OF CLAUSE:

The VALUE OF clause is entirely implementation-dependent. In most compilers this clause is used to specify a file title. The clause in such cases has the form.

```
VALUE OF { ID / IDENTIFICATION } IS { data-name / literal }
```

The following are examples of the VALUE OF clause.

```
VALUE  OF  IDENTIFICATION  IS  "FILEA"
VALUE  OF  ID  MY-FILE
```

DATA RECORD CLAUSE:

This clause documents the record names defined for the file. For example, DATA RECORDS ARE REC-1, REC-2, REC-3 means that there are three different record descriptions following the FD entry in which this DATA RECORDS clause is used.

CODE-SET CLAUSE:

The CODE-SET clause is used to describe the code in which the data is recorded on the external medium. The alphabet name must be specified in the SPECIAL-NAMES paragraph. The format is as follows:

```
Alphabet-name  IS  implementor-name
```

NONSTANDARD CLAUSES:

The different clauses described above are as per the ANSI standard. Most compilers also provide for additional nonstandard clauses to meet the specific requirements of the corresponding computer.

Example of file-description entries:

```
FD  FILE – A
    RECORD CONTAINS 130 CHARACTERS
    BLOCK CONTAINS 20 RECORDS
    DATA RECORD IS FIRST-RECORD
    LABEL RECORDS ARE STANDARD
    VALUE OF ID IS "MY-LIFE".

01  FIRST-RECORD  PIC  X(130).
```

16.4 STATEMENTS FOR SEQUENTIAL FILES

Basic operations on a file involve the reading and writing of its records.

When the file is sequential, there are three verbs for the purpose. These are READ, WRITE and REWRITE.

OPEN statement:

We know that the processing of a file should begin with the execution of an OPEN statement. A file can be opened in any one of the four open modes – INPUT, OUTPUT, EXTEND and I-O. The following is the syntax (simplified) of the OPEN statement.

```
OPEN  { INPUT  }
      { OUTPUT }   file-name-1   [ , file-name-2 ] …
      { EXTEND }
      { I-O    }

( { INPUT  }
  { OUTPUT }   file-name-3   [ , file-name-4 ] …   … )
  { EXTEND }
  { I-O    }
```

CLOSE statement:

The following is the syntax (simplified) of the CLOSE statement.

```
CLOSE   file-name-1  [WITH  LOCK  ]

        [ , file-name-2  [WITH  LOCK  ]  ] …
```

The CLOSE statement terminates the processing of the file.

WRITE statement:

The WRITE statement for tape and sequential-disk files has the following syntax.

```
WRITE   record-name   [ FROM  identifier ]
```

As a result of the execution of the WRITE statement, the record is released from the record area and is written onto the file.

REWRITE statement:

The REWRITE statement is used to update an existing record in the disk file.

The general format is as follows:

REWRIE record-name [FROM identifier]

It may be noted that the syntax is similar to that in the case of a WRITE statement and the FROM option has the same meaning.

16.5 SEQUENTIAL FILES WITH VARIABLE-LENGTH RECORDS

Magnetic - tape or disk files can contain variable-length records. In this case the file can have records with different fixed lengths or one or more records can contain variable number of table elements. In the latter case the table elements are defined with the OCCUR….DEPENDING clause.

FD Entry for variable-length Records:

The RECORDS CONTAINS and BLOCK CONTAINS clause are quite different in the case of files with variable-length records. The syntax of these two clauses are as follows:

BLOCK CONTAINS [integer-1 TO] integer-2 { RECORDS / CHARACTERS }

RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS

Record Description for Variable-length Records:

When the variable-length records consists of records of different lengths, each record type is to be described at level 01 following the FD entry for the file. The variable part is to be defined with the OCCURS…DEPENDING clause.

```
FD VARIABLE-FILE
        RECORD CONTAINS 40 TO 92 CHARACTERS
        BLOCK CONTAINS 933 TO 1024 CHARACTES
        LABEL RECORDS AERA ATANDARDA
        RECORDING MODE IS V
        VALUS OF INDENTIFICATION IS “VARFILE”.

01 VARIABLE-RECORD.
        02 ACCOUNT-NUMBER    PIC 9(6).
        02 NAME  PIC  X(20).
        02 NO-OF-PAYAMENTS    PIC 9.
        02 PAYMENTS    OCCURS 1 TO 5 TIMES DEPENDING ON
                NO-OF-PAYMENTS   INDEXED BY TAB-INDEX.

                03 DATE-OF-PAYAMENT   9(6).
                03 AMOUNT-OF-PAYMENT          9(5)V99.
```

16.6 LET US SUM UP

This lesson has introduced the learner the concepts like File characteristics, File-Control Entries For Sequential Files, File Description for Fixed-Length Records, Statements For Sequential Files and Sequential Files with Variable-Length Records. Having learnt all these concepts the learner gets knowledge about sequential file entries in different forms.

16.7 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Specify the file characteristics.
2. Bring out file-control entries for sequential files.
3. Bring out the file description entries for fixed length records.
4. With syntax explain OPEN,WRITE statements of sequential file.
5. With syntax explain REWRITE,CLOSE statements of sequential file.

16.8 POINTS FOR DISCUSSION

1) Explain in detail about the file manipulation statements for a sequential file.

2) Write notes on different modes of file operation.

16.9 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 17: DIRECT ACCESS FILES

CONTENTS

17.0 AIMS AND OBJECTIVES

The aim of this lesson is to introduce the concepts related with relative files. It also exposes the learner to the file-control and procedure division statements that are required for relative files.

17.1 RELATIVE FILES

Files which are stored on a direct access storage medium such as a magnetic disk, are often called direct access files. COBOL supports three different organization for disk files-sequential, relative and index sequential.

A relative file is a magnetic-disk file organized in such a way that each record is identified by a relative record number. The relative record number specifies the position of the record from the beginning of the file. Thus the relative record number 1identifies the first record, the relative record number 2 identifies the second record and so on.

A relative file can be access either sequentially or randomly. When the file is accessed sequentially the records are accessed in the increasing order of their relative record numbers. When a file is accessed randomly, the programmer must specify the relative record number.

It may be noted that in the case of relative organization, the reading as well as the writing can be done randomly. Thus when a file is created by writing the record randomly, some of the record position may remain empty. While these positions can be filled in sub sequentially, the programmer should avoid specifying these empty positions while reading such as relative file randomly. If a relative file is read in a sequential manner, such empty position within it, if any, are ignored.

The handling of relative files requires some special codes in the FILE-CONTROL paragraph as well as in the PRCEDURE DIVISION. These are discussed bellow.

17.2 FILE-CONTROL paragraph for relative files:

The general format for the SELECT clause for a relative file is as follows.

SELECT file-name ASSIGN TO implementor-name

[; RESERVE integer-1 AREA / AREAS]

; ORGANIZATION IS RELATIVE

; ACCESS MODE IS SEQUENTIAL [, RELATIVE KEY IS data-name-1]
RANDOM
DYNAMIC , RELATIVE KEY IS data-name-1

[; FILE STATUS IS data-name-2]

Whether the file should be used sequentially or randomly, should specified through the word SEQUNTIALLY or RANDOMLY in the access mode clause the clause ACCESS MODE IS DYNAMIC it indicates that the file is accessed sequentially and / or randomly in the PROCEDURE DIVISION.

17.3 PROCEDURE DIVISION statements for relative files

The statements OPEN,CLOSE,READ,WRITE and REWRITE which are available for sequential files are available for the relative files. In addition, two other words, namely, DELETE and START are also available. As regards the OPEN and CLOSE statements, There is no difference between relative file and sequential disk file.

READ STATEMENTS:

The general format for the read statements are shown bellow.

Format 1:

READ file-name RECORD [INTO identifier]
[; AT END imperative-statements]

Format 2:

READ file-name RECORD [INTO identifier]
[; INVALID KEY imperative-statement]

Format 3:

READ file-name [NEXT] RECORD [INTO identifier]
[; AT END imperative-statements]

As usual, a READ statements reads a record of the file. The file must be open in either the input or I-O mode.

Format 1 is the normal form of the READ statements.

Format 2 is used when the access mode is either random or dynamic. For example, suppose REL-KEY are the names for the relative file and the relative key data item respectively. The following statements will read the 50th record from this file.

```
MOVE   50   TO   REL-KEY.
READ   REL-FILE  RECORD  INVALID-KEY   GO  TO  PARA-INVALID.
```

Format 3 of the read statement can be used when the access mode is dynamic and the records are to be read sequentially. The next record is identified according to the following rules:

(i) When the READ NEXT statements is the first statement to be executed after the open statement on the file, the next record is the first record of the file.

(ii) When the execution of the READ NEXT statement follows the execution of another READ statement on the same file (format 2 or format 3 above), the next record is the record following the one previously read.

(iii) When the execution of the READ NEXT statement follows the execution of the start statement .

WRITE Statement

The WRITE statement for a relative file has the following format.

```
WRITE    record-name   [ FORM   identifier ]
                       [ ;  INVALID KEY  imperative statement  ]
```

At the time of execution of the WRITE statement, the file must be open either in the OUTPUT or I-O mode. For example, suppose REL-OUTPUT AND REL-KEY are the record name and relative key data item name for a relative file opened in the I-O mode. Then, upon execution of the following statements

```
MOVE    50  TO  REL-KEY.
WRITE REL-OUTPUT  INVALID KEY GO TO PARA-INVALID.
```

The record is written at the 50^{th} record position on the file.

The imperative statement of the INVALID KEY phrase is execution in the following case:

(i) when an attempt is made to write beyond the externally-defined boundaries of the file.

(ii) When an attempt is made to write in the record position which already contains a valid record.

REWRITE STATEMENT:

The REWRITE statement has the following format for a relative file,

```
REWRITE  record-name  [ FORM identifier ]
                      [ ;  INVALID  KEY   imperative –statement  ]
```

The REWRITE statement is used to replace an exiting record by the contents of the record specified in the record name. The file must be opened in the I-O mode.

DELETE STATEMENT:

The format of the delete statement is as follows:

```
DELETE   file-name  RECORD   [ ;  INVALID   KEY  imperative-statement  ]
```

When the ACCESS MODE IS SEQUENTIAL the execution of the DELETE statement must be preceded by the execution of a READ statement on the file and the INVALID KEY phrase should not be specified.

17.4 LET US SUM UP

In this lesson the learner has been introduced the concepts related with relative files. It has also exposed the learner to the file-control and procedure division statements that are required for relative files. Having learnt these concepts the learner can write programs that involve relative files comfortably.

17.5 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. What is meant by relative file?
2. Explain file control paragraph of relative file
3. Explain READ statement of Relative File.
4. Explain WRITE statement of Relative File.
5. Explain REWRITE statement of Relative File.

17.6 POINTS FOR DISCUSSION

1) Discuss in detail about the procedure division statements for a relative file.

2) Differentiate WRITE statement from RE-WRITE statement.

17.7 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 18: INDEXED SEQUENTIAL FILES

CONTENTS

18.0 AIMS AND OBJECTIVES

The aim of this lesson is to introduce the Indexed Sequential Files, File-Control Paragraph for Indexed Files, Procedure Division Statements for Indexed Files, Updating of Relative and Indexed Files**.**

18.1 INDEXED SEQUENTIAL FILES

In indexed sequential files (also referred to as indexed files), the records are stored in the key sequence order (usually ascending order). In addition, some index tables are also created and maintained with the file.

An indexed file on COBOL can be accessed either sequential or randomly. However, while creating an indexed file the records can be written only sequentially and in the ascending order of the key. When an indexed file is accessed randomly, the sequence in which the records are accessed is controlled by the programmer by specifying the value of a data item called record key.

Indexed files in a COBOL program can be handled through suitable special codes in the FILE-CONTROL paragraph and in the PROCEDURE DIVISION. These are described below.

18.2 FILE-CONTROL paragraph for indexed files

The general format for the SELECT clause for an files is as follows:

SELECT file-name ASSIGN TO implementor –name

[; RESERVE INTEGER-1 [AREA / AREA]]
, ORGANIZATION IS INDEXED

[; ACCESS MODE IS { SEQUENTIAL / RANDOM / DYNAMIC }]
; RECORD KEY IS data-name-1

[; ALTERNATIVE RECORD KEY IS data-name-2 [WITH DUPLICATES]] .....
[; FILE STATUS IS data-name-3]

The ORGANIZATION clause indicates that the file is an indexed file. The RECORD KEY clause specifies the record key data item on the basis of which the file is sequenced. The field which is specified in the RECORD KEY clause (data-name-1) is also known as the primary key. While the files is stored and stored on the basis of the prime key, the records of an alternative key.

18.3 PROCEDURE DIVISION statements for indexed files

All the statements that are available for a relative file are also available for the indexed files.

READ STTATEMENT

When either the RANDOM or DYNAMIC access mode is specified and the records are to be read in a random manner, the syntax is as follows:

READ file-name RECORD [INTO IDENTIFIER]
[; KEY IS data-name]
[; INVALID KEY imperative-statement]

The data name in the KEY IS phrase must be either the prime key or the alternative key item. If the phrase is not specified, the prime key is assumed. Let PERSONNEL be an indexed file and let EMP-NO be the prime key and NAME the alternative key.

WRITE STATEMENT

The records are written to be logical position as determined from the value of the record key. The INVALID KEY condition arises in the following cases:

(i) when an attempt is made to write a record beyond the externally defined boundaries of the file.
(ii) When the file is opened in the OUTPUT mode and the value of the record key is not grater than the value of the record key for the previous record written.
(iii) When the value of the record key is equal to the record key of a record key already present in the file.

REWRITE STATEMENT

As in the case of a relative file, the REWRITE statement requires that the file must be opened in the I-O mode, and if the SEQENTIAL access mode is specified, the value of the record key of the record being replaced must be equal to that of the record last from this file. The INVALID KEY condition arises in the following cases:

(i) when the record key does not match that of an existing record in the file.
(ii) For SEQUENTIAL access, when the value of the record key is not identical to that of the last record read from the file.

DELETE STATEMENT

The file must be opened in the I-O mode. If the access is SEQUENTIAL, the INVALID KEY phrase should be specified. Instead, the last input-output statement executed on the file must be a successful READ statement for the said record.

START STATEMENT

The START statement positions the files to the first logical record whose record key satisfies the condition specified by the KEY phrase. The access mode must be SEQUENTIAL or DYNAMIC and the file must be opened in the I-O mode.

18.4 UPDATING OF RELATIVE AND INDEXED FILES

Sequential files are updated by creating a new maser file from an existing old master file and a transaction file. Such an Updating is known as updating by copy. Direct access files can be updated by the technique known as Updating by overlay. In this case no new file is created. instead, the necessary changes are incorporated in the body of the file.

The transaction code (T-CODE) is a one-digit code having the following meaning.

Transaction Code	Meaning
1	The transaction record is to be inserted
2	The corresponding master record is to be deleted.
Other than 1 or 2	The master record is to be replaced by the transaction record.

File Description for Relative and Indexed Files

The FD entry for a relative or an indexed file is identical to that of a sequential file. Some compilers do not allow variable-length record or the blocking of records in the case of direct access files.

DECLARATIVE and FILE STATUS Clause

The input-output exception condition in the case of a direct access files can be handled by a declarative procedure in a manner similar that of sequential files.

Direct Organization

Besides the relative or indexed organization, a direct access file can also be designed to have what is known as direct organization. In this organization, data records are stored or

"accessed" using a scheme of converting the of a record into the disk address to which the record is placed. Thus, no index table as in the case of an indexed file is necessary.

Selection of file Organization

While designing a file, the programmer must select a suitable organization for a file. The order in which the choice is to be made is as follows:

(i) Implementation support difficulty.
(ii) Software support required.
(iii) Efficiency of processing.

File Activity

The file activity is a measure of the proportion of records processed during a update run. Thus we define the activity ration as follows.

Activity ratio = m / n

Where m = number of records to be inserted, modified or deleted and
n = number of records in the file.

File volatility

File volatility relates to the number of times the updating of records are required during some time period.

File interrogation

Some files contain reference data. These files are used mainly for the purpose of interrogation. interrogation means a reference to a specific record for a specific response without changing the record in any manner.

Eg: Price list can be file which is to be constantly interrogative during a billing run.

18.5 LET US SUM UP

With the help of this lesson, the learner gets clear ideas about the concepts related with Indexed Sequential Files. File-Control Paragraph for Indexed Files, Procedure Division Statements for Indexed Files, Updating of Relative and Indexed Files. Having leant these concepts , the learner can write application programs that involve indexed sequential files.

18.6 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Explain about Indexed Sequential files.
2. Explain the file control Para of Indexed Sequential files.
3. Explain READ,WRITE statements of Indexed Sequential file.
4. Explain REWRITE,DELETE statements of Indexed Sequential file.
5. What parameters are to be considered for selecting file organization?

18.7 POINTS FOR DISCUSSION

1) Explain in detail about procedure division statements for indexed sequential files.

18.8 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 19: SORTING AND MERGING OF FILES

CONTENTS

19.0 AIMS AND OBJECTIVES

The aim of this lesson is to introduce the learner the Sort Verb, File Updating, Simple Merge Verb, Input and Output Procedure in Sort Statement and Merge Verb with Output Procedure ,as they play important role in many application programs.

19.1 THE SIMPLE SORT VERB

The process of sequencing the records in some desired manner is known as sorting. Sorting is done upon some key data item in the record. For example, consider the case of a pay roll file where each record contains all the necessary information of an employee, such as his identification number, name, address, department number, basic pay, allowances, deductions, etc.

When a sequential file is to be sorted, its record reside on the file medium and can be accessed to only serially. In COBOL, there is no specific feature for the sorting of a table.

However, its provides a sort verb that can be used to SORT a sequential file. In addition to the sort verb, the MERGE verb can be used to merge several sorted files to create a new file containing the records of these files in the sorting order.

The sort verb like many other Cobol verbs, then have different forms. This form is to be used when it is required to sort a given input file. The simple sort verb requires the naming of three files – the unsorted input file, the sorted output file and the work file. The format of the simple SORT verb is as follows:

SORT file –name-1 { ACENDING DESCENDING} KEY data-name-1 [,data-name-2]….
[ON { ASCENDING DESCENDING} KEY data-name-3 [,data-name-4]…]….
USING file-name-2 GIVING file-name-3.

The work file is to be defined by a sort description entry(SD entry).The format of SD entry is as follows.

SD file-name
[; RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS]

[; DATA { RECORD IS / RECORDS ARE } data-name-1 [, data-name-2]….]

The following rules should be taken into considerations while specifying this sort verb

(i) The input, output as well as the work file are open by the sort statement before the starting begins and are closed by the sort statement itself after the sorting is over.
(ii) There can be any number of SORT statement in a program.
(iii) The sorting can be done on any number of keys.
(iv) All the keys on which the sorting is done, must appear with their description in the record description of file name1.
(v) Keys in the sort statement do not require any qualification.
(vi) When two or more records in the input file have identical keys.
(vii) The SELECT clauses for the work file file- name-1 is SELECT file-name-1 ASSIGN TO hardware-name.

Eg: Assume that we have a card file with the following records description in the data division.

```
FD   KARD-FILE.
     01   INPUT- RECORD.
          02   ID-NUMBER          PIC      9(6).
          02   NAME                PIC      X(24).
          02   DEPARTMENT         PIC      X(10).
          02   BASIC-PAY          PIC      9(5)V99.
          02   ALLOWANCE          PIC      9(4)V99.
          02   DETECTION          PIC      9(4)V99.
```

The names of the work file and output file be SORT-FILE and OUTPUT-FILE respectively. the DATA DIVISION entries for these two files are as follows.

```
SD   SORT-FILE.
     01   SORT-RECORD
          02   FILLER             PIC      X(30).
          02   DEPARTMENT         PIC      X(10).
          02   BASIC-PAY          PIC      9(5)V99.
          02   FILLER             PIC      X(12).

FD   OUTPUT-FILE.
     01   OUTPUT-RECORD          PIC      X(59).
```

The following statement will sort the input file and will create the sorted output file.

19.2 FILE UPDATION

The process of modifying an old file with current information is known as file updating.

Master file

A master file is a file that is used to as an authority in a given job. It may contain somewhat permanent, historical, statistical or identification type of data.

Transaction file

A transaction file is a file that contains a new records are changes to old records which are used to update the master file.

The problem of file updating can be defined as follows.

- ❖ Insertion of new records.
- ❖ Modification of some existing records.
- ❖ Deletion of obsolete records.
- ❖ Copy of those records which are neither obsolete nor require any modification.

19.3 SIMPLE MERGE VERB

Like sorting, the merging of files is frequently required in various commercial application. It is possible to merge two or more files with one MERGE statement. The format of the simple MERGE verb is as follows.

```
MERGE  file-name

ON   { ASCENDING  }     KEY    data-name-1        [ , data-name-2 ] ……
     { DECENDING  }

( ON { ASCENDING  }     KEY    data-name-3        [ , data-name-4 ]…….. )
     { DECENDING  }

USING    file-name-2,  file-name-3  [ , file-name-4 ]  …..

GIVING   file-name-5.
```

The input files to be merged through the MERGE statement are specified in the USING phrase. These files must be sequential files and must be sorted on the merge keys. The rules of the SORT statement in respect of the ASCENDING/DECENDING KEY phrase are also applicable in this case.

Let us assume there are three zones and as such three sales files are to be merged. Three files are named as ZONE-FILE-1, ZONE-FILE-2 and ZONE-FILE-3.

```
FILE CONTROL.
            SELECT   ZONE-FILE-1    ASSIGN  TO  TAPE.
            SELECT   ZONE-FILE-2    ASSIGN  TO  TAPE.
            SELECT   ZONE-FILE-3    ASSIGN  TO  TAPE.
            SELECT   WORK-FILE      ASSIGN  TO  MERGE-DISK.
            SELECT   MERGED-FILE    ASSIGN  TO  TAPE.
DATA DIVISION.
FILE  SECTION.
FD    ZONE-FILE-1
      BLOCK  CONTAINS  20  RECORDS
      VALUE  OF  ID  “ZONEFILE1”.
01    FILE-1-RECORD     PIC      X(90).

FD    ZONE-FILE-2
```

```
       BLOCK  CONTAINS  15  RECORDS
       VALUE  OF  ID  "ZONEFILE2".
01     FILE-2-RECORD     PIC      X(90).

FD     ZONE-FILE-3
       BLOCK  CONTAINS  10  RECORDS
       VALUE  OF  ID  "ZONEFILE3".
01     FILE-3-RECORD     PIC      X(90).

FD     MERGED-FILE
       BLOCK  CONTAINS  20  RECORDS
       VALUE  OF  ID  "MERGEDFILE".
01     MERGED-RECORD     PIC      X(90).

SD WORK-FILE.
01 WORK-RECORD.
   02  FILLER                  PIC          X(50).
   02  PROCEDURE-NAME          PIC          X(20).
   02   FILLER                  PIC          X(20).

PROCEDURE DIVISION.
MERGING-PARA.
            MERGE  WORK-FILE  ON  ASCENDING  KEY  PRODUCT-NAME
            USING  ZONE-FILE-1,  ZONE-FILE-2,  ZONE-FILE-3
            GIVING  MERGED-FILE.
            STOP  RUN.
```

It has been assumed that all the three input files and the final merged file named as MERGED-FILE are tape files.

19.4 INPUT AND OUTPUT PROCEDURE IN SORT STATEMENT

The general format of the SORT statement can written as follows.

```
SORT  file-name-1  ON  { ASCENDING  } KEY  data-name-1 [data-name-2]
                      { DESCENDING }

( ON { ASCENDING  } KEY  data-name-3  [,  data-name-4] ....  ..... )
     { DESCENDING }

[ COLLATING  SEQUENTIAL  IS  alphabet-name]
```

```
{ INPUT  PROCEDURE    IS    section-name-1  ( { THRU      } section-name-2 ) }
{ USING  file-name-2  [ , file-name-3] ….   ( { THROUGH   }                ) }

{ OUTPUT PROCEDURE    IS    section-name-3  ( { THRU      } section-name-4 ) }
{ GIVING file-name-4                        ( { THROUGH   }                ) }
```

When an INPUT PROCEDURE is mentioned

```
Section-name-1 ( { THRU    } section-name-2 )
               ( { THROUGH }                )
```

The format of the RELEASE statement is as follows:

RELEASE record-name [FROM identifier]

The format of the RETURN statement is as follows:

RETURN file-name RECORD [INTO identifier]
; AT END imperative statement.

The following are the restrictions when these procedure are used.

(i) Procedure must not contain any SORT/MERGE statement.
(ii) An explicit transfer of control outside the procedures is not allowed.
(iii) The control must reach the statements only through associated SORT statement.
(iv) Procedure must consist of one or more sections and they must appear contiguously in the body of the program.
(v) The input and output procedure must not shae any section or any part between them.

19.5 MERGE VERB WITH OUTPUT PROCEDURE

Like the sort verb, the merge verb can also have an output procedure. The syntax of the MERGE verb is given below.

```
MERGE  file-name-1
ON { ASCENDING  } KEY  data-name-1 [ , data-name-2 ] …..
   { DESCENDING }

( ON { ASCENDING  } KEY  data-name-3 [ , data-name-4 ] …..
     { DESCENDING }
  [COLLATING SEQUENCE IS alphabet-name ] )
```

USING file-name2, file-name-3 [, file-name-4] ….

```
( OUTPUT PROCEDURE IS section-name-1 ( { THRU      } section-name-2 )
                                      { THROUGH   }
  GIVING file-name-5                                                )
```

The rules for specifying and coding the output procedure is identical to those in the case of the SORT verb.

SAME SORT AREA CLAUSE:

Like the SAME AREA clause, this clause can also be specified in the I-O-CONTROL paragraph to have two or more files to share same memory area during execution.

```
SAME { SORT       } AREA FOR file-name-2 [ , file-name-3 ] …..
     { SORT-MERGE }
```

At least one of the files quoted in this clause must be defined with SD. The SAME SORT AREA clause enables two or more SORT/MERGE work files to use same area.

19.6 LET US SUM UP

This lesson has introduced the learner the Sort Verb, File Updating, Simple Merge Verb, Input and Output Procedure in Sort Statement and Merge Verb with Output Procedure. These are essential for any COBOL programmer for writing industry oriented applications.

19.7 Lesson-end Activities

Try to find the answers for the following exercises on your own.

1. What do you mean by Sorting?
2. Explain with syntax SORT verb
3. State the rules for SORT verb
4. Explain how input-output procedure is used with SORT statement.
5. Explain with syntax MERGE verb

19.8 POINTS FOR DISCUSSION

1) Write a simple COBOL program to illustrate the SORT verb.

2) Discuss the usage of MERGE verb with example.

19.9 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, Tata McGraw Hill , 4th Edition.

LESSON – 20: PROGRAMS

CONTENTS

20.0 AIMS AND OBJECTIVES

The aim of this lesson is to introduce the learner the programs for sequential files and indexed sequential files. It also focuses the programs that involve the concept of Sorting and Merging.

20.1 SEQUENTIAL FILE CREATION AND REWRITING

Write a program to create a student file with just two fields : sno (Student Number) and same (Student Name). Add a few records. Modify the record with sno=1 as sno=10. Use sequential file I-O.

```
identification division.
program-id. s2.
environment division.
input-output section.
file-control.
        select stu-file assign to disk
        organization is line sequential
        access mode is sequential
        file status is fs.
data division.
file section.
fd stu-file
        label records are standard
        value of file-id is 'stu.dat'
   data record is stu-rec.

01  stu-rec.
        02  sno   pic 9(2).
        02  sname pic x(10).
working-storage section.
```

```
01   ans pic x value space.
01   fs  pic x(2) value spaces.
01   eof pic x value space.
procedure division.
p-1.
        display(1 1) erase.
        open extend stu-file.
        perform g-w-para until ans = "n".
        close stu-file.

        open i-o stu-file.
           if fs = '30'
                        open output stu-file
                        close stu-file
                        open i-o stu-file.
        read stu-file at end move 'y' to eof.
        perform rewrite-para until eof = 'y'.
        close stu-file.
        stop run.
g-w-para.
        display(1 1) erase.
        display(3 5) "Sno : ".
        accept sno.
        display(5 5) "Sname : ".
        accept sname.
        write stu-rec.
        display(10 5) "Continue [ y/ n ] : ".
        accept ans.

rewrite-para.
        if sno="01"
                move 10 to sno
                rewrite stu-rec.
        read stu-file at end move 'y' to eof.
```

20.2 Program for Sequential File Creation & Rewriting

Write a program to create a Length file with just two fields : l (Length) and l-c (Length-Code) . Add a few records. Modify the record with l-c =1 as l-c=5. Use sequential file I-O.

```
identification division.
program-id.
environment division.
input-output section.
file-control.
        select len-file assign to disk.
data division.
file section.
fd len-file
        label records are standard
        value of file-id is "len.dat".
```

```
01 len-rec.
        02   l   pic 9(2).
        02   l-c pic 9(1).
working-storage section.
01   ans pic x value space.
01   eof pic x value space.

screen section.
01  cls-screen.
        02  blank screen.
01  get-screen.
        02  line 3 column 5 value "Length = ".
        02   column plus 3 pic 9(2) to l auto bell.
        02  line 5 column 5 value "Code = ".
        02  column plus 3 pic 9 to l-c bell reverse-video.

procedure division.
p-1.
        open output len-file.
        perform g-w-para until ans = 'n' or 'N'.
        close len-file.

        open i-o len-file.
        read len-file at end move 'y' to eof.
        perform rewrite-para until eof = 'y'.
        close len-file.
        stop run.

g-w-para.
        display cls-screen.
        display get-screen.
        accept get-screen.
        write len-rec.
        display (10 5) "Continue [y/n] :".
        accept ans.

rewrite-para.
        if l-c = 1
                move 5 to l-c
                rewrite len-rec.
        read len-file at end move 'y' to eof.
```

20.3 PROGRAM FOR INDEXED SEQUENTIAL FILE CREATION (DYNAMIC MODE)

Write a program to create an Indexed Sequential File in dynamic mode for Student particulars. Assume just 3 fields : rno(Roll Number), cl (Class) and m(Mark). Read the file and display the records.

```
identification division.
program-id.
environment division.
input-output section.
file-control.
        select stu-file assign to disk
        organization is indexed
        access mode is dynamic
        record key is rno
        file status is fs.

data division.
file section.
fd stu-file
        label records are standard
        value of file-id is 'stu.dat'.
01 stu-rec.
        02  rno   pic 9(3).
        02  cl pic x(4).
        02  m  pic 9(3).
working-storage section.
01  ans pic x value space.
01  a-rno pic 9(3) value 0.
01  fs  pic x(2) value spaces.
procedure division.
p-1.
        open i-o  stu-file.
                if fs = "30"
                        open output stu-file
                        close stu-file
                        open i-o stu-file.
        perform g-w-para until ans = 'n'.
        go to p-2.
g-w-para.
        display(1 1) erase.
        display "Enter Data :".
        accept rno.
        accept cl.
        accept m.
        write stu-rec invalid key
                        display "Record Exists!".
        display "Continue [y/n] : ".
        accept ans.
p-2.
        display(1 1) erase.
        display(3 5) "Give Roll No : ".
        accept a-rno.
        move a-rno to rno.
        read stu-file  key is rno
                invalid key
```

```
                    display(10 5) "No Record Found"
                    go to c-para.
            display(5 5) "Rno = " rno.
            display(7 5) "Class = " cl.
            display(9 5) "Mark = " m.
c-para.
            display(20 5) "Continue [y/n]: ".
            accept ans.
            if ans = 'y' or 'Y' go to p-2.
            close stu-file.

            stop run.
```

20.4 PROGRAM FOR INDEXED SEQUENTIAL FILE CREATION (RANDOM MODE)

Write a program to create an Indexed Sequential File in random mode for Student particulars. Assume just 2 fields : rno(Roll Number), name(Name of Student)

```
identification division.
program-id.
environment division.
input-output section.
file-control.
        select stu-file assign to disk
        organization is indexed
        access mode is random
        record key is rno
        file status is fs.
data division.
file section.
fd stu-file
        label records are standard
        value of file-id is "stu.dat".
01 stu-rec.
        02 rno pic 9(3).
        02 name pic x(20).
working-storage section.
01 fs pic x(2) value spaces.
01 ans pic x value space.

procedure division.
p-1.
        open i-o stu-file.
        if fs = "30"
                open output stu-file
                close stu-file
                open i-o stu-file.
        perform g-w-para until ans = "n".
        close stu-file.
        stop run.
g-w-para.
```

```
        display(1 1) erase.
        display(3 5) "Rno : ".
        accept rno.
        display(5 5) "Name : ".
        accept name.
        write stu-rec invalid key display(15 5) "Error!".
        display(20 5) "Continue [y/n] : ".
        accept ans.
```

20.5 PROGRAM TO DEMONSTRATE SORT VERB

A file for which a record having 2 fields, namely, Account Number and Name is already available. Sort the file based on the ascending order of Account Number.

```
identification division.
program-id.
environment division.
input-output section.
file-control.
        select o1-file assign to disk
        organization is line sequential.

select s1-file assign to disk
        organization is line sequential.

        select w-file assign to disk.
data division.
file section.
fd o1-file
        label records are standard
        value of file-id is "o1.dat".
01  o1-rec.
        02 o1-acc-no pic 9(2).
        02 o1-name pic x(4).

fd s1-file
        label records are standard
        value of file-id is "s1.dat".

01  s1-rec.
        02 s1-acc-no pic 9(2).
        02 s1-name pic x(4).

sd w-file.

01  w-rec.
        02 w-acc-no pic 9(2).
        02 w-name pic x(4).

procedure division.
p-1.
```

```
        sort w-file on ascending key w-acc-no using o1-file
                                    giving s1-file.

            stop run.
```

20.6 PROGRAM TO DEMONSTRATE MERGE VERB

Two files for which a record having 2 fields namely Account Number and Name are already available. Merge these two files and create a new file based on the ascending order of Account Number.

```
identification division.
program-id.
environment division.
input-output section.
file-control.
        select o1-file assign to disk
        organization is line sequential.

        select o2-file assign to disk
        organization is line sequential.


        select s1-file assign to disk
        organization is line sequential.

        select s2-file assign to disk
        organization is line sequential.

        select m-file assign to disk
        organization is line sequential.

        select w-file assign to disk.
data division.
file section.
fd o1-file
        label records are standard
        value of file-id is "o1.dat".
01  o1-rec.
        02 o1-acc-no pic 9(2).
        02 o1-name pic x(4).

fd o2-file
        label records are standard
        value of file-id is "o2.dat".
01  o2-rec.
        02 o2-acc-no pic 9(2).
        02 o2-name pic x(4).

fd s1-file
        label records are standard
```

```
        value of file-id is "s1.dat".
01  s1-rec.
        02 s1-acc-no pic 9(2).
        02 s1-name pic x(4).

fd s2-file
        label records are standard
        value of file-id is "s2.dat".
01  s2-rec.
        02 s2-acc-no pic 9(2).
        02 s2-name pic x(4).

fd m-file
        label records are standard
        value of file-id is "m.dat".
01  m-rec.
        02 m-acc-no pic 9(2).
        02 m-name pic x(4).

sd w-file.

01  w-rec.
        02 w-acc-no pic 9(2).
        02 w-name pic x(4).

procedure division.
p-1.

   sort w-file on ascending key w-acc-no using o1-file giving s1-file.
   sort w-file on ascending key w-acc-no using o2-file giving s2-file.
  merge w-file on ascending key w-acc-no  using s1-file  s2-file giving m-file.

stop run.
```

20.7 LET US SUM UP

With the help of the above programs, the learner gets very clear ideas about how to create a sequential file , how to rewrite records , how to cerate indexed sequential file in dynamic and random modes. Also the learner becomes quite familiar with the concepts of Sorting and Merging after having tried these programs.

20.8 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1) Write a program to create a sequential file for bank details. Assume the necessary fields.
2) Write a program to read the records of a bank file and display the customers who have amount > 10000 in their account.

3) Using Indexed Sequential Organization create an employee file. Assume necessary details.
4) Write a program to sort the marks of students in the ascending order.
5) Write a program to merge two student files based on the ascending order of marks.

20.9 POINTS FOR DISCUSSION

1) Discuss in detail about the creation and rewriting of a sequential file in COBOL.

20.10 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

UNIT – V

LESSON – 21: TABLE HANDLING

CONTENTS

21.0 AIMS AND OBJECTIVES

It becomes quite often necessary to handle a group consisting of similar items. Such a group is called a table. The aim of this lesson is to introduce the learner Occurs clause and rules for Occurs clause .

21.1 Occurs Clause

Let us introduce tables with the help of an example. Suppose there are ten different types of income-tax rates which are read from some input medium and these rates are stored in a table named INCOME-TAX-RATE. The DATA DIVISION entries for this table may be as follows:

```
01      INCOME-TAX-RATE

    02    TAX-RATE-1        PIC   99.
    02    TAX-RATE-2        PIC   99.
    02    TAX-RATE-3        PIC   99.
    02    TAX-RATE-4        PIC   99.
    02    TAX-RATE-5        PIC   99.
    02    TAX-RATE-6        PIC   99.
    02    TAX-RATE-7        PIC   99.
    02    TAX-RATE-8        PIC   99.
    02    TAX-RATE-9        PIC   99.
    02    TAX-RATE-10       PIC   99.
```

Obviously, there are ten different data names, such as TAX-RATE-1, TAX-RATE-2 etc., in the table named INCOME-TAX-RATE and each of these items is of two digits. Since the picture of all these items are identical, these can be described by having just one entry and then specifying that the description is to be repeated ten times. This is done as follows:

```
01      INCOME-TAX-RATE

    02    TAX-RATE        PIC   99    OCCURS    10    TIMES.
```

This OCCURS clause indicates that the table named INCOME-TAX-RATE is having ten elements and each one is of two digits. Now in order to refer to an individual element uniquely we must use a subscript. The first element is referred to as TAX-RATE (1), the second one as TAX-RATE (2), the seventh one as TAX-RATE (7), and so on. TAX-RATE (1), TAX-RATE (2) etc., are known as subscripted data names and 1, 2 etc., which are enclosed in parentheses are called subscripts.

The general format of OCCURS clause is as follows:

{OC} integer TIMES
{OCCURS}

21.2 RULES FOR OCCURS CLAUSE

The following rules apply for the OCCURS clause and the subscripts.

(i) The integer in the OCCURS clause must be a positive integer.

(ii) The OCCURS clause can be specified for an elementary item or for a group item. The clause causes contiguous fields to be set up internally. Each field is equivalent to the elementary or group item for which the OCCURS clause has been specified. The number of fields that are set up is equal to the integer in the OCCURS clause. The OCCURS clause cannot be specified for an item whose level number is 01, 66, 77 or 88.

(iii) When a data name is defined with the occurs clause that data name as well as any of its subordinate items cannot be referred to in the PROCEDURE DIVISION without a subscript. A subscript may be a positive integer constant, a numeric integral data item or an arithmetic expression. For example, an element of the above INCOME-TAX-RATE table, can be referred to in the PROCEDURE DIVISION as

TAX-RATE (I) or as (3*J)

In the first case, a data name I has been used as the subscript. If the current value of I is, say 5, then TAX-RATE (I) will refer to the fifth element of the table. In the second case, an arithmetic expression has been used as a subscript. The value of this expression is used to identify the particular element of the table. Thus, if the current value of J is 1, TAX-RATE (3*J) will refer to the third element of the table.

(iv) The highest value that a subscript can take is the one specified in the OCCURS clause. For any table, the lowest value of a subscript is implicitly assumed to be 1. By the range of a subscript we mean the range of values from 1 to the highest possible value of the subscript. In the above example, the range of the subscript is 1 to 10. If during the execution of a program, the value of a subscript is found to be outside its range, an execution error occurs and the program is terminated by the system.

(v) The subscripts should be enclosed in a set of parentheses. In general, blank space may not follow the left parenthesis whereas there must be a space preceding the left parenthesis and following the right one.

(vi) If a data name with the OCCURS clause requires any qualification by its higher level, the subscripts to be written after the last qualified name. For example, if TAX-RATE should be qualified, it must appear as TAX-RATE OF INCOME-TAX-RATE (I) and not as TAX-RATE (I) OF INCOME-TAX-RATE.

(vii) When an entry is defined with the OCCURS clause, the VALUE clause cannot be specified for that particular item or any item subordinate to it.

(viii) The REDEFINES clause cannot appear in the same data description entry which contains an OCCURS clause. However, the REDEFINES clause can appear for a group item whose subordinate items are defined with the occurs clause.

(ix) The OCCURS clause can appear in the data description entry in any order.

Example:

Consider the following table:

```
02          AMOUNT-TABLE  OCCURS    20    TIMES.
            03    AMOUNT      PIC   9(6) V99
            03    AMOUNT-CODE PIC   X.
```

Suppose it is required to find the total of all the amounts of the table in the following manner. If the amount code is 1, the corresponding amount is to be considered positive, otherwise the corresponding amount should be considered negative. (Note that the amount fields being unsigned contain only absolute value.) The following statements will perform the said task. It is assumed that the field named TOTAL and I are suitably defined, say with picture S9 (7) V 99 and 99 respectively.

```
        MOVE    ZERO TO    TOTAL.    MOVE1    TO    I.

PARA-LOOP
    IF    AMOUNT-CODE    (I)    IS    EQUAL    TO    "1"
          ADD  AMOUNT    (I)    TO    TOTAL
    ELSE  SUBTRACT  AMOUNT    (I)    FROM  TOTAL.
    ADD   1      TO    I
    IF    I      IS    NOT   GREATER   THAN 20
                 GO    TO    PARA-LOOP.
```

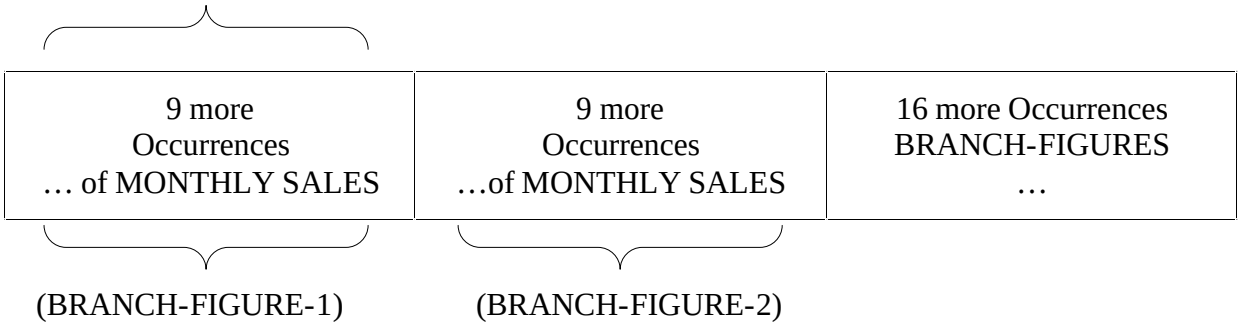
It may be noted how to use of the data name as a subscript helps to write the above code. The reader may try to find the required total without using data name or arithmetic expression as subscripts. In that case the loop cannot be designed and one must use twenty IF sentences to do the job.

21. 3 MULTI-DIMENSIONAL TABLES

The kind of table that has been considered above is called a one-dimensional table. When a table is such that each of its elements in turn is a table of one dimension, it is called a two-dimensional table. The following is an example of a two-dimensional table.

```
01      SALES-TABLE
          02    BRANCH-FIGURES OCCURS    18    TIMES.
                03    MONTHLY-SALES  PIC   9(6)   V99   OCCURS    12
          TIMES.
```

The table is assumed to store monthly sales figures for 12 months for each of the 18 br . Note that this is a two-dimensional table because each of the 18 BRANCH-FIGURES is itself a table having 12 elements. It may be further noted that a reference to an element of a two-dimensional table requires tow subscripts. We must specify the branch as well as the month so that the desired element is identified. Thus MONTHLY-SALES (3, 5) means the sales figure for fifth month of the third branch. Because of the organization specified in the above description of the table, the first subscript implicitly refers to the branch and the second subscript to the month. The two-dimensional table has been divided first into 18 one-dimensional tables through the entry at level 02. Each of these tables has then been defined by the entry at level 03. This organization can be diagrammatically shown as



If required, the tables for the individual branches can be referred to by the name BRANCH-FIGURE with only one subscript indicating the branch. Thus BRANCH-FIGURE (4) will indicate the monthly sales table for the fourth branch.

The above notion of a two-dimensional table can be easily extended to tables having three or more dimensions. Handling of tables up to three dimensions are allowed by most compilers; some even allow more than three. The following rules may be noted in connection with multi-dimensional tables.

(i) Multi-dimensional tables are to be defined as records with OCCURS clauses at various levels. As we go down the hierarchy, each lower level item with an OCCURS clause specifies an additional dimension. For example, consider the following table.

```
01      TABLE-EXAMPLE
          02    A      PIC   9(5)   OCCURS          50    TIMES
          02    B      OCCURS      20    TIMES
                03    C      PIC   9(3)
```

```
03    D OCCURS  10    TIMES.
      04 E  OCCURS    15    TIMES PIC 9(4)V99.
      04    F    PIC  X (4).
```

A and C are one-dimensional, F is a two-dimensional table and E is a three dimensional table. B and D are group items which can be referred to as one-dimensional and two dimensional tables respectively.

(ii) A table is stored in such a way that a subscript on the right of another subscript changes more rapidly than the latter.

The organization of the SALES-TABLE shown above illustrates this. The elements MONTHLY-SALES (1, 12) are stored first. The elements MONTHLY-SALES (2, 1) to MONTHLY-SALES (2, 12) are stored next, and so on. Note that the second subscript is changed more frequently than the first subscript. This fact should be taken into consideration while redefining a multi-dimensional table.

(ii) Multiple subscripts should be separated from one another either by a comma or space.

21.4 LET US SUM UP

This lesson has introduced the learner the Occurs clause, rules for OCCURS clause and multi-dimensional tables. These are necessary to write many application programs.

21.5 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Why we need OCCURS clause?
2. State the rules for OCCURS clause.
3. Give the syntax for OCCURS clause.

21.6 POINTS FOR DISCUSSION

1) Explain the usage of OCCURS clause with example.

2) Write notes on

a) Single dimensional one dimensional table.

b) Multidimensional table.

3) What do you mean by a TABLE? How to manipulate tables in COBOL? Explain.

21.7 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 22: INDEXED TABLES AND INDEX NAMES

CONTENTS

22.0 AIMS AND OBJECTIVES

The aim of this lesson is to introduce the learner Indexed Tables, Rules for Indexed Tables and SET verb. Along with their syntaxes and rules governing them, these will be discussed in this lesson.

22.1 INDEXED TABLES

The OCCURS clause which is used to define tables can optionally have an INDEXED phrase. This phrase includes the names of data items of data items that are to be used as subscripts to identify table elements. Such a data item is called an index. The following example illustrates the table description with the INDEXED phrase.

```
01    ENROLL-TABLE.
      02    FACULTY OCCURS 3 TIMES INDEXED BY F1.
            03    DEPARTMENT OCCURS 6 TIMES INDEXED BY D1.
            04    YEAR PIC 9(4) OCCURS 5 TMES.
                        INDEXED BY Y1.
```

The reference to an element of this table can be done as YEAR (F1, D1, Y1) having set appropriate values to the index names F1, D1 and Y1.

The general form of the INDEXED phrase is as follows:

INDEXED BY index-name-1 [, index-name-2]…

The OCCURS clause with the INDEXED phrase takes the following form

```
{ OC       }
{ OCCURS}  }  integer      TIMES
                    [INDEXED BY index-name-1 [, index-name-2]…]
```

22.2 RULES FOR INDEXED TABLES

The following are the rules of indexing a table with the INDEXED phrase:

(i) If indexing is done for any one level of a table, then indexing must be used for all levels. Thus it will be error if in the above the INDEXED phrase is used only for FACULTY and not for DEPARTMENT and YEAR.

(ii) Index names cannot be used in combination with subscripts. Thus a reference as YEAR (F1, S2, S3) will be treated as an error as F1 is an index name but S2 and S3 are data names. However, index names can be used in combination with numeric positive integral literals. Thus YEAR (F1, 2, 3) is valid because F1 is an index name, whereas 2 and 3 are numeric integral literals.

(iii) Indexes are valid only for the tables where they have been specified. Indexes for one table cannot be used for another table. Thus F1, D1 and Y1, being indexes for the table ENROLL-TABLE, cannot be used for other tables in the same program.

(iv) The index names must be unique. The same index name must not be used for different levels of a table.

(v) The indexes must not appear anywhere in the DATA DIVISION except in the INDEXED phrase of the OCCURS clause. This means that the index names should be implicitly defined and should not be defined explicitly.

(vi) Indexes can be manipulated only by the SET, SEARCH and PERFORM statements. The value of an index is often called the occurrence number. The internal representation of the occurrence number is system dependent.

(vii) An index can be coded plus or minus an integer literal for the relative addressing of the table elements. For example, YEAR (F1+1, D1-2, Y1-1) is valid. If F1, D1 and assume the value 1,3 and 4 respectively then this will refer to the third YEAR of the first DEPARTMENT of the second FACULTY.

(viii) There can be more than one index for each level. For example, the ENROLL-TABLE can also be defined as

```
01      ENROLL-TABLE.
          02FACULTY OCCURS 3 TIMES INDEXED BY F1, F2, F3.
              02    DEPRTMENT OCCURS 6 TIMES INDEXED BY
                    D1, D2, and D3
                    03    DEPRTMENT OCCURS 6 TIMES INDEXED
                          BY D1, D2, and D3
                          04    YEAR PIC 9(4) OCCURS 5 TIMES
                                INDEXED BY Y1, Y2, and Y3.
```

Index items defined through the INDEXED phrase of the OCCURS clause are one kind of indexes. There can be another kind of index items which are defined like data names in the DATA DIVISION with USAGE IS INDEX clause. Note that earlier we discussed only the DISPLAY and COMPUTATIONAL usages. Index is another type of usage. An index name defined with INDEX usage should not have nay picture clause in the entry.

For example, the entry

```
77    I    USAGE    IS    INDEX
```

defines the index I.

The indexes defined with the usage INDEX are called index data items. They are functionally identical to the indexes defined through the INDEXED phrase with the exception that when an index name is defined with the USAGE IS INDEX phrase, the same index name can be used for subscripts in more than one table or in more than one level of table.

22.3. SET VERB

The set verb is used to set, increase or decrease the values of the indexes. For example, the statement

SET F1 TO 4

will set the value of the index F1 to 4.
There are several forms of the SET verb:

(I) To set one particular value to tone one or more index names we can use the following form.

SET index-name-1 [, index-name-2]… TO {identifier-1 integer-1}

For example, SET F1, Y1 TO 3.
Only positive integral values can be set to an index.

(II) To move the current value of an index to one or more identifiers, the following form of the SET verb can be used.

SET identifier-2 [, identifier-3]… TO index-name-3

If A and B are data names and F1 is an index name, the statement

SET A, B TO F1

Indicates that the current value of F1 will be stored in both the data names A and B.

(III) When it becomes necessary to increment or decrement one or more indexes by a positive integer value, the following form may be applied.

SET index-name-4 [, index-name-5] … {UP BY}{DOWN BY} {identifier-4 integer-2}

The phrase UP BY is used to increment the values of the indexes and the phrase DOWN BY is used to decrement their values.

Thus, to increment the current value of F1 and Y1 by 2, the following statement may be used.

SET F1, Y1 UP BY 2.

On the other hand, the statement

SET D1 DOWN BY A

Indicates that the current value of the index D1 will be decremented by the current value of the data name A. If before the execution of the above statement, A and D1 contain 3 and 7 respectively, then after the execution of this statement, D1 will contain 4.

22.4 LET US SUM UP

This lesson has taught the learner in detail about Indexed Tables, Rules for Indexed Tables and SET verb. The learner can now be in a position to employ them wherever any application program finds suitability of these discussed verbs.

22.5 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. What do you mean by Indexed Table?
2. State the rules for Indexed Tables.
3. What is the use of SET Verb?
4. Specify different forms of SET verb.

22.6 POINTS FOR DISCUSSION

1) Differentiate table from indexed Table.

22.7 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 23: SEARCH AND START VERBS

CONTENTS

23.0 AIMS AND OBJECTIVES

The learner will be introduced Search Verb, rules for SEARCH verb and START Statement in this lesson. With the syntaxes and examples of these verbs, the concepts will be discussed.

23.1 SEARCH VERB

The SEARCH verb is used to locate elements in one-dimensional tables. Let us consider the following problem. Suppose each element of a table consists of three fields, namely, the account number of a person, the name of that and the amount that he has deposited. There are four hundred such elements in the table and we want to find out whether a particular name is present in the table or not. The desired name is given in the field is called NAME and if this name appears in an element of the table, we would like to display the name as well as the corresponding account number and amount. The DATA DIVISION entries for this problem are as follows:

```
DATA DIVISION
     .
     .
     .
     .
     77    NAME              PIC     X (20).
     01    SAVINGS-BANK-ACCOUNT.
           02    ACCOUNT-TABLE OCCURS 400 TIMES
                       INDEXED BY A1.
                 03    ACCOUNT-NUMBER        PIC     9(6).
                 03    NAME-OF-THE-PERSON    PIC     X (20).
                 03    AMOUNT                PIC     9(6).99.
```

The following PROCEDURE DIVISION statements can be a solution to the above problem.

```
PROCEDURE DIVISION
        .
        .
        .
SET     A1 TO 1.
SEARCH ACCOUNT-TABLE
        AT      END DISPLAY “NAME NOT FOUND”
        NAME=NAME-OF-THE-PERSON (A1)
        DISPLAY ACCOUNT-NUMBER (A1), NAME,
                AMOUNT (A1).
```

In the above SEARCH statement, there are two parts- the AT END part and the WHEN part. If the condition NAME= NAME-OF-THE-PERSON (A1) is satisfied for some value of the index name A1, the statement DISPLAY ACCOUNT-NUMBER (A1), NAME, AMOUNT (A1) is executed. The AT END part is executed only when the entire table is searched and the condition is not satisfied for any value of A1. The increment of A1 is taken care of by the SEARCH verb.

To illustrate another use of the SEARCH verb, suppose we wish to search the same table to find the number of persons whose deposited amount is greater than 5000.00. For this we describe another data NO-OF-PERSONS in the DATA DIVISION with the picture say 999. The following statements in the PROCEDURE DIVISION will perform the desired search.

```
                MOVE ZEROS TO NO-OF-PERSONS
                SET A1 TO 1.

PARA-REPEAT
                SEARCH ACCOUNT-TABLE
                        AT END GO TO PARA-NEXT
                WHEN AMOUNT (A1) IS-GREATER THAN 5000.00
                        ADD 1 TO NO-OF-PERSONS
                        SET A1 UP BY 1
                        GO TO PARA-REPAEAT
PARA NEXT
        .
        .
        .
        .
SEARCH      identifier-1    [VARYING     identifier-2 index-name-1}]
        [; AT END     imperative-statement-1]
;       WHEN        Condition-1    {imperative-statement-2       NEXT   SENTENECE}
[; when     Condition-2    {imperative-statement-2       NEXT          SENTENCE}]…
```

23.2 RULES FOR SEARCH VERB

The following rules apply for the SEARCH verb.

(I) The SEARCH verb can only be applied to a table which has the OCCURS clause and INDEXED phrase. The identifier-1 indicates the table to be searched and it must not be indexed or subscripted.

(II) Before the use of the SEARCH verb, the index must have some initial value. The initial value must not exceed the size of the table. If it exceeds, the search is

terminated immediately. Then if the AT END clause is specified, statements after AT END will be executed; otherwise the control passes to the next sentence.

(III) If the AT END condition is specified, as in the case of the first example, and if the element which is being searched is not found in the table, the statement after the AT END clause will be executed if statements after AT END do not transfer the control elsewhere in the program. On the other hand, if AT END is not used and the end of the table is reached, the control will be automatically transferred to the next sentence.

(IV) The SEARCH verb starts with the initial value of the index and tests whether the conditions stated in the WHEN clauses have been satisfied or not. If none of the conditions are satisfied the index is incremented automatically by 1. The process is continued until the index value exceeds the size of the table, the statements following the condition in the relevant WHEN clauses are executed. If these statements do not transfer the control elsewhere, after their execution, it is transferred to the next sentence. The value of the index remains set at the point where the condition has been satisfied.

(V) Connected with the VARYING option, identifier-2 can be either a data, an integral elementary item or an index data item (described with USAGE AS INDEX CLAUSE). The purpose of specifying the VARYING clause is that identifier-2 is also incremented each time the index of the table is incremented.

23.3 START STATEMENT

The format of the START statement is given below.

START file-name [KEY IS {EQUAL TO
=
GREATER THAN
>
NOT LESS THAN
NOT < THAN} data-name]
[; INVALID KEY imperative statement]

The START statement enables the programmer to position the relative file at some specified point so that subsequent sequential operations on the file can start form this point instead of the beginning. The KEY IS phrase indicates how the file is to be positioned. The data name in this phrase must be the data name in the RELATIVE KEY phrase of the SELECT clause. When the EQULA TO OF NOT LESS THAN condition is specified, the file is GREATER THAN condition is specified, the file is positioned at the next relative position indicated by the relative key data item. Thus

```
START MY-FILE KEY IS GREATER THAN REL-KEY
      ;       INVALID KEY GO TO INVALID PARA.
```

will position the file at the fifty-first record position if the relative key data item REL-KEY contains 50.

The INVALID KEY condition arises if the specified record position if the specified record position is empty. In that case the imperative statement after the INVALID KEY phrase is executed.

The START statement requires that the file must be opened in the INPUT or I-O mode. The access mode can only be SEQUENTIAL or DYNAMIC

23.4 LET US SUM UP

In this lesson the learner has been introduced the Search Verb, Rules for SEARCH verb and START Statement. With this knowledge the learner can make use of these concepts in application programs.

23.5 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. What is the role of SEARCH verb?
2. Give the syntax of SEARCH verb.
3. State the rules for SEARCH verb.
4. Explain with syntax START statement

23.6 POINTS FOR DISCUSSION

1) Explain the process of locating the elements in one-dimensional table with example.

23.7 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 24: PROGRAMS USING OCCURS & SCREEN SECTION

CONTENTS

24.0 AIMS AND OBJECTIVES

In this lesson the learner will be introduced the programs for occurs clause, occurs indexed by clause and Screen section. These programs also make use of the SET and SEARCH verbs. The learner is expected to type these programs and try them on the system.

24.1 OCCURS CLAUSE – PROGRAM-1

Write a program to demonstrate occurs clause. Get names of the students and display them on the screen.

```
identification division.
program-id.
environment division.
data division.
working-storage section.
01  n  pic  9(2) value 0.
01  i  pic  9(2) value 0.
01  name-in.
  02 name pic x(20) occurs 20 times.
01  key-in pic x value space.

procedure division.
p-1.
        display(1 1) erase.
        display(3 5) "Enter How many times".
        accept n.
        display(1 1) erase.
        perform get-para n times.
        display(1 1) erase.
        move 0 to i.
        perform disp-para n times.
```

```
            stop run.
get-para.
            accept name(i).
            add 1 to i.
disp-para.
            display name(i).
            accept key-in.
            add 1 to i.
```

24.2 OCCURS CLAUSE – PROGRAM-2

Write a program to demonstrate occurs clause. Get names and marks of the students and display them on the screen.

```
identification division.
program-id.
environment division.
data division.
working-storage section.
01  n  pic  9(2) value 0.
01  i  pic  9(2) value 1.
01  stu-det.
  02 stu-rec occurs 10 times.
            03 name pic x(20).
            03 mark pic 9(3).
01  key-in pic x value space.

procedure division.
p-1.
            display(1 1) erase.
            display(3 5) "Enter How many times".
            accept n.
            display(1 1) erase.
            perform get-para n times.
            display(1 1) erase.
            move 1 to i.
            perform disp-para n times.
            stop run.
get-para.
            accept name(i).
            accept mark(i).
            add 1 to i.
disp-para.
            display name (i).
            display mark (i).
            accept key-in.
            add 1 to i.
```

24.3 OCCURS – INDEXED BY PROGRAM

```
identification division.
program-id. searching.
environment division.
```

```
data division.
working-storage section.
01 table1.
   02 data1 occurs 10  times indexed by a1.
        03  name pic x(25).
        03  sal  pic 9(6).
01   n   pic 9 value 0.
01   tot pic 9(2) value 0.
01   i   pic 9 value 0.
procedure division.
p-1.
        display(1 1) erase.
        display(5 5) "Enter Data .....".
        display(7 5) "Enter No.of Records : ".
        accept n.
        perform init-para varying i from 1 by 1 until i > n.
        perform get-para varying i from 1 by 1
                        until i > n.
        set a1 to 1.
p-2.
      search data1 at end go to p-3
        when  sal (a1) > 1000
                add 1 to tot.
      set a1 up by 1
      go to p-2.
  p-3.
        display(20 5) "Total Records > 1000 [sal] = " tot.
        stop run.

  get-para.
        display(1 1) erase.
        display(3 5) "Name : ".
        accept name (i).
        display(5 5) "Salary : ".
        accept sal (i).

  init-para.
        move spaces to name(i).
        move 0 to sal(i).
```

24.4 PROGRAM FOR SCREEN SECTION

```
identification division.
program-id.
environment division.
data division.

working-storage section.
01  a pic 9(2) value 0.
01  b pic 9(2) value 0.
01  c  pic 9(3) value 123.
```

```
screen section.
01   b-screen.
          02  blank screen.

01   screen1.
   02   line 3 column 5  pic 9(2) to a auto bell.
   02   line 5 column 5 pic 9(2) to b auto bell.

01   screen2.
   02   line 7 column 5  pic 9(2) from a blink reverse-video.
   02   line 9 column 5 pic 9(2) from b highlight blink.

procedure division.
p0.
          display b-screen.
p1.
          display screen1.
          accept screen1.
          display screen2.
          compute c = a + b.
          display " ".
          display " c = " c.
          display "Using exhibit".
          exhibit c.

          stop run.
```

24.5 PROGRAM FOR SCREEN SECTION WITH FILES

```
identification division.
program-id.
environment division.
input-output section.
file-control.
          select stu-file assign to disk
          file status is fs.
data division.
file section.
fd stu-file
          label records are standard
          value of file-id is 'stu.dat'.
01 stu-rec.
          02   name  pic x(20).
          02   mark  pic 9(3).
working-storage section.
01   ans  pic  x value space.
01   eof  pic  x value space.
01   fs   pic  x(2) value spaces.
screen section.
01 get-screen.
          02   line 3 column 5 value "Name : ".
```

```
            02   line 3 column 15   pic x(5)
                 to  name auto bell reverse-video.
            02   line 5 column 5 value "Mark : ".
            02   line 5 column 15  pic 9(3) to mark bell blink.
            02   line 7 column 5 value "Continue [y/n] : ".
            02   column  plus 3  pic x to ans  bell blink.
   01 b-screen.
            02  blank screen.
   01 put-screen.
            02   line 3 column 25 value "Name : " highlight.
            02   column plus 2 pic x(20) from name blink.
            02   line 5 column 25 value "Mark : "  blink.
            02   column plus 2 pic 9(3) from mark underline.
procedure division.
   p-1.
       display b-screen.
       open output stu-file.
       display " File Staus Value  ... Exhibit ... display..".
       exhibit fs.
       display fs.
       display " Press a Key!".
       accept ans.
       perform g-w-para until ans = 'n'.
       close stu-file.

       move space to ans.
       open input stu-file.
       read stu-file at end move 'y' to eof.
       perform disp-para until eof  = 'y'.
       close stu-file.
       stop run.

g-w-para.
       display b-screen.
       display get-screen.
       accept get-screen.
       write stu-rec.
disp-para.
       display (1 1) erase.
       display put-screen.
       display (15 5) "Press any Key ! ".
       accept ans.
       read stu-file at end move 'y' to eof.
```

24.6 LET US SUM UP

The above lesson has introduced the learner the programs related with Occurs clause, Occurs – Indexed, Screen Section and Screen Section with files. These programs will give confidence to the learner to make use of them in industry applications wherever need arises.

24.7 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Write a program to read the names and marks of 5 students. Display them using OCCURS clause.
2. Write a program to display a welcome message using Screen Section features.
3. Write a program using Screen Section the creation of a College file. Assume necessary details.

24.8 POINTS FOR DISCUSSION

1) Discuss the usage of screen section COBOL.

24.9 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.

LESSON – 25: LIST OF PROGRAMS

CONTENTS

25.0 AIMS AND OBJECTIVES

The main aim of this lesson is to introduce the learner the various programs that make use of the concepts he/she has learnt so far. There are 11 programs in this lesson. The learner is expected to try all these programs in system.

25.1 PROGRAM FOR ADD CORRESPONDING

Write a simple program to demonstrate Add Corresponding.

```
identification division.
program-id.
environment division.
data division.
working-storage section.
01  rec-1.
        02   sno   pic  9(2) value 11.
        02   mark1 pic 9(2) value 1.
01  rec-2.
        05   sno   pic  9(2)  value 10.
        05   mark2 pic 9(2) value 2.
procedure division.
p-1.
        display(1 1) erase.
        display(3 5) mark1 of rec-1.
        add corr rec-1 to rec-2.
        display(5 5) mark2 of rec-2.
```

```
        display (10 5) sno of rec-2.
        stop run.
```

25.2 PROGRAM FOR MOVE CORRESPONDING

Write a simple program to demonstrate Move Corresponding.

```
identification division.
program-id.
environment division.
data division.
working-storage section.
01  rec-1.
        02   sno   pic  9(2) value 11.
        02   name  pic  x(4) value "Ravi".

01  rec-2.
        05   sno   pic  z(2).
        05   f     pic  x(10) value spaces.
        05   name   pic  x(4).
        05   f     pic  x(10) value spaces.

procedure division.
p-1.
        display(1 1) erase.
        move corr rec-1 to rec-2.
        display rec-2.
        stop run.
```

25.3 PROGRAM FOR CONDITION NAMES

Write a simple program to demonstrate Condition names usage.

```
identification division.
program-id.
environment division.
data division.
working-storage section.
01   ms     pic  9(2).
88   s    value 0 thru  9 .
88   m    value 10 thru 99.
procedure division.
p-1.
        display(1 1) erase.
        display(5 5) "Enter Marriage Status : ".
        display(7 5) " 0 to 9   .... Single Person ".
        display(8 5) " 10 to 99 .... Married Person".
        accept ms.
        if s   display(10 5) "Single".
        if m   display(10 5) "Married".
        stop run.
```

25.4 PROGRAM USING OCCURS CLAUSE

Write a simple program to demonstrate Occurs Clause. Get the employee details for 3 persons and calculate the total salary.

```
identification division.
program-id. emp-occurs.
environment division.
data division.
working-storage section.
01  emp-rec.
        02 emp-det occurs 3 times.
                05 emp-name  pic x(20).
                05 emp-sal   pic 9(5)v9(2).

01 head-1   pic x(80) value all "-".
01 head-2.
        02  f  pic x(15) Value spaces.
        02  f  pic x(20) value "NAME".
        02  f  pic x(15) Value spaces.
        02  f  pic x(10) value "SALARY".
        02  f pic x(20) value spaces.

01 head-3.
        02  f pic x(15) value spaces.
        02  e-name pic x(20).
        02  f  pic x(15) Value spaces.
        02  e-sal   pic z(5).z(2).
        02  f  pic x(22) Value spaces.
01   i  pic 9 value 1.
01   tot-sal pic 9(6)v9(2) value 0.
01   e-tot-sal pic z(6).z(2).

procedure division.
para-1.
        display(1 1) erase.
        display(3 5) "Enter Data for 3 Employees...".
        perform get-para 3 times.

        display head-1.
        display head-2.
        display head-1.

        move 1 to i.
        perform disp-para until i > 3.
        display head-1.

        move tot-sal to e-tot-sal.
        display "Total Salary = " e-tot-sal.
        display " ".
        stop run.
```

```
get-para.
        display(1 1) erase.
        display(5 5) "Name :  ".
        accept emp-name(i).
        display(10 5) "Salary : ".
        accept emp-sal(i).
        add 1 to i.

disp-para.
        move  emp-name(i) to e-name.
        move  emp-sal(i)  to e-sal.
        compute tot-sal = tot-sal + emp-sal(i).
        display head-3.
        add 1 to i.
```

25.5 PROGRAM FOR MASTER FILE MAINTENANCE

Write a program to maintain the stu-file for which a record has just 2 fields, namely, rno(Roll Number) and name(Student Name). Give the provisions to add, modify, delete the records in the file

```
identification division.
program-id.
environment division.
input-output section.
file-control.
        select stu-file assign to disk
        organization is indexed
        access mode is random
        record key is rno
        file status is fs.

data division.
file section.

fd stu-file
        label records are standard
        value of file-id is "stu.dat".
01 stu-rec.
        02 rno pic 9(3).
        02 name pic x(20).

working-storage section.
01 fs pic x(2) value spaces.
01 ans pic x value space.
01 ch pic 9 value 0.

procedure division.
open-para.
        open i-o stu-file.
        if fs = "30"
```

```
                open output stu-file
                close stu-file
                open i-o stu-file.
p-1.
        display(1 1) erase.
        display(3 5) "Main Menu".
        display(5 5) "1 to Add".
        display(7 5) "2 to Modify".
        display(9 5) "3 to Delete".
        display(11 5) "4 to Exit".
        display(13 5) "Your Choice [1..4] : ".
        accept ch.
        if ch < 0 or ch > 4 go to p-1.
        go to add-para modi-para del-para exit-para
                depending on ch.
add-para.
        display(1 1) erase.
        display(3 5) "Rno : ".
        accept rno.
        display(5 5) "Name : ".
        accept name.
        write stu-rec invalid key display(15 5) "Error!".
        display(20 5) "Continue Add Records [y/n] : ".
        accept ans.
        if ans = "y" go to add-para else go to p-1.
modi-para.
        display(1 1) erase.
        display(3 5) "Enter Roll no to Modify".
        accept rno.
        read stu-file key is rno
                invalid key display(13 5) "No Record Found"
                        go to c-para.
        display(5 5) "Rno = " rno.
        display(7 5) "Name = " name.
        display(9 5) "Sure to Modify [y/n] : ".
        accept ans.
        if ans  = 'y'
                display(1 1) erase
                display(5 5) "Name : "
                accept name
              rewrite stu-rec.
c-para.
        display(15 5) "Continue Modification [y/n] : ".
        accept ans.
        if ans = 'y' go to modi-para else go to p-1.
del-para.
      display(1 1) erase.
        display(3 5) "Enter Roll no to Delete".
        accept rno.
        read stu-file key is rno
```

```
                invalid key display(13 5) "No Record Found"
                        go to c-para1.
        display(5 5) "Rno = " rno.
        display(7 5) "Name = " name.
        display(9 5) "Sure to Delete [y/n] : ".
        accept ans.
        if ans  = 'y'
                delete stu-file record.
c-para1.
        display(15 5) "Continue  Deletion  [y/n] : ".
        accept ans.
        if ans = 'y' go to del-para else go to p-1.

exit-para.
        close stu-file.
        stop run.
```

25.6 PROGRAM FOR MASTER FILE MAINTENANCE & ALTERNATE KEY USAGE

Write a program to maintain the stu-file for which a record has just 2 fields, namely, rno(Roll Number) and name(Student Name). Give the provisions to add, modify, delete the records in the file. Make use of the alternate key option for your program.

```
identification division.
program-id.
environment division.
input-output section.
file-control.
        select stu-file assign to disk
        organization is indexed
        access mode is random
        record key is rno
        alternate record key is name with duplicates
        file status is fs.
data division.
file section.

fd stu-file
        label records are standard
        value of file-id is "stu.dat".
01 stu-rec.
        02 rno pic 9(3).
        02 name pic x(20).

working-storage section.
01 fs pic x(2) value spaces.
01 ans pic x value space.
01 ch pic 9 value 0.
```

```
procedure division.
open-para.
        open i-o stu-file.
        if fs = "30"
                open output stu-file
                close stu-file
                open i-o stu-file.
p-1.
        display(1 1) erase.
        display(3 5) "Main Menu".
        display(5 5) "1 to Add".
        display(7 5) "2 to Modify".
        display(9 5) "3 to Delete".
        display(11 5) "4 to Exit".
        display(13 5) "Your Choice [1..4] : ".
        accept ch.
        if ch < 0 or ch > 4 go to p-1.
        go to add-para modi-para del-para exit-para
                depending on ch.
add-para.
        display(1 1) erase.
        display(3 5) "Rno : ".
        accept rno.
        display(5 5) "Name : ".
        accept name.
        write stu-rec invalid key display(15 5) "Error!".
        display(20 5) "Continue Add Records [y/n] : ".
        accept ans.
        if ans = "y" go to add-para else go to p-1.
modi-para.
        display(1 1) erase.
        display(3 5) " Press 1 if you know Roll No , 2 if you
        -    "    know Name ".
        accept ch.
        if ch = 1
          display(5 5) "Enter Roll no to Modify"
        accept rno
        read stu-file key is rno
                invalid key display(13 5) "No Record Found"
                        go to c-para
        else if ch = 2
         display(5 5) "Enter Name to Modify"
        accept name
        read stu-file key is name
                invalid key display(13 5) "No Record Found"
                        go to c-para.
        display(1 1) erase.
        display(5 5) "Rno = " rno.
        display(7 5) "Name = " name.
        display(9 5) "Sure to Modify [y/n] : ".
```

```
            accept ans.
            if ans  = 'y'
                    display(1 1) erase
                    display(5 5) "Name : "
                    accept name
                  rewrite stu-rec.
      c-para.
            display(15 5) "Continue Modification [y/n] : ".
            accept ans.
            if ans = 'y' go to modi-para else go to p-1.
      del-para.
            display(1 1) erase.
            display(3 5) " Press 1 if you know Roll No , 2 if you
-     "    know Name ".
            accept ch.
            if ch = 1
              display(5 5) "Enter Roll no to Delete"
            accept rno
            read stu-file key is rno
                    invalid key display(13 5) "No Record Found"
                            go to c-para1
            else if ch = 2
             display(5 5) "Enter Name to Delete"
            accept name
            read stu-file key is name
                    invalid key display(13 5) "No Record Found"
                            go to c-para1.
            display(1 1) erase.
            display(5 5) "Rno = " rno.
            display(7 5) "Name = " name.
            display(9 5) "Sure to Delete [y/n] : ".
            accept ans.
            if ans  = 'y'
                    delete stu-file record.
      c-para1.
            display(15 5) "Continue  Deletion  [y/n] : ".
            accept ans.
            if ans = 'y' go to del-para else go to p-1.

      exit-para.
            close stu-file.
            stop run.
```

25.7 PROGRAM TO FIND INTEREST ON BANK DEPOSITS

Write a program to find interest on deposits
The criteria is given below :

Principal	Years of Deposit	IntRate
>=5000	>=3	10%
>=5000	<3	8%
<5000	Any	7%

```
identification division.
program-id.
environment division.
input-output section.
file-control.
        select bankfile assign to disk
        organization is line sequential.
data division.
file section.
fd bankfile
        label records are standard
        value of file-id is "bank.dat".
01 bankrec.
        02 dno   pic 9(5).
        02 dname pic x(21).
        02 p     pic 9(4)v9(2).
        02 n     pic 9(2).

working-storage section.
01   ans pic x value space.
01   tot pic 9(6)v9(2) value 0.
01   int pic 9(6)v9(2) value 0.
01   key-in pic x value space.
01 r pic 9(2) value 0.
01 head-1 pic x(80) value all '-'.
01 head-2.

        02  f pic x(8) value "Deps No".
        02  f pic x(3) value spaces.
        02  f pic x(10) value "Deps Name".
        02  f pic x(5) value spaces.
        02  f pic x(8) value "Deposit".
        02  f pic x(5) value spaces.
        02  f pic  x(8) value "Period".
        02  f pic x(5) value spaces.
        02  f pic x(5) value "Rate".
        02  f pic x(5) value spaces.
        02  f pic x(8) value "Interest".
        02  f pic x(10) value "  Nett".
01  head-3.
        02  e-dno  pic z(5).
        02  e-dname pic x(26).
        02  e-p  pic  z(4).z(2).
```

```
        02  f pic x(5) value spaces.
        02  e-n  pic z(2).
        02  f pic x(5) value spaces.
        02  e-r pic z(2).
        02  f pic x(3) value spaces.
        02  e-int  pic z(4).z(2).
        02  f pic x(3) value spaces.
        02  e-tot pic z(6).z(2).

procedure division.
p-1.
        open output bankfile.
        perform g-w-para until ans = 'N' or 'n'.
        close bankfile.

        open input bankfile.
        display(1 1) erase.
        display head-1.
        display head-2.
        display head-1.
read-para.
        read bankfile at end go to close-para.
        if ( p not < 5000 and n not < 3 )
                move 10 to r
                compute tot = p * (1 +  r / 100) ** n.

        if ( p not < 5000 and n < 3)
                move 8 to r
                compute tot = p * (1 +  r / 100) ** n.

        if ( p < 5000)
                move 7 to r
                compute tot = p * (1 + r / 100) ** n.

        move tot to e-tot.
        compute int = tot - p.
        move int to e-int.
        move dno to e-dno.
        move dname to e-dname.
        move p to e-p.
        move n to e-n.
        move r to e-r.
        display head-3.
        go to read-para.
close-para.
        display head-1.
        close bankfile.
        stop run.

g-w-para.
```

```
        display(1 1) erase.
        display(3 5) "Dep No : ".
        accept dno.
        display(5 5) "Dep Name : ".
        accept dname.
        display(7 5) "Amount : ".
        accept p.
        display(9 5) "Years : ".
        accept n.
        write bankrec.
        display(15 5) "Add more [y/n] : ".
        accept ans.
```

25.8 PROGRAM FOR INVENTORY

Write a program to update the inventory file. Consider 2 files,namely, invfile and tranfile. Invfile has 3 fields pno,name,qty. Tranfile has 3 fileds tpno,trcode,tqty. By reading tranfile records if trcode=1 then update the qty with qty+tqty in invfile. If trcode=2 then update the qty with qty-tqty in invfile.

```
identification division.
program-id.
environment division.
input-output section.
file-control.
        select invfile assign to disk
        organization is indexed
        access mode is dynamic
        record key is pno
        file status is fs.
        select tranfile assign to disk
        organization is line sequential.
data division.
file section.
fd  invfile
        label records are standard
        value of file-id is "inv.dat".
01  invrec.
        02 pno  pic 9(5).
        02 name pic x(5).
        02 qty  pic 9(5).

fd tranfile
        label records are standard
        value of file-id is "tran.dat".
01 tranrec.
        02 tpno          pic 9(5).
        02 trcode pic 9.
        02 tqty    pic 9(5).

working-storage section.
01  ans pic x value space.
```

```
01  fs  pic x(2) value spaces.
01  key-in pic x value space.

procedure division.
p-1.
        open i-o invfile.
        if fs = "30"
                open output invfile
                close invfile
                open i-o invfile.
        perform g-w-inv until ans = 'n' or 'N'.
        move space to ans.
        open output tranfile.
        perform g-w-tran until ans = 'n' or 'N'.
        close tranfile.
        open input tranfile.
read-para.
        read tranfile at end go to close-para.
        move tpno to pno.
        read invfile key is pno invalid key
          display(5 5) "No Record Found for" ; pno
          accept key-in
          go to read-para.
        if trcode = 1
                add tqty to qty
                rewrite invrec.
        if trcode = 2
                subtract tqty from qty
                rewrite invrec.
        go to read-para.
close-para.
        close invfile tranfile.
op-para.
        open input invfile.
r-para.
        read invfile next record at end go to cl-para.
        display(1 1) erase.
        display(3 5) "Part No: " pno.
        display(5 5) "Name : " name.
        display(7 5) "Qty : " qty.
        accept key-in.
        go to r-para.
cl-para.
        close invfile.
        stop run.
g-w-tran.
        display(1 1) erase.
        display(2 5) "Tran Details ....".
        display(3 5) "Part No: ".
        accept tpno.
```

```
        display(5 5) "Trancode : ".
        accept trcode.
        display(7 5) "Quantity : ".
        accept tqty.
        write tranrec.
        display(12 5) "Add more [y/n] :".
        accept ans.
g-w-inv.
        display(1 1) erase.
        display(2 5) "Inven Details....".
        display(3 5) "Part No: ".
        accept pno.
        display(5 5) "Name : ".
        accept name.
        display(7 5) "Quantity : ".
        accept qty.
        write invrec invalid key display(10 5) "Rec Exists!".
        display(12 5) "Add more [y/n] :".
        accept ans.
```

25.9 PROGRAM TO REWRITE PAY RATE

Write a program to modify the hourly-pay-rate based on the criteria given below:

Existing Hpr in Rs)	Increase in Hpr
<=5	25%
>5 but <=8	20%
>8 but <=12	15%
>12	10%

```
identification division.
program-id.
environment division.
input-output section.
file-control.
        select payfile assign to disk.
data division.
file section.
fd  payfile
        label records are standard
        value of file-id is "pay.dat".
01  payrec.
        02 idno          pic x(3)9(6).
        02 name          pic  x(25).
        02 hpr           pic 99v99.
        02 mis           pic x(42).
working-storage section.
01   ans pic x value space.
01 head-1 pic x(80) value all '-'.
```

```
01 head-2.
        02  f pic x(5) value spaces.
        02  f pic x(5) value "IDNO".
        02  f pic x(5) value spaces.
        02  f pic x(5) value "NAME".
        02  f pic x(5) value spaces.
        02  f pic x(7) value "NEWHPR".
        02  f pic x(36) value "MIS INFO".
        02  f  pic x(12) value spaces.
01 head-3.
        02  e-idno pic x(3)9(6).
        02  e-name pic x(25).
        02  e-hpr  pic z(2).z(2).
        02  e-mis pic x(41).
procedure division.
p-1.
        open extend payfile.
        perform g-w-para until ans = 'N' or 'n'.
        close payfile.
        open i-o payfile.
read-para.
        read payfile  at end go to close-para.
        display(1 1) erase.
        display(3 5) "HPR = " hpr.
        accept ans.
        if ( hpr not greater 5 )
          compute hpr = hpr + hpr * 0.25.
        if ( hpr > 5 and hpr not greater 8 )
          compute hpr = hpr + hpr * 0.20.
        if ( hpr > 8 and hpr not greater 12  )
          compute hpr = hpr + hpr * 0.15.
        if ( hpr > 12 )
          compute hpr = hpr + hpr * 0.10.
        rewrite payrec.
        move 0 to hpr.
        go to read-para.
close-para.
        display head-1.
        close payfile.
disp-para.
        open input payfile.
        display head-1.
        display head-2.
        display head-1.
read-para-1.
        read payfile  at end go to close-para-1.
        move idno to e-idno.
        move name to e-name.
        move hpr to e-hpr.
        move mis to e-mis.
```

```
        display head-3.
        go to read-para-1.
close-para-1.
        display head-1.
        close payfile.
        stop run.
g-w-para.
        display(1 1) erase.
        display(3 5) "ID no [ x(3) 9(6)] : ".
        accept idno.
        display(5 5) "Emp Name :".
        accept name.
        display(7 5) " Hrly Pay Rate : ".
        accept hpr.
        display(9 5) "Miscell Info : ".
        accept mis.
        write payrec.
        display(15 5) "Add more (y/n) :".
        accept ans.
```

25.10 PROGRAM FOR ELECTRICITY BILL

An electricity company supplies electricity to 4 types of customers coded 1,2,3 & 4.The rate schedule for customers is shown in one table as

Customer code	1	2	2	3	3	4
Consumption	All	Below 1000	1000& above	Below 5000	5000& above	All
Rate/unit	0.50	0.40	0.50	0.30	0.50	0.30

The customer record is:

col:1-5 customer no.
col:6-60 name & address.
col:61-67 consumption in units.
col:68 customer code.

Read a customer record and print a bill using GOTO DEPENDING ON statement.

```
identification division.
program-id.
environment division.
input-output section.
file-control.
        select efile assign to disk
        organization is line sequential.
data division.
file section.
fd  efile
        label records are standard
        value of file-id is "e.dat".
```

```
01  erec.
        02  ccode        pic 9.
        02  units       pic 9(7).

working-storage section.
01   ans pic x value space.
01   head-1 pic x(80) value all '-'.
01   amt  pic 9(5)v9(2) value 0.

01   head-2.
        02  f pic x(11) value spaces.
        02  f pic x(6) value "CSCODE".
        02  f pic x(11) value spaces.
        02  f pic x(5) value "Units".
        02  f pic x(11) value spaces.
        02  f pic x(10) value "Amount".
        02  f pic x(11) value spaces.

01 head-3.
        02  f pic x(5) value spaces.
        02  e-ccode  pic z.
        02  f pic x(5) value spaces.
        02  e-units pic z(7).
        02  f pic x(10) value spaces.
        02  e-amt  pic z(5).z(2).

screen section.
01  b-screen.
        02 blank screen.

procedure division.
p-1.
        open extend efile.
        perform g-w-para until ans = 'n' or 'N'.
        close efile.

        open input efile.
        display head-1.
        display head-2.
        display head-1.
read-para.
        read efile at end go to close-para.

        go to c1  c2  c3 c4 depending on ccode.

c1.
        compute amt = units * 0.50
        go to m-para.
c2.
        if ( units < 1000)
```

```
                compute amt = units * 0.40
                go to m-para.
          if (units not < 1000 )
        compute amt = (units - 999) * 0.5 + 999 * 0.40
                go to m-para.
   c3.
          if (units < 5000)
          compute amt =  units * 0.30
                go to m-para
          else
         compute amt = (units - 4999) * 0.5 + 4999 * 0.30
                go to m-para.
   c4.
          compute amt = units * 0.30.
m-para.

          move amt to e-amt.
          move ccode to e-ccode.
          move units to e-units.
          display head-3.
          go to read-para.

   close-para.
          display head-1.
          close efile.
          stop run.

   g-w-para.
          display b-screen.
          display(3 5) "Cus Code: ".
          accept ccode.
          display (5 5) "Units : ".
          accept units.
          write erec.
          display(10 5) "Continue [y/n] : ".
          accept ans.
```

25.11 PROGRAM TO CREATE 2 FILES USING AN EXISTING FILE

A file contains the following records about a class.

Fields	Columns
Serial No.	1-4
Roll No.	5-10
Name	11-30
Age	31-32
Sex	33
Year in college	34

Select records with the following characteristics and write

them in 2 files.
File-1: Records of all males over 18 years of age who are in the third year in the college.
File-2: Records of all females under 19 years of age in the fourth year in the college.
Use condition names for sex and year in college.

```
identification division.
program-id.
environment division.
input-output section.
file-control.
        select collfile assign to disk
        organization is line sequential.

        select malefile assign to disk
        organization is line sequential.

        select femalefile assign to disk
        organization is line sequential.

data division.
file section.
fd collfile

        label records are standard
        value of file-id is "coll.dat".

01 collrec.
        02  sno   pic 9(4).
        02  rno   pic 9(6).
        02  name  pic x(20).
        02  age pic 9(2).
        02  sex pic 9.
                88 male value 1.
                88 female value is   2.
        02  yr  pic 9.
                88 third value 3.
                88 fourth value is 4.

fd malefile
        label records are standard
        value of file-id is "male.dat".
01 malerec pic x(80).

fd femalefile
        label records are standard
        value of file-id is "female.dat".
01 femalerec pic x(80).
```

```
working-storage section.
01   ans pic x value space.

procedure division.
p-1.
        display(1 1) erase.
        open output collfile.
        perform g-w-para until ans = 'N' or 'n'.
        close collfile.

        open input collfile output malefile femalefile.
read-para.
        read collfile at end go to close-para.
        if (male and age > 18 and third)
          move collrec to malerec
          write malerec.
        if(female and age < 19 and fourth)
          move collrec to femalerec
          write femalerec.
        go to read-para.

close-para.
        close collfile malefile femalefile.
        stop run.

g-w-para.
        display(1 1) erase.
        display(3 5) "Sno : ".
        accept sno.
        display(5 5) "Rno: ".
        accept rno.
        display(7 5) "Name: ".
        accept name.
        display(9 5) "Age : ".
        accept age.
        display(11 5) "SEX (1 - Male ; 2 -Female) : ".
        accept sex.
        display(13 5) "YEAR (3 - third; 4 - fourth : ".
        accept yr.
        write collrec.
        display(20 5) "Add more(y/n) :".
        accept ans.
```

25.12 LET US SUM UP

The learner in this lesson has been introduced 11 different programs that include Program for Add Corresponding, Program for Move Corresponding, Program for Condition names, Program using Occurs Clause, Program for Master File Maintenance, Program for Master File Maintenance (Alternate Key usage), Program to find Interest on Bank Deposits, Program for Inventory, Program to rewrite Pay Rate, Program for Electricity Bill and

Program to create 2 files using an Existing File. Having tried all these programs in system, the learner can definitely feel confident to write good COBOL programs on his own.

25.13 LESSON-END ACTIVITIES

Try to find the answers for the following exercises on your own.

1. Accept from the terminal the age and name of a student and if he is over 21,display that he is eligible to vote ; else display the number of years he must wait before he can vote. Also check whether the last ACCEPT was terminated by pressing f2 key or not, if f2 key was used as the terminating key, transfer control to the paragraph known as FUNC-TWO
2. The balance b in an account with a principal p and simple interest of r % after n years is given by the formula: B=P (1+NR/100). Given p in rupees calculate and display b to the nearest rupee. Assume that n and r are integers obtained through ACCEPT statement. (one program may use arithmetic Verbs and the other may use compute statement)
3. The format of the input record is as follows

columns	**fields**
1 - 10	part number
11 - 30	Description of item
31 – 36	stock quantity(xxxx.xx)
37 - 42	stock value(xxxx.xx)

(i) print the total number of records
(ii) print the heading in the format shown below:

5 BLANKS**Part NO** 5 BLANKS **Description** 5 BLANKS **Stock quantity** 5 BLANKS **stockvalue**

4. A questionnaire is distributed to participant of a course. The filled in questionnaire are manually checked and keyed-in one per card. Analyze the questionnaire and tabulate the following data:
   (1) Average age of participants.
   (2) Average experiences.
   (3) Number of participants tabulated by sex, degree.

QUESTIONNARIE

	Serial no.	2 col.
1.	Sex : Male = 1 Female = 2	1 col.
2.	Age	2 col.
3.	Martial status Single = 1 Married = 2 Other = 3	1 col.
4.	Number of Experience	2 cols.
5.	Highest degree. Bachelor = 1	1 col.

Master = 2
Doctorate = 3

5. Create a sequential file with the following record layout using the SCREEN SECTION

FIELDS	PICTURES
Order number	9(6)
Customer number	9(5)
Salesman number	9(4)
Date	9(6)
Number of items	9
Product code	x(6)
Quantity	9(5)v99

6. Display the following menu:

1.triangle
2.square
3.rectangle

Depending on the choice, display the shape filled with asterisks (*). Assume suitable size for these shapes.

7. The input record layout is given below:

Positions	fields
1-8	Account-number (alphanumeric)
9-15	Not used
16-17	Trans-code (numeric integer)
18-60	Not used.

Write a program that will sort the file in to the trans-code into the account-number order

8. The telephone department maintains the following information regarding the subscribers in a file as follows:

Columns	**Field**
1-5	Subscriber number
6-25	Subscriber name
26-41	Address
42-48	Phone-no

The above information about new subscribers is stored in a new file. Assuming the records are already stored in both the files, merge them and create a new file.

25.14 REFERENCES

1. COBOL Programming , M.K.Roy & Ghosh Dastidar , Tata McGraw Hill, 2nd Edition,1998
2. COBOL Programming , V. RAJARAMAN, PHI Pub
3. Introduction to COBOL programming – Dr. R.Krishnamoorthy, JJ Publ
4. Structured COBOL , Welburn, TataMcGraw Hill , 4th Edition.