INDEX

LESSON 1: INTRODUCTION TO DATA PROCESSING

LESSON 2: CONCEPTS OF FILES

LESSON 3: DATA STORAGE

LESSON 4: INTRODUCTION TO COBOL

LESSON 5: COBOL VERBS-I

LESSON 6: COBOL VERBS-II

LESSON 7: ADVANCED COBOL VERBS

LESSON 8: COBOL CLAUSES

LESSON 9: TABLE HANDLING- I

LESSON 10: TABLE HANDLING- II

LESSON 11: STRUCTURED PROGRAMMINIG

LESSON 12: FILES IN COBOL

LESSON 13: SORTING AND MERGING OF FILES

LESSON 14: CHARACTER HANDLING
1.0 Objectives
At the conclusion of this lesson you should be able to know:

- Data Processing
- Data & Information
- Types of Data
- Input, Processing and Output
- Architecture of Computer System
- Input Devices
- Output Devices

1.1 Introduction

Data processing is any computer process that converts data into information. The processing is usually assumed to be automated and running on a mainframe, minicomputer, microcomputer, or personal computer. Because data are most useful when well-presented and actually informative, data-processing systems are often referred to as information systems to emphasize their practicality. Nevertheless, both terms are roughly synonymous, performing similar conversions; data-processing systems typically manipulate raw data into information, and likewise information systems typically take raw data as input to produce information as output.

To better market their profession, a computer programmer or a systems analyst that might once have referred, such as during the 1970s, to the computer systems that they produce as data-processing systems more often than not nowadays refers to the computer systems that they produce by some
other term that includes the word information, such as information systems, information technology systems, or management information systems.

In the context of data processing, data are defined as numbers or characters that represent measurements from the real world. A single datum is a single measurement from the real world. Measured information is then algorithmically derived and/or logically deduced and/or statistically calculated from multiple data. Information is defined as either a meaningful answer to a query or a meaningful stimulus that can cascade into further queries.

More generally, the term *data processing* can apply to any process that converts data from one format to another, although *data conversion* would be the more logical and correct term. From this perspective, data processing becomes the process of converting information into data and also the converting of data back into information. The distinction is that conversion doesn't require a question (query) to be answered. For example, information in the form of a string of characters forming a sentence in English is converted or encoded from a keyboard's key-presses as represented by hardware-oriented integer codes into ASCII integer codes after which it may be more easily processed by a computer—not as merely raw, amorphous integer data, but as a meaningful character in a natural language's set of graphemes—and finally converted or decoded to be displayed as characters, represented by a font on the computer display. In that example we can see the stage-by-stage conversion of the presence of and then absence of electrical conductivity in the key-press and subsequent release at the keyboard from raw substantially-meaningless integer hardware-oriented data to evermore-meaningful information as the processing proceeds toward the human being.

A more conventional example of the established practice of using the term *data processing* is that a business has collected numerous data concerning an aspect of its operations and that this multitude of data must be presented in meaningful, easy-to-access presentations for the managers who must then use that information to increase revenue or to decrease cost. That conversion
and presentation of data as information is typically performed by a data-processing application.

When the domain from which the data are harvested is a science or an engineering, data processing and information systems are considered too broad of terms and the more specialized term data analysis is typically used, focusing on the highly-specialized and highly-accurate algorithmic derivations and statistical calculations that are less often observed in the typical general business environment. This divergence of culture is exhibited in the typical numerical representations used in data processing versus numerical; data processing's measurements are typically represented by integers or by fixed-point or binary-coded decimal representations of real numbers whereas the majority of data analysis's measurements are often represented by floating-point representation of real numbers.

Practically all naturally occurring processes can be viewed as examples of data processing systems where "real world" information in the form of pressure, light, etc. are converted by human observers into electrical signals in the nervous system as the senses we recognise as touch, sound, and vision. Even the interaction of non-living systems may be viewed in this way as rudimentary information processing systems. Conventional usage of the terms data processing and information systems restricts their use to refer to the algorithmic derivations, logical deductions, and statistical calculations that recur perennially in general business environments, rather than in the more expansive sense of all conversions of real-world measurements into real-world information in, say, an organic biological system or even a scientific or engineering system.

1.1.1 Data

Data are any facts, numbers, or text that can be processed by a computer. Today, organizations are accumulating vast and growing amounts of data in different formats and different databases. This includes:
• operational or transactional data such as, sales, cost, inventory, payroll, and accounting

• non-operational data, such as industry sales, forecast data, and macro economic data

• meta data - data about the data itself, such as logical database design or data dictionary definitions

1.1.2 Information

The patterns, associations, or relationships among all this data can provide information. For example, analysis of retail point of sale transaction data can yield information on which products are selling and when.

1.1.3 Types of Data

Think about any collected data that you have experience of; for example, weight, sex, ethnicity, job grade, and consider their different attributes. These variables can be described as categorical or quantitative.

The table summarizes data types and their associated measurement level, plus some examples. It is important to appreciate that appropriate methods for summary and display depend on the type of data being used. This is also true for ensuring the appropriate statistical test is employed.

<table>
<thead>
<tr>
<th>Type of data</th>
<th>Level of measurement</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Categorical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nominal</td>
<td>(no inherent order in categories)</td>
<td>Eye color, ethnicity, diagnosis</td>
</tr>
<tr>
<td>Ordinal</td>
<td>(categories have inherent order)</td>
<td>Job grade, age groups</td>
</tr>
<tr>
<td>Binary</td>
<td></td>
<td>Gender</td>
</tr>
<tr>
<td>Quantitative (Interval/Ratio) (NB units of measurement used)</td>
<td>(2 categories – special case of above)</td>
<td>Discrete (usually whole numbers)</td>
</tr>
</tbody>
</table>

Table 1.1 Types of Data

1.2. Input, Processing and output

Whenever a computer is used it must work its way through three basic stages before any task can be completed. These are input, processing and output. A computer works through these stages by running a program. A program is a set of step-by-step instructions which tells the computer exactly what to do with the input in order to produce the required output.

1.2.1 Input

The input stage of computing is concerned with getting the data needed by the program into the computer. Input devices are used to do this. The most commonly used input devices are the mouse and the keyboard.

1.2.2 Processing

The program contains instructions about what to do with the input. During the processing stage the computer follows these instructions using the data which has just been input. What the computer produces at the end of this stage, the output, will only be as good as the instructions given in the program. In other words if garbage has been put in to the program, garbage is what will come out of the computer. This is known as GIGO, or Garbage In Garbage Out.
1.2.3 Output

The output stage of computing is concerned with giving out processed data as information in a form that is useful to the user. Output devices are used to do this. The most commonly used output devices are the screen, which is also called a monitor or VDU and the printer.

1.3. Architecture of Computer System

This is the 'brain' of the computer. It is where all the searching, sorting, calculating and decision making takes place. The CPU collects all of the raw data from various input devices (such a keyboard or mouse) and converts it into useful information by carrying out software instructions. The result of all that work is then sent to output devices such as monitors and printers.

The CPU is a microprocessor - a silicon chip - composed of tiny electrical switches called 'transistors'. The speed at which the processor carries out its operations is measured in megahertz (MHz) or Gigahertz (GHz). The higher the number of MHz the faster the computer can process information. A common CPU today runs at around 3 GHz or more.

The Intel Pentium processor and the Athlon are examples of a CPU.
1.3.1 The Control Unit (CU)

The Control Unit (CU) co-ordinates the work of the whole computer system.

It has three main jobs:

1. It controls the hardware attached to the system. The Control Unit monitors the hardware to make sure that the commands given to it by the current program are activated.

2. It controls the input and output of data, so all the signals go to the right place at the right time.

3. It controls the flow of data within the CPU.

1.3.2 The Immediate Access Store (IAS)

The Immediate Access Store (IAS) holds the data and programs needed at that instant by the Control Unit. The CPU reads data and programs kept on the backing storage and store them temporarily in the IAS’s memory.

The CPU needs to do this because Backing Store is much too slow to be able to run data and programs from directly. For example, lets pretend that a modern CPU was slowed down to carry out one instruction in 1 second, then the hard disk (ie Backing Store) would take 3 months to supply the data it needs!

So the trick is to call in enough of the data and programs into fast Immediate Access Store memory so as to keep the CPU busy.

1.3.3 ALU stands for Arithmetic and Logic Unit.

It is where the computer processes data by either manipulating it or acting upon it. It has two parts:
1. **Arithmetic** part - does exactly what you think it should - it does the calculations on data such as 3 + 2.

2. **Logic** part - This section deals with carrying out logic and comparison operations on data. For example working out if one data value is bigger than another data value.

### 1.4. Input Devices

Due to a constant research in the computer hardware we have a large number of input devices recall that before data can be processed by the computer they must be translated into machine readable form and entered into the computer by an input device. Here we will introduce a variety of input devices.

#### 1.4.1 Keyboard

The **keyboard** is the most widely used input device and is used to enter data or commands to the computer. It has a set of alphabet keys, a set of digit keys, and various function keys and is divided into four main areas:

- Function keys across the top
- Letter keys in the main section
- A numeric keypad on the right
- Cursor movement and editing keys between the main section and the numeric keypad.

The layout of the letters on a keyboard is standard across many countries and is called a **QWERTY** keyboard. The name comes from the first six keys on the top row of the alphabetic characters.

Some keyboards come with added keys for using the Internet and others have an integrated wrist support. Ergonomic keyboards have been developed to
reduce the risk of repetitive strain injury to workers who use keyboards for long periods of time.

The computer's processor scans the keyboard hundreds of times per second to see if a key has been pressed. When a key is pressed, a digital code is sent to the Central Processing Unit (CPU). This digital code is translated into ASCII code (American Standard Code of Information Interchange).

For example, pressing the 'A' key produces the binary code 01100001 representing the lower case letter 'a'. Holding down the shift key at the same time produces the binary code 01000001 representing the upper case letter 'A'.

Advantages:

- Most computers have this device attached to it
- It is a reliable method for data input of text and numbers
- A skilled typist can enter data very quickly.
- Specialist keyboards are available

Disadvantages:

- It is very easy to make mistakes when typing data in
- It can be very time consuming to enter data using a keyboard, especially if you are not a skilled typist.
- It is very difficult to enter some data, for example, details of diagrams and pictures.
- It is very slow to access menus and not flexible when you want to move objects around the screen.
- Difficult for people unable to use keyboards through paralysis or muscular disorder.

1.4.2 Mouse
A mouse is the most common pointing device that you will come across. It enables you to control the movement and position of the on-screen cursor by moving it around on the desk.

Buttons on the mouse let you select options from menus and drag objects around the screen. Pressing a mouse button produces a 'mouse click'. You might have heard the expressions 'double click', 'click and drag' and 'drag and drop'.

Most mice use a small ball located underneath them to calculate the direction that you are moving the mouse in. The movement of the ball causes two rollers to rotate inside the mouse; one records the movement in a north-south direction and the other records the east-west movement. The mouse monitors how far the ball turns and in what direction and sends this information to the computer to move the pointer.

**Advantages:**

- Ideal for use with desktop computers.
- Usually supplied with a computer so no additional cost.
- All computer users tend to be familiar with using them.

**Disadvantages:**

- They need a flat space close to the computer.
- The mouse cannot easily be used with laptop, notebook or palmtop computers. (These need a tracker ball or a touch sensitive pad called a touch pad).

**1.4.3 Trackball**

A tracker ball is like an upside down mouse with the ball on top. Turning the ball with your hand moves the pointer on the
screen. It has buttons like a standard mouse, but requires very little space to operate and is often used in conjunction with computer aided design. You will often find a small tracker ball built into laptop computers in place of the conventional mouse.

**Advantages:**

- Ideal for use where flat space close to the computer is limited.
- Can be useful with laptops as they can be built into the computer keyboard or clipped on.

**Disadvantages:**

- Not supplied as standard so an additional cost and users have to learn how to use them

1.4.4 Joystick

A **Joystick** is similar to a tracker ball in operation except you have a stick which is moved rather than a rolling ball.

Joysticks are used to play computer games. You can move a standard joystick in any one of eight directions. The joystick tells the computer in which direction it is being pulled and the computer uses this information to (for example) move a racing car on screen. A joystick may also have several buttons which can be pressed to trigger actions such as firing a missile.

**Advantages:**

- There is an immediate feel of direction due to the movement of the stick

**Disadvantages:**
• Some people find the joystick difficult to control rather than other point and click devices. This is probably because more arm and wrist movement is required to control the pointer than with a mouse or tracker ball.
• Joysticks are not particularly strong and can break easily when used with games software.

1.4.5 Touch Screen

These screens do a similar job to concept keyboards. A grid of light beams or fine wires criss-cross the computer screen. When you touch the screen with your finger, the rays are blocked and the computer 'senses' where you have pressed. Touch screens can be used to choose options which are displayed on the screen.

Touch screens are easy to use and are often found as input devices in public places such as museums, building societies (ATMs), airports or travel agents. However, they are not commonly used elsewhere since they are not very accurate, tiring to use for a long period and are more expensive than alternatives such as a mouse.

Advantages:

• Easy to use
• Software can alter the screen while it is running, making it more flexible than a concept keyboard with a permanent overlay
• No extra peripherals are needed apart from the touch screen monitor itself.
• No experience or competence with computer systems are needed to be able to use it.

Disadvantages:
• Not suitable for inputting large amounts of data
• Not very accurate, selecting detailed objects can be difficult with fingers
• Tiring to use for a long period of time
• More expensive than alternatives such as a mouse.
• Touch screens are not robust and can soon become faulty.

1.4.6 Digital Camera

A digital camera looks very similar to a traditional camera. However, unlike photographic cameras, digital cameras do not use film. Inside a digital camera is an array of light sensors. When a picture is taken, the different colors that make up the picture are converted into digital signals (binary) by sensors placed behind the lens.

Most digital cameras let you view the image as soon as you have taken the picture and, if you don't like what you see, it can be deleted. The image can then be stored in the camera's RAM or on a floppy disk. Later, the pictures can be transferred onto a computer for editing using photo imaging software.

The amount of memory taken up by each picture depends on its resolution. The resolution is determined by the number of dots which make up the picture: the greater the number of dots which make up the picture, the clearer the image. However, higher resolution pictures take up more memory (and are more expensive!).

Resolution range from about 3 million (or Mega) pixels up to 12 Mega pixels

Digital cameras are extremely useful for tasks such as producing newsletters.

There is often a digital camera built into mobile phones that operates in exactly the same way as a standard one.

Advantages:

• No film is needed and there are no film developing costs
• Unwanted images can be deleted straight away
• You can edit, enlarge or enhance the images
• Images can be incorporated easily into documents, sent by e-mail or added to a website.

Disadvantages:

• Digital cameras are generally more expensive than ordinary cameras.
• Images often have to be compressed to avoid using up too much expensive memory
• When they are full, the images must be downloaded to a computer or deleted before any more can be taken.

1.4.7 Scanner

A scanner is another way in which we can capture still images or text to be stored and used on a computer. Images are stored as 'pixels'.

A scanner works by shining a beam of light on to the surface of the object you are scanning. This light is reflected back on to a sensor that detects the color of the light.

The reflected light is then digitized to build up a digital image.

Scanner software usually allows you to choose between a high resolution (very high quality images taking up a lot of memory) and lower resolutions.

Special software can also be used to convert images of text into actual text data which can be edited by a word processor. This software is called an "Optical Character Reader" or OCR.

There are two types of scanner:

• Flatbed Scanner
Handheld Scanner

The most popular type of scanner is the flatbed. It works in a similar way to a photocopier. Flatbed scanners can scan larger images and are more accurate than handheld scanners.

Handheld scanners are usually only a few inches wide and are rolled across the document to be scanned. They perform the same job but the amount of information that can be scanned is limited by the width of the scanner and the images produced are not of the same quality as those produced by flatbed scanners.

Advantages:

- Flat-bed scanners are very accurate and can produce images with a far higher resolution than a digital camera
- Any image can be converted from paper into digital format and later enhanced and used in other computer documents.

Disadvantages:

- Images can take up a lot of memory space.
- The quality of the final image depends greatly upon the quality of the original document.

1.4.8 Graphics Tablets

Graphics tablets are often used by graphics designers and illustrators. Using a graphics tablet a designer can produce much more accurate drawings on the screen than they could with a mouse or other pointing device.

A graphics tablet consists of a flat pad (the tablet) on which you draw with a special pen. As you draw on the pad the image is created on the screen. By using a graphics tablet a designer can produce very accurate on-screen drawings.
Drawings created using a graphics tablet can be accurate to within hundredths of an inch.

The 'stylus' or pen that you use may have buttons on it that act like a set of mouse buttons. Sometimes, instead of a stylus a highly accurate mouse-like device called a puck is used to draw on the tablet.

**Advantages:**

- In a design environment where it is more natural to draw diagrams with pencil and paper, it is an effective method of inputting the data into the computer.

**Disadvantages:**

- Not as good as a mouse for clicking on menu items.

1.5. Output Devices

Once data has been input into a computer and processed, it is of little use unless it can be retrieved quickly and easily from the system. To allow this, the computer must be connected to an output device.

The most common output devices are computer monitors and printers. However, output can also be to a modem, a plotter, speakers, a computer disk, another computer or even a robot.

1.5.1 Monitor

A Monitor (or "screen") is the most common form of output from a computer. It displays information in a similar way to that shown on a television screen.
On a typical computer the monitor may measure 17 inches (43 cm) across its display area. Larger monitors make working at a computer easier on the eyes. Of course the larger the screen, the higher its cost! Typical larger sizes are 19 inch, 20 inch and 21 inches.

Part of the quality of the output on a monitor depends on what resolution it is capable of displaying. Other factors include how much contrast it has, its viewing angle and how fast does it refresh the screen. For example a good computer game needs a fast screen refresh so you can see all the action.

The picture on a monitor is made up of thousands of tiny colored dots called pixels. The quality and detail of the picture on a monitor depends on the number of pixels that it can display. The more dense the pixels the greater the clarity of the screen image.

A PC monitor contains a matrix of dots of Red, Green and Blue known as RGB. these can be blended to display millions of colors.

This is one RGB pixel of light

\[
\begin{align*}
R + B &= M \text{ (magenta)} \\
B + G &= C \text{ (cyan)} \\
G + R &= Y \text{ (yellow)} \\
R + G + B &= W \text{ (white)}
\end{align*}
\]

The two most common types of monitor are a cathode-ray tube (CRT) monitor and a liquid crystal display (LCD).

**Liquid Crystal Display (or "TFT" Display)**

This is smaller and lighter than the CRT (see below), which makes them ideal for use with portable laptops, PDAs and Palmtops. Even desktop computers are using them now that their price has become comparable to CRT monitors.
**Liquid Crystal** is the material used to create each pixel on the screen. The material has a special property - it can 'polarize' light depending on the electrical charge across it. Charge it one way and all the light passing through it is set to "vertical" polarity, charge it another way and the light polarity is set to "horizontal". This feature allows the pixels to be created. Each tiny cell of liquid crystal is a pixel.

TFT (or Thin Film Transistor) is the device within each pixel that sets the charge. And so sometimes they are called "Liquid Crystal Display" referring to the material they use or they are called "TFT displays" referring to the tiny transistors that make them work.

LCDs use much less power than a normal monitor.

**Cathode Ray Tube**

The CRT works in the same way as a television - it contains an electron gun at the back of the glass tube. This fires electrons at groups of phosphor dots which coat the inside of the screen. When the electrons strike the phosphor dots they glow to give the colors.

**Advantages of monitors**

- Relatively cheap
- Reliable
- Can display text and graphics in a wide range of colours
- As each task is processed, the results can be displayed immediately on the screen
- Output can be scrolled backwards and forwards easily.
- Quiet
- Do not waste paper

**Disadvantages of monitors:**
• No permanent copy to keep - the results will disappear when the computer is switched off.
• Unsuitable for users with visual problems.
• Only a limited amount of information can be displayed at any one time
• Screens are made of glass and can be very fragile.

1.5.2 Printers

Printers are output devices. They are dedicated to creating paper copies from the computer.

Printers can produce text and images on paper. Paper can be either separate sheets such as A4 A5 A3 etc. or they may be able to print on continuous (fanfold) paper that feed through the machine.

A ream of A4 paper

Continuous paper with holes on the edges, used by dot matrix printers. After you print on fanfold paper, you have to separate the pages and tear off the edge strips

Very specialist printers can also print on plastic or even textiles such as T-shirts.

Some printers are dedicated to only producing black and white output. Their advantage is that they are often faster than a color printer because effectively there is only one color to print (Black).
Color Printers are dedicated to creating text and images in full color. Some types can even produce photographs when special paper is used.

There are three main types of printer that you need to know about. You will be expected to understand the main differences i.e. purchase costs, running costs, quality and speed.

The three types are Laser, Dot Matrix and Inkjet.

1.5.3 Plotter

These are output devices that can produce high quality line diagrams on paper. They are often used by engineering, architects and scientific organizations to draw plans, diagrams of machines and printed circuit boards.

A plotter differs from a printer in that it draws images using a pen that can be lowered, raised and moved across the page to form continuous lines. The electronically controlled pen is moved by two computer-controlled motors. The pen is lifted on and off the page by switching an electromagnet on and off.

The paper is handled in different ways depending on the type of plotter.

Flatbed plotters hold the paper still while the pens move.

Drum plotters roll the paper over a cylinder

Pinch-roller plotters are a mixture of the two.

Advantages:
Drawings are of the same quality as if an expert drew them
Larger sizes of paper can be used than would be found on most printers

Disadvantages:

- Plotters are slower than printers, drawing each line separately.
- They are often more expensive to buy than printers
- Although drawings are completed to the highest quality they are not suitable for text (although text can be produced)
- There is a limit to the amount of detail these plotters can produce, although there are plotters which are "pen-less" the set are used for high-density drawings as may be used for printed circuit board layout.
- In recent years, cheaper printers that can handle A3 and A2 sized paper have resulted in a decline in the need for smaller plotters.

1.6 Summary

- **Data processing** is any computer process that converts data into information.
- Data are any facts, numbers, or text that can be processed by a computer.
- The patterns, associations, or relationships among all this data can provide information.
- The CPU is a microprocessor - a silicon chip - composed of tiny electrical switches called 'transistors'.
- The **keyboard** is the most widely used input device and is used to enter data or commands to the computer.
- A **Joystick** is similar to a tracker ball in operation except you have a stick which is moved rather than a rolling ball.
- **Graphics tablets** are often used by graphics designers and illustrators.
- The most common output devices are computer monitors and printers.
• Meta data - data about the data itself, such as logical database design or data dictionary definitions.
• Resolution of a digital camera range from about 3 million (or Mega) pixels up to 12 Mega pixels.

1.7 Key words
• Operational Data - Operational or transactional data such as, sales, cost, inventory, payroll, and accounting.
• Non-operational Data - non-operational data, such as industry sales, forecast data, and macro economic data.
• Input - The input stage of computing is concerned with getting the data needed by the program into the computer.
• Output - The output stage of computing is concerned with giving out processed data as information in a form that is useful to the user.
• Pixels - The picture on a monitor is made up of thousands of tiny colored dots called pixels.

1.8 Self Assessment Questions (SAQ)
• What do you mean by information? How it is different from data? Explain.
• Explain the process of input – processing - output with the help of suitable examples.
• Explain the architecture of a Computer System.
• Explain what is meant by the term input device? Give three examples of input devices. Also give possible advantages and disadvantage of the same.
• Explain what is meant by the term output device? Give three examples of output devices. Also give possible advantages and disadvantage of the same.
• What are different types of printers? How a plotter is different from a printer?

1.9 References/Suggested Readings
• Computer Fundamental, P.K. Sinha, BPB Publications – 2004
• Sams Teach Yourself COBOL in 24 Hours, Hubbell, Sams, Dec 1998
• Structured COBOL Methods, Noll P, Murach, Sep 1998
• ICT for you, Stephon Doyle, Nelson Thornes, 2003
• Information and Communication Technology, Denise Walmsley, Hodder Murray 2004
• Information Technology, P Evans, BPB Publications, 2000
LESSON 2
CONCEPTS OF FILES

2.0 Objectives

At the conclusion of this lesson you should be able to know:

- File
- File Contents
- Operations on the file
- File Organization
- Storing Files
- Backing-up files
- File Terminology
- Data Capturing
- Data Verification
- Data Validation

2.1. Introduction

A computer file is a piece of arbitrary information, or resource for storing information, that is available to a computer program and is usually based on some kind of durable storage. A file is *durable* in the sense that it remains available for programs to use after the current program has finished. Computer files can be considered as the modern counterpart of the files of printed documents that traditionally existed in offices and libraries.

2.1.1. File contents

As far as the operating system is concerned, a file is in most cases just a sequence of binary digits. At a higher level, where the content of the file is being considered, these binary digits may represent integer values or text
characters, it is up to the program using the file to understand the meaning and internal layout of information in the file and present it to a user as a document, image, song, or program.

At any instant in time, a file has might have a size, normally expressed in bytes, that indicates how much storage is associated with the file.

Information in a computer file can consist of smaller packets of information (often called records or lines) that are individually different but share some trait in common. For example, a payroll file might contain information concerning all the employees in a company and their payroll details; each record in the payroll file concerns just one employee, and all the records have the common trait of being related to payroll—this is very similar to placing all payroll information into a specific filing cabinet in an office that does not have a computer. A text file may contain lines of text, corresponding to printed lines on a piece of paper.

The way information is grouped into a file is entirely up to the person designing the file. This has led to a plethora of more or less standardized file structures for all imaginable purposes, from the simplest to the most complex. Most computer files are used by computer programs. These programs create, modify and delete files for their own use on an as-needed basis. The programmers who create the programs decide what files are needed, how they are to be used and (often) their names.

In some cases, computer programs manipulate files that are made visible to the computer user. For example, in a word-processing program, the user manipulates document files that she names herself. The content of the document file is arranged in a way that the word-processing program understands, but the user chooses the name and location of the file, and she provides the bulk of the information (such as words and text) that will be stored in the file.

Files on a computer can be created, moved, modified, grown, shrunk and deleted. In most cases, computer programs that are executed on the
computer handle these operations, but the user of a computer can also manipulate files if necessary. For instance, Microsoft Word files are normally created and modified by the Microsoft Word program in response to user commands, but the user can also move, rename, or delete these files directly by using a file manager program such as Windows Explorer (on Windows computers).

2.1.2. Operations on the file

Opening a file to use its contents

Reading or updating the contents

Committing updated contents to durable storage

Closing the file, thereby losing access until it is opened again

2.1.3 File Organization

2.1.3.1 Sequential file

Access to records in a Sequential file is serial. To reach a particular record, all the preceding records must be read.

As we observed when the topic was introduced earlier in the course, the organization of an unordered Sequential file means it is only practical to read records from the file and add records to the end of the file (OPEN..EXTEND). It is not practical to delete or update records.

While it is possible to delete, update and insert records in an ordered Sequential file, these operations have some drawbacks.

2.1.3.1.1 Problems accessing ordered Sequential files

Records in an ordered Sequential file are arranged, in order, on some key field or fields. When we want to insert, delete or amend a record we must preserve the ordering. The only way to do this is to create a new file. In the case of an insertion or update, the new file will contain the inserted or updated record. In the case of a deletion, the deleted record will be missing from the new file.
The main drawback to inserting, deleting or amending records in an ordered Sequential file is that the entire file must be read and then the records written to a new file. Since disk access is one of the slowest things we can do in computing this is very wasteful of computer time when only a few records are involved.

For instance, if 10 records are to be inserted into a 10,000 record file, then 10,000 records will have to be read from the old file and 10,010 written to the new file. The average time to insert a new record will thus be very great.

2.1.3.1.2 Inserting records in an ordered Sequential file

To insert a record in an ordered Sequential file:

1. All the records with a key value less than the record to be inserted must be read and then written to the new file.
2. Then the record to be inserted must be written to the new file.
3. Finally, the remaining records must be written to the new file.

2.1.3.1.3 Deleting records from an ordered Sequential file

To delete a record in an ordered Sequential file:

1. All the records with a key value less than the record to be deleted must be written to the new file.
2. When the record to be deleted is encountered it is not written to the new file.
3. Finally, all the remaining records must be written to the new file.

2.1.3.1.4 Amending records in an ordered Sequential file

To amend a record in an ordered Sequential file:

1. All the records with a key value less than the record to be amended must be read and then written to the new file.
2. Then the record to be amended must be read the amendments applied to it and the amended record must then be written to the new file.
3. Finally, all the remaining records must be written to the new file.

2.1.3.2 Relative File
As we have already noted, the problem with Sequential files is that access to the records is serial. To reach a particular record, all the proceeding records must be read.

Direct access files allow direct access to a particular record in the file using a key and this greatly facilitates the operations of reading, deleting, updating and inserting records.

COBOL supports two kinds of direct access file organizations - Relative and Indexed.

### 2.1.3.2.1 Organization of Relative files

Records in relative files are organized on ascending Relative Record Number. A Relative file may be visualized as a one dimension table stored on disk, where the Relative Record Number is the index into the table. Relative files support sequential access by allowing the active records to be read one after another.

Relative files support only one key. The key must be numeric and must take a value between 1 and the current highest Relative Record Number. Enough room is allocated to the file to contain records with Relative Record Numbers between 1 and the highest record number.

For instance, if the highest relative record number used is 10,000 then room for 10,000 records is allocated to the file.

Figure 1 below contains a schematic representation of a Relative file. In this example, enough room has been allocated on disk for 328 records. But although there is room for 328 records in the current allocation, not all the record locations contain records. The record areas labeled "free", have not yet had record values written to them.

Relative File - Organization
Figure 1

2.1.3.2.2 Accessing records in a Relative file
To access a record in a Relative file a Relative Record Number must be provided. Supplying this number allows the record to be accessed directly because the system can use

the start position of the file on disk,
the size of the record,
and the Relative Record Number

to calculate the position of the record.

Because the file management system only has to make a few calculations to find the record position the Relative file organization is the fastest of the two direct access file organizations available in COBOL. It is also the most storage efficient.

2.1.3.3 Indexed Files

While the usefulness of a Relative file is constrained by its restrictive key, Indexed files suffer from no such limitation.

Indexed files may have up to 255 keys, the keys can be alphanumeric and only the primary key must be unique.

In addition, it is possible to read an Indexed file sequentially on any of its keys.

2.1.3.3.1 Organization of Indexed files

An Indexed file may have multiple keys. The key upon which the data records are ordered is called the primary key. The other keys are called alternate keys.

Records in the Indexed file are sequenced on ascending primary key. Over the actual data records, the file system builds an index. When direct access is required, the file system uses this index to find, read, insert, update or delete, the required record.

For each of the alternate keys specified in an Indexed file, an alternate index is built. However, the lowest level of an alternate index does not contain actual data records. Instead, this level made up of base records which contain only the alternate key value and a pointer to where the actual record is. These base records are organized in ascending alternate key order.
As well as allowing direct access to records on the primary key or any of the 254 alternate keys, indexed files may also be processed sequentially. When processed sequentially, the records may be read in ascending order on the primary key or on any of the alternate keys.

Since the data records are in held in ascending primary key sequence it is easy to see how the file may be accessed sequentially on the primary key. It is not quite so obvious how sequential on the alternate keys is achieved. This is covered in the unit on Indexed files.

Organizing files and folders

Files and folders arranged in a hierarchy

In modern computer systems, files are typically accessed using names. In some operating systems, the name is associated with the file itself. In others, the file is anonymous, and is pointed to by links that have names. In the latter case, a user can identify the name of the link with the file itself, but this is a false analogue, especially where there exists more than one link to the same file.

Files (or links to files) can be located in directories. However, more generally, a directory can contain either a list of files, or a list of links to files. Within this definition, it is of paramount importance that the term "file" includes
directories. This permits the existence of directory hierarchies. A name that refers to a file within a directory must be unique. In other words, there must be no identical names in a directory. However, in some operating systems, a name may include a specification of type that means a directory can contain an identical name to more than one type of object such as a directory and a file.

In environments in which a file is named, a file’s name and the path to the file’s directory must uniquely identify it among all other files in the computer system—no two files can have the same name and path. Where a file is anonymous, named references to it will exist within a namespace. In most cases, any name within the namespace will refer to exactly zero or one file. However, any file may be represented within any namespace by zero, one or more names.

Any string of characters may or may not be a well-formed name for a file or a link depending upon the context of application. Whether or not a name is well-formed depends on the type of computer system being used. Early computers permitted only a few letters or digits in the name of a file, but modern computers allow long names (some up to 255) containing almost any combination of unicode letters or unicode digits, making it easier to understand the purpose of a file at a glance. Some computer systems allow file names to contain spaces; others do not. Such characters such as / or \ are forbidden. Case-sensitivity of file names is determined by the file system.

Most computers organize files into hierarchies using folders, directories, or catalogs. (The concept is the same irrespective of the terminology used.) Each folder can contain an arbitrary number of files, and it can also contain other folders. These other folders are referred to as subfolders. Subfolders can contain still more files and folders and so on, thus building a tree-like structure in which one “master folder” (or “root folder” — the name varies from one operating system to another) can contain any number of levels of other folders and files. Folders can be named just as files can (except for the root folder, which often does not have a name). The use of folders makes it easier to organize files in a logical way.
Protecting files

Many modern computer systems provide methods for protecting files against accidental and deliberate damage. Computers that allow for multiple users implement file permissions to control who may or may not modify, delete, or create files and folders. A given user may be granted only permission to modify a file or folder, but not to delete it; or a user may be given permission to create files or folders, but not to delete them. Permissions may also be used to allow only certain users to see the contents of a file or folder. Permissions protect against unauthorized tampering or destruction of information in files, and keep private information confidential by preventing unauthorized users from seeing certain files.

Another protection mechanism implemented in many computers is a read-only flag. When this flag is turned on for a file (which can be accomplished by a computer program or by a human user), the file can be examined, but it cannot be modified. This flag is useful for critical information that must not be modified or erased, such as special files that are used only by internal parts of the computer system. Some systems also include a hidden flag to make certain files invisible; this flag is used by the computer system to hide essential system files that users must never modify.

2.1.6 Storing files

In physical terms, most computer files are stored on hard disks—spinning magnetic disks inside a computer that can record information indefinitely. Hard disks allow almost instant access to computer files.

On large computers, some computer files may be stored on magnetic tape. Files can also be stored on other media in some cases, such as writeable compact discs, Zip drives, etc.
2.1.7 Backing up files

When computer files contain information that is extremely important, a backup process is used to protect against disasters that might destroy the files. Backing up files simply means making copies of the files in a separate location so that they can be restored if something happens to the computer, or if they are deleted accidentally.

There are many ways to back up files. Most computer systems provide utility programs to assist in the back-up process, which can become very time-consuming if there are many files to safeguard. Files are often copied to removable media such as writeable CDs or cartridge tapes. Copying files to another hard disk in the same computer protects against failure of one disk, but if it is necessary to protect against failure or destruction of the entire computer, then copies of the files must be made on other media that can be taken away from the computer and stored in a safe, distant location.

2.2. File Terminology

There are a few terms that you need to understand when learning about file system. These will be explained over the next couple of pages.

File can store data or information in various formats. Suppose in a file data is stored in the tables just like the one below:
2.2.1 Records

As you saw previously, each table stores can hold a great deal of data.

Each table contains a lot of records.

A record is all of the data or information about one person or one thing.

In the table below, all of the information about each cartoon character is stored in a 'row' or record.

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Address</th>
<th>City</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mickey</td>
<td>Mouse</td>
<td>123 Fantasy Way</td>
<td>Anaheim</td>
<td>73</td>
</tr>
<tr>
<td>Bat</td>
<td>Man</td>
<td>321 Cavern Ave</td>
<td>Gotham</td>
<td>54</td>
</tr>
<tr>
<td>Wonder</td>
<td>Woman</td>
<td>987 Truth Way</td>
<td>Paradise</td>
<td>39</td>
</tr>
<tr>
<td>Donald</td>
<td>Duck</td>
<td>555 Quack Street</td>
<td>Mallard</td>
<td>65</td>
</tr>
<tr>
<td>Bugs</td>
<td>Bunny</td>
<td>567 Carrot Street</td>
<td>Rascal</td>
<td>58</td>
</tr>
<tr>
<td>Wiley</td>
<td>Coyote</td>
<td>999 Acme Way</td>
<td>Canyon</td>
<td>61</td>
</tr>
<tr>
<td>Cat</td>
<td>Woman</td>
<td>234 Purfect Street</td>
<td>Hairball</td>
<td>32</td>
</tr>
<tr>
<td>Tweety</td>
<td>Bird</td>
<td>543</td>
<td>Itohka</td>
<td>28</td>
</tr>
</tbody>
</table>

What information could you find in the record for Cat Woman?

What do you think the database at your school stores records about?

How about the library? What records would be stored on that database?
2.2.2 Fields

Each table contains a lot of records.

A record is made up of lots of individual pieces of information. Look at Wonder Woman's record; it stores her first name, last name, address, city and age.

Each of these individual pieces of information in a record are called a 'field'

A 'field' is one piece of data or information about a person or thing.

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Address</th>
<th>City</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mickey</td>
<td>Mouse</td>
<td>123 Fantasy Way</td>
<td>Anaheim</td>
<td>73</td>
</tr>
<tr>
<td>Bat</td>
<td>Man</td>
<td>321 Cavern Ave</td>
<td>Gotham</td>
<td>54</td>
</tr>
<tr>
<td>Wonder</td>
<td>Woman</td>
<td>987 Truth Way</td>
<td>Paradise</td>
<td>39</td>
</tr>
<tr>
<td>Donald</td>
<td>Duck</td>
<td>555 Quack Street</td>
<td>Mallard</td>
<td>65</td>
</tr>
<tr>
<td>Bugs</td>
<td>Bunny</td>
<td>567 Carrot Street</td>
<td>Rascal</td>
<td>58</td>
</tr>
<tr>
<td>Wiley</td>
<td>Coyote</td>
<td>999 Acme Way</td>
<td>Canyon</td>
<td>61</td>
</tr>
<tr>
<td>Cat</td>
<td>Woman</td>
<td>234 Purrfect Street</td>
<td>Hairball</td>
<td>32</td>
</tr>
<tr>
<td>Tweety</td>
<td>Bird</td>
<td>543</td>
<td>Itotitaw</td>
<td>28</td>
</tr>
</tbody>
</table>

What fields can you find about Tweety Bird?

What fields do you think would be stored in your student record on the school database?

What fields would be stored in a book record in the library database?

2.3. Data Capturing
Any database or information system needs data entered into it, in order for it to be of any use.

There are many methods which can be used to collect and enter data, some manual, some automatic.

We will also look in particular detail at designing an effective paper-based data capture form.

### 2.3.1 Direct Data Capturing

Here are some of the methods that can be used to capture data directly.

#### 2.3.1.1 Barcode reader

A bar code reader uses visible red light to scan and 'read' the barcode. As the red light shines across the light and dark bands of the barcode, so the reflected red light is also lighter and darker (do you see that on the picture opposite?)

The Hand Scanner senses the reflected light and translates it into digital data. The digital data is then input into the computer. The computer may display the results on a screen and also input it into the correct fields in the database.

Typical uses:

Shop - to find details on the product sold and price

Library - record the ISBN number of the book and the borrower's card number

Warehouse - to check the lables on boxes delivered against what is recorded on the delivery sheet.

#### 2.3.1.2 Magnetic ink character recognition (MICR)
The numbers at the bottom of a cheque are written in a special ink which contains iron particles. This ink is magnetised and commonly called 'magnetic ink'. It can be read by a special machine called a Magnetic Ink Character Reader (MICR).

2.3.1.3 Optical Mark Readers (OMR)

An Optical Mark Reader is a scanning device that reads carefully placed pencil marks on a specially designed form or document.

A simple pen or pencil mark is made on the form to indicate the correct choice e.g. a multiple choice exam paper or on the National Lottery ticket selection form.

The completed forms are scanned by an Optical Mark Reader (OMR) which detects the presence of a mark by measuring the reflected light. Less light is reflected where a mark has been made.

The OMR then interprets the pattern of marks into a data record and sends this to the computer for storage, analysis and reporting.

This provides a very fast and accurate method of inputting large amounts of data, provided the marks have been made accurately and clearly.

2.3.1.4 Optical Character Recognition (OCR)

Optical Character Recognition (OCR) enables the computer to identify written or printed characters.

An OCR system consists of a normal scanner and some special software. The scanner is used to scan the text from a document into the computer. The software then examines the page and extracts the text from it, storing it in
a form that can be edited or processed by normal word processing software.

The ability to scan the characters accurately depends on how clear the writing is. Scanners have been improved to be able to read different styles and sizes of text as well as neat handwriting. Although they are often up to 95% accurate, any text scanned with OCR needs careful checking because some letters can be misread.

OCR is also used to automatically recognise postcodes on letters at sorting offices.

2.3.1.5 Speech Recognition

The user talks into a microphone. The computer 'listens' to the speaker, then translates that information to written words and phrases. It then displays the text on to the monitor.

This process happens immediately, so as you say the words, they appear on the screen. The software often needs some "training" in order for it to get used to your voice, but after that it is simple to use.

2.3.2 Data Capture Forms

Although there are many methods of capturing data automatically, many businesses prefer to capture it manually.

2.3.2.1 Paper-based data capture forms

This is the most commonly used method of collecting or capturing data.
People are given a form to fill in with their personal details, e.g. name, address, telephone number, date of birth etc.

Once the form is completed, it is given to a member of staff who will enter the data from it, into a database or information system.

2.3.2.2 Computerised data entry forms

A member of staff could type the information directly into a computerised data entry form whilst the customer is with them. They ask the question in the order it appears on the form and enter the answer using a keyboard.

More commonly though, the details will be typed in by copying what was written on the paper-based data capture form. When this method is used, it is important that the fields on both forms are laid out in the same order to speed up the process of entering the data.

2.3.3 Designing Data Capture Form

A data capture form looks simple enough to design, don't you just type out a few questions, put a couple of boxes for customers to fill in their information and then print it out? No, it's not as simple as that. If you want to collect good quality data, you need to think carefully about the design of the form.

All forms should have the name of the organisation at the top.

---

**GREYSIDE YOUTH CLUB**

Membership Application Form

Please complete the top section of this membership application form. Then send the form to The Youth Club, 15 Greyside Park, Liverpool, Merseyside, L2 3XJ.
They should also have an explanation to tell the customer what the form is for, in this case 'membership application form', or 'data collection form', or 'customer details form' or something similar.

Lastly, they should give the customer instructions to tell them what they should do with the form once they have completed it. Here it tells the person filling the form in, to send it back to the address given.

Where possible, it is a good idea to try to limit the options that people can enter. If you can manage to do this, then you can set up your computerised system with a drop down box that gives all of the options on the form - making it faster for staff to enter the data.

For Example: The first form shown above, limits the choice of title to 'Mr' or 'Miss'. This is sufficient in this case because it is an application form for a childrens' youth club, so it is unlikely that there will be any 'Mrs' or 'Dr' or 'Reverend'

The second form gives people the different options for travel, they have to tick one of the options since there isn't any room for them to write something different. The same method has been used for types of lunches.

2.4. Verification

It was mentioned that validation cannot make sure that data you enter is correct, it can only check that it is sensible, reasonable and allowable.
However, it is important that the data in your database is as accurate as possible. Have you ever heard of the term 'Garbage in, garbage out' or 'GIGO'? This means that if you enter data that is full of mistakes (garbage in) then when you want to search for a record you will get data with mistakes presented to you (garbage out).

This is where Verification can help to make sure that the data in your database contains as few mistakes as possible.

Verification means to check something twice.

Think about when you choose a new password, you have to type it in twice. This lets the computer check if you have typed it exactly the same both times and not made a mistake.

The data in your database can be verified or checked twice.

<table>
<thead>
<tr>
<th>FRUIT</th>
<th>COLOUR</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>Yellow</td>
<td>Medium</td>
</tr>
<tr>
<td>Orange</td>
<td>Orange</td>
<td>Medium</td>
</tr>
<tr>
<td>Strawberry</td>
<td>Red</td>
<td>Small</td>
</tr>
<tr>
<td>Banana</td>
<td>Yellow</td>
<td>Medium</td>
</tr>
<tr>
<td>Cherry</td>
<td>Red</td>
<td>Small</td>
</tr>
<tr>
<td>Grape</td>
<td>Green</td>
<td>Small</td>
</tr>
<tr>
<td>Melon</td>
<td>Yellow</td>
<td>Large</td>
</tr>
<tr>
<td>Blackberry</td>
<td>Black</td>
<td>Small</td>
</tr>
</tbody>
</table>

This can be done in different ways:

Somebody else can check the data on the screen for you against the original paper documents

You could print out your table and check it against the original paper documents

You could type in the data twice (like you do with your password), and get the computer to check that both sets of data are identical.
Other methods of verification include control, batch or hash totals. To find out more about these, visit the mini-website on Validation and Verification.

### 2.5. Editing and Checking

As well as choosing the correct data types to try to reduce the number of errors made when entering data into the database, there is another method that can be used when setting up the table. This is called 'Validation'.

It is very important to remember that Validation cannot stop the wrong data being entered, you can still enter 'Smiht' instead of 'Smith' or 'Brown' instead of 'Green' or '78' instead of '87'.

What Validation can do, is to check that the data is sensible, reasonable and allowable.

This page will not go into any great depth about different methods of validation as there is a whole mini-website on Validation alone. Go and have a look at it to find out more details about the best kind of Validation to use and the reasons why.

Some of the types of Validation that you could set up for your database are:

<table>
<thead>
<tr>
<th>Validation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type Check</td>
<td></td>
</tr>
<tr>
<td>If the datatype number has been chosen, then only that type of data will be allowed to be entered i.e. numbers</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td>If a field is only to accept certain choices e.g. title might be restricted to 'Mr', 'Mrs', 'Miss' and 'Ms', then 'Dr' wouldn't be allowed.</td>
<td>Mr, Mrs, Miss, Ms</td>
</tr>
<tr>
<td></td>
<td>Brown, Green, Blue, Yellow, Red</td>
</tr>
</tbody>
</table>
### Range Check

A shop may only sell items between the price of £10.00 and £50.00. To stop mistakes being made, a range check can be set up to stop £500.00 being entered by accident.

A social club may not want people below the age of 18 to be able to join.

Notice the use of maths symbols:
- `>` 'greater than'
- `< 'less than'
- `=' equals

<table>
<thead>
<tr>
<th>&gt;=10 AND &lt;=50</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;=18</td>
</tr>
</tbody>
</table>

### Presence check

There might be an important piece of data that you want to make sure is always stored. For example, a school will always want to know an emergency contact number, a video rental store might always want to know a customer’s address, a wedding dress shop might always want a record of the bride’s wedding date.

A presence check makes sure that a critical field cannot be left blank, it must be filled in.

| School database: Emergency contact number |
| DVLA database: Date test passed          |
| Electoral database: Date of birth        |
| Vet’s database: Type of pet              |
Some things are always entered in the same format. Think about postcode, it always has a letter, letter, number, number, number, letter and letter e.g. CV43 9PB. There may be the odd occasion where it differs slightly e.g. a Birmingham postcode B19 8WR, but the letters and numbers are still in the same order.

A picture or format check can be set up to make sure that you can only put letters where letters should be and numbers where numbers should be.

**Postcode:** CV43 9PB

**Telephone number:** (01926) 615432

### 2.6 Summary

- A computer file is a piece of arbitrary information, or resource for storing information, that is available to a computer program and is usually based on some kind of durable storage.
- Operations on a file includes Opening a file to use its contents, reading or updating the contents, Committing updated contents to durable storage and Closing the file, thereby losing access until it is opened again.
• The main drawback to inserting, deleting or amending records in an ordered Sequential file is that the entire file must be read and then the records written to a new file.

• Direct access files allow direct access to a particular record in the file using a key and this greatly facilitates the operations of reading, deleting, updating and inserting records.

• An Indexed file may have multiple keys.

• In modern computer systems, files are typically accessed using names.

• When computer files contain information that is extremely important, a back-up process is used to protect against disasters that might destroy the files.

• A member of staff could type the information directly into a computerized data entry form whilst the customer is with them.

• It was mentioned that validation cannot make sure that data you enter is correct, it can only check that it is sensible, reasonable and allowable.

• Indexed files may have up to 255 keys, the keys can be alphanumeric and only the primary key must be unique.

2.7 Key words

• File - A file is durable in the sense that it remains available for programs to use after the current program has finished.

• COBOL supports two kinds of direct access file organizations - Relative and Indexed.

• Record - A record is all of the data or information about one person or one thing.

• Field - A record is made up of lots of individual pieces of information. Look at Wonder Woman's record; it stores her first name, last name, address, city and age.

• OMR - An Optical Mark Reader is a scanning device that reads carefully placed pencil marks on a specially designed form or document.
OCR - Optical Character Recognition (OCR) enables the computer to identify written or printed characters.

2.8 Self Assessment Questions (SAQ)

- Define the term File. Explain the different types of operations that can be performed on files with the help of suitable examples.
- Explain the architecture of file organization.
- What are different types of files? Explain insertion, modification and deletion operation in context with these files types.
- What do you mean by field, record and table? Explain with the help of suitable examples.
- Define the term Data Capturing. Explain different data capturing techniques.
- Explain what is meant by the term back – up? Why it is important to keep the back up copy away from the computer system?
- When the contents of a file are changed, a transaction log is often kept. Explain briefly the reason for the transaction log.
- Explain how the transaction file and the master file are used to produce a new updated master file?
- Validation and Verification help to reduce the errors when inputting data. Justify the statement.
- Explain the difference between validation and verification. Give the names of three validations checks that can be used.

2.9 References/Suggested Readings

- Sams Teach Yourself COBOL in 24 Hours, Hubbell, Sams, Dec 1998
- Structured COBOL Methods, Noll P, Murach, Sep 1998
- ICT for you, Stephon Doyle, Nelson Thornes, 2003
- Information and Communication Technology, Denise Walmsley, Hodder Murray 2004
- Information Technology, P Evans, BPB Publications, 2000
3.0 Objectives
At the conclusion of this lesson you should be able to know:

- Data Storage
- Storage Capacity
- Storage Devices
- Manual file System
- Types of Files
- File Recovery Procedure
- File Backup

3.1. Introduction
Unless you want to lose all of the work you have done on your computer, you must have some means of storing the information.

There are various storage devices that will do this for you. Some of the most common ones that you are likely to have come across are:

- hard disks,
- floppy disks,
- CD-ROMs
- DVDs.

3.1.1. Storage Capacity

Storage capacity is measured in bytes. One byte contains 8 bits (Binary Digits) which is the smallest unit of data that can be stored.
A bit is represented as a 1 or 0 - binary numbers.

A single byte (Binary term) equals a keyboard letter, number or symbol. If you think of all of the files that you have saved on your computer and how many characters (letters) you have written, you will need millions of bytes of storage data to keep your work safe.

We normally refer to the storage capacity of a computer in terms of Kilobytes (kB), Megabytes (MB) and Gigabytes (GB) - (or even Terabytes on very large systems!).

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>Smallest unit of data, either a 0 or 1</td>
</tr>
<tr>
<td>Byte</td>
<td>8 bits. This is the lowest 'data' level and is a series of 0s and 1s, e.g.</td>
</tr>
<tr>
<td></td>
<td>00111010 = 1 byte with each 0 or 1 equal to 1 bit.</td>
</tr>
<tr>
<td></td>
<td>Each keyboard character = 1 byte</td>
</tr>
<tr>
<td>Kilobyte (kB)</td>
<td>1000 keyboard characters = 1000 bytes or 1 KB (kilobyte).</td>
</tr>
<tr>
<td></td>
<td>In reality it is really 1024 bytes which make a kilobyte, but generally</td>
</tr>
<tr>
<td></td>
<td>people refer to 1000 bytes as a kb.</td>
</tr>
<tr>
<td>Megabyte (MB)</td>
<td>1000 kilobytes = 1 MB (1 million keyboard characters).</td>
</tr>
<tr>
<td></td>
<td>Floppy disks have a capacity of 1.44 MB</td>
</tr>
<tr>
<td></td>
<td>CD ROM disks have a capacity of 650 MB.</td>
</tr>
<tr>
<td>Gigabyte (GB)</td>
<td>1000 megabytes = 1 GB (gigabytes or 1 billion characters).</td>
</tr>
<tr>
<td></td>
<td>Single sided DVD disks can typically hold 4.7Gb of data</td>
</tr>
<tr>
<td>Terabyte (TB)</td>
<td>Equal to 1,099,000,000,000 bytes or 240</td>
</tr>
</tbody>
</table>

3.1.2. Read Only Memory (ROM)

Data stored in Read Only Memory (ROM) is not erased when the power is switched off - it is permanent. This type of memory is also called 'non volatile memory'.
A Motherboard within a PC may contain a ROM chip. This chip contains the instructions required to start up the computer. Another name for this software is the BIOS.

Whenever some data needs to be stored on a permanent basis, a ROM is the best solution. For example, many car computers will contain ROM chips that store the basic information required to run the car engine.

### 3.1.3 Random Access Memory (RAM)

In contrast to ROM, Random Access Memory is volatile memory. The data is held on a chip, but only temporarily. The data disappears when the power is switched off.

Have you ever forgotten to save your work before the computer crashed? When you log back on, your work has disappeared. This is because it was stored in RAM and was erased when the PC switched off. However, if you had saved your work from RAM to the hard disk, it would have been safe!

A part of the RAM is allocated for the 'clipboard'. This is the area that stores the information when you CUT, COPY and PASTE from within programs such as Microsoft Word and Excel.

As computer programs and operating systems have become more complex, the size of RAM has increased. Today most computers are sold with either 256MB or 512 MB of RAM.
3.1.4 HARD DISK

The hard disk drive is the storage device, rather like a filing cabinet, where all the applications software and data is kept. Data stored on a hard disk can be accessed much more quickly than data stored on a floppy disk.

A Hard disk spins around thousands of times per minute inside its metal casing, which is why it makes that whirring noise. Less than a hairs breadth above the disk, a magnetic read and write head creates the 1 and 0s on to the circular tracks beneath.

Most hard drives are installed out of the way inside the computer, however you can also purchase external drives that plug into the machine. Modern Hard drives are measured in gigabytes (GB). A typical hard disk drive may be 120 Gbytes. Some computers use two hard disks, with one hard disk automatically making a backup copy of the other - another name for this is disk mirroring.

Hard disk drives can turn up in some surprising places, for example:-

- iPods (not the Nano) have a hard drive to store the music.
- Some Game machines have them installed to allow games to be stored.
- They appear inside some "Personal Video Recorders" (PVR) to act just like a video recorder - the programs can then be burned on DVD for permanent storage if needed.

Advantages:

- Necessary to support the way your computer works
- Large storage capacity
- Stores and retrieves data much faster than a floppy disk or CD-ROM
- Stored items not lost when you switch off the computer
- Usually fixed inside the computer so don't get lost or damaged
- Cheap on a cost per megabyte compared to other storage media.

Disadvantages:
- Far slower to access data than the ROM or RAM chips because the read-write heads have to move to the correct part of the disk first.
- Hard disks can crash which stops the computer from working
- Regular crashes can damage the surface of the disk, leading to loss of data in that sector.
- The disk is fixed inside the computer and cannot easily be transferred to another computer.

The hard disk shown below has a SCSI 'interface' which is one kind of standard connection method. Other connection methods are "IDE" and "SATA" interfaces. Each kind of interface has a different type of socket so they cannot get mixed up accidentally.
3.1.5 Floppy Disk

Floppy disks are one of the oldest type of portable storage devices still in use, having been around since about 1980. They have lasted, whilst so many other ideas have disappeared because they are so handy to use. (See "Floppy History" term in the box opposite for more information).

The floppy disk drive enables you to transfer small files between computers and also to make backup copies to protect against lost work. A floppy disk is made of a flexible substance called Mylar. They have a magnetic surface which allows the recording of data. Early floppy disks were indeed ‘floppy’, but the ones we use now (3 1/2 inch) are protected by a hard plastic cover. The disk turns in the drive allowing the read/write head to access the disk.
A standard floppy disk can store up to 1.44 Mb of data which is approximately equivalent to 300 pages of A4 text. However, graphic images are often very large, so you may well find that if you have used Word Art or a large picture, your work will not fit onto a floppy disk.

All disks must be formatted before data can be written to the disk. Formatting divides the disk up into sections or sectors onto which data files are stored. Floppy disks are often sold pre-formatted.

Care should be taken when handling disks, to protect the data. The surface of the disk should not be touched and they should be kept away from extreme temperatures and strong magnetic fields such as may appear close to audio speakers - otherwise you might find all your data has been wiped!

**Advantages:**

- Portable - small and lightweight
- Can provide a valuable means of backing up data
- Inexpensive
- Useful for transferring files between computers or home and school.
- Private data can be stored securely on a floppy disk so that other users on a network cannot gain access to it.
- Security tab to stop data being written over.
- Most computers have a floppy drive (although now they appear less)
- Can be written to many times.

**Disadvantages:**

- Not very strong - easy to damage
- Data can be erased if the disk comes into contact with a magnetic field
- Quite slow to access and retrieve data.
- Can transport viruses from one machine to another
- Small storage capacity, especially if graphics need to be saved
- New computers are starting to be made without floppy drives

3.1.6 ZIP DRIVE
The Zip drive is similar to a floppy drive but can store 100 MB of data, at least 70 times more than a floppy. Some zip disks store as much as 250 MB. The Zip disk is slightly thicker than a floppy disk and needs a separate drive. Zip disks are particularly useful for backing up important data or for moving data easily from one computer to another. Data is compressed, thereby reducing the size of files that are too large to fit onto a floppy disk.

**Advantage:**
- Stores more than a floppy disk
- Portable

**Disadvantage:**
- More expensive than floppies
- Drives to read the disks are not that common

### 3.1.7 Magnetic Tape

The amount of work you do on your computer at home can easily be backed up onto floppy disks or DVD for safety. However, many organisations need to back up large volumes of data and floppy disks or DVD are not the best method for doing this. In some case, Terabytes of data may need to be stored safely at low cost.

Examples of organizations that would hold this much information:-
- Satellite imaging firms holding huge backlog of images
- Movie companies holding their digitized films in archive
- Architect, car and design firms holding thousands of CAD drawings.
- Science organizations such as CERN holding the results of past experiments
- Weather organizations.

So they tend to make their back up copies onto magnetic tape.

Magnetic tape comes in two forms:
- tape reels - these are fairly large and are usually used to back up data from mainframe computers.
- cassettes or cartridges - these are fairly small in size but able to hold enough data to back up the data held on a personal computer or a small network.

Because it takes a long time to back up onto magnetic tape, it may be done at night or over a weekend when the computer network is not so busy.

The main advantage of using magnetic tape as backing storage is that it is relatively cheap and can store large amounts of data.

### 3.2. Manual Filing System

We are all used to dealing with some sort of manual information system. In manual information system some of the data is the same on each file. This is called data duplication and is one of the main problems with manual filing system. Data duplication means that more space is taken up by the files and more work is needed to retrieve the information. The main problems arise in the following situations are

- We may need to obtain information that is held on several files.
- As the data is not shared, a change in information would cause many files to need updating.
- It is time consuming and wasteful.

To overcome these anomalies, computerized systems are used. The main advantages of computerized system are as follows:

- The information is stored only once.
- Files can be linked together.
- Access to the information is rapid and there are less chances of the data becoming lost.

In Computerized systems, we can create data files, alter the data in these files and extract the data from the files.
3.3. Types of files

There are mainly four types of files:

1. Master File

A Master file is a most important file as it is the most complete and up to date version of a file. If a master file is lost or damaged and it is the only copy, the whole system will break down.

2. Transaction file

Transaction files are used to hold temporary data which is used to update the master file. A transaction is a piece of business, hence the name given as transaction file. Transactions can occur in any order, so it is necessary to sort a transaction file into the same order as the master file before it is used to update the master file.

3. Backup or Security file

Backup copies of files are kept in case the original is damaged or lost and cannot be used. Because of the importance of the master file, backup copies of it should be taken at regular intervals in case it is stolen, lost, damaged or corrupted. If the storage capacity of your disk is not enough you should always keep backup copies of all important data.

4. Transaction Log File

Transactions are bits of business such as placing an order, updating the stock, making a payment etc. If these transactions are performed in real time the data input will over write the previous data. This make it impossible to check past data and so would make it easy for people to commit fraud. A record of transaction is kept in the form of transaction log file which shows all
the transactions made over a certain period. Using the log you can see what
the data was before the changes were made and also what the changes were
and who made it. Transaction log files therefore maintain security and can
also be used to recover to transactions lost due to hardware failures.
In practice companies will keep several generations of files. This is because
there may be a problem (eg disk crash) and the update runs may have to be
done again to re-create the current master file.

3.4. File Recovery Procedure

There is always a slight chance that data contain on a master file may be
destroyed. It could be destroyed by an inexperienced user, a power failure or
even theft. For a large company, the lost of vital data could prove disastrous.
But by creating the different generations of files it is possible to recreate the
master file if it is lost.

The three generation of files are

• Oldest Master File called grand father file
• New Master File called father file
• And the most up to date Transaction file is called the son file.

When a transaction file is used to update a master file, the process creates a
new master file.

Sometimes the old master file is referred to as the father file and the new
master file as the son file.
When the update is next run...
the son file becomes the father file
the father file becomes the grandfather file
..etc...
3.4.1 Backups

In the field of information technology, backup refers to the copying of data so that these additional copies may be restored after a data loss event. Backups are useful primarily for two purposes: to restore a computer to an operational state following a disaster (called disaster recovery) and to restore small numbers of files after they have been accidentally deleted or corrupted. Backups differ from archives in the sense that archives are the primary copy of data and backups are a secondary copy of data. Backup systems differ from fault-tolerant systems in the sense that backup systems assume that a fault will cause a data loss event and fault-tolerant systems assume a fault will not. Backups are typically that last line of defense against data loss, and consequently the least granular and the least convenient to use.

Since a backup system contains at least one copy of all data worth saving, the data storage requirements are considerable. Organizing this storage space and managing the backup process is a complicated undertaking.

Backup Media

Storage media

Regardless of the repository model that is used, the data has to be stored on some data storage medium somewhere.
3.4.1.1 Magnetic tape
Magnetic tape has long been the most commonly used medium for bulk data storage, backup, archiving, and interchange. Tape has typically had an order of magnitude better capacity/price ratio when compared to hard disk, but recently the ratios for tape and hard disk have become a lot closer. There are myriad formats, many of which are proprietary or specific to certain markets like mainframes or a particular brand of personal computers. Tape is a sequential access medium, so even though access times may be poor, the rate of continuously writing or reading data can actually be very fast. Some new tape drives are even faster than modern hard disks.

3.4.1.2 Hard disk
The capacity/price ratio of hard disk has been rapidly improving for many years. This is making it more competitive with magnetic tape as a bulk storage medium. The main advantages of hard disk storage are the high capacity and low access times.

3.4.1.3 Optical disk
A CD-R can be used as a backup device. One advantage of CDs is that they can hold 650 MiB of data on a 12 cm (4.75") reflective optical disc. (This is equivalent to 12,000 images or 200,000 pages of text.) They can also be restored on any machine with a CD-ROM drive. Another common format is DVD+R. Many optical disk formats are WORM type, which makes them useful for archival purposes since the data can't be changed.

3.4.1.4 Floppy disk
During the 1980s and early 1990s, many personal/home computer users associated backup mostly with copying floppy disks. The low data capacity of a floppy disk makes it an unpopular choice in 2006.

Solid state storage
Also known as flash memory, thumb drives, USB keys, compact flash, smart media, memory stick, Secure Digital cards, etc., these devices are relatively costly for their low capacity, but offer excellent portability and ease-of-use.
Remote backup service
As broadband internet access becomes more widespread, remote backup services are gaining in popularity. Backing up via the internet to a remote location can protect against some worse case scenarios, such as someone’s house burning down, destroying any backups along with everything else. A drawback to remote backup is the internet connection is usually substantially slower than the speed of local data storage devices, so this can be a problem for people with large amounts of data. It also has the risk of potentially losing control over personal or sensitive data.

Approaches to backing up files
Deciding what to backup at any given time is a harder process than it seems. By backing up too much redundant data, the data repository will fill up too quickly. If we don't backup enough data, critical information can get lost. The key concept is to only backup files that have changed.

3.4.2 Copying files
Just copy the files in question somewhere.

3.4.3 File System dump
Copy the file system that holds the files in question somewhere. This usually involves un-mounting the file system and running a program like dump. This is also known as a raw partition backup. This type of backup has the possibility of running faster than a backup that simply copies files. A feature of some dump software is the ability to restore specific files from the dump image.

Identification of changes
Some file systems have an archive bit for each file that says it was recently changed. Some backup software looks at the date of the file and compares it with the last backup, to determine whether the file was changed.

3.4.4 Block Level Incremental

A more sophisticated method of backing up changes to files is to only backup the blocks within the file that changed. This requires a higher level of integration between the file system and the backup software.
3.4.5 Versioning file system

A versioning file system keeps track of all changes to a file and makes those changes accessible to the user. This is a form of backup that is integrated into the computing environment.

3.4.6 Backing up on-line databases

An on-line database is constantly being updated. To make sure no data is lost in the event of hardware failure, special back-up methods are used. Transaction logging and RAID (Redundant Array of Inexpensive Disks) are two commonly used methods.

Transaction logging involves storing the details of each update in a transaction log file. A “before” and “after” image of each updated record is also saved. If any part of the database is destroyed an up-to-date copy can be recreated by a utility program using the transaction log file and the before and the after image of updated records.

RAID involves keeping several copies of a database on different disks at the same time. Whenever a record is updated the same changes are made to each copy of the database. This is so that if one disk falls the data will still be safe on the others.

3.4.7 Advice

The more important the data that are stored in the computer the greater is the need for backing up these data. A backup is only as useful as its associated restore strategy. Storing the copy near the original is unwise, since many disasters such as fire, flood and electrical surges are likely to cause damage to the backup at the same time. Automated backup should be considered, as manual backups are affected by human error.
3.4.8 Rules for Backing up

a) Never keep back-up disks near the computer.
b) If you hold a lot of data which would be very expensive to recreate then you invest in a file proof safe to protect your back-ups against thief and fire.
c) Keep at least one set of back-ups disks in a different place.

3.5 Summary

- Storage capacity is measured in bytes.
- We normally refer to the storage capacity of a computer in terms of Kilobytes (KB), Megabytes (MB) and Gigabytes (GB) - (or even Terabytes on very large systems!).
- A Hard disk spins around thousands of times per minute inside its metal casing, which is why it makes that whirring noise.
- Floppy disks are one of the oldest types of portable storage devices still in use, having been around since about 1980.
- A Master file is a most important file as it is the most complete and up to date version of a file. If a master file is lost or damaged and it is the only copy, the whole system will break down.
- A transaction is a piece of business, hence the name given as transaction file.
- When a transaction file is used to update a master file, the process creates a new master file.
- A more sophisticated method of backing up changes to files is to only backup the blocks within the file that changed.
- A versioning file system keeps track of all changes to a file and makes those changes accessible to the user.
- The amount of work you do on your computer at home can easily be backed up onto floppy disks or DVD for safety.

3.6 Key words
• Transaction File - Transaction files are used to hold temporary data which is used to update the master file.
• Back-up - In the field of information technology, backup refers to the copying of data so that these additional copies may be restored after a data loss event.
• Transaction logging - involves storing the details of each update in a transaction log file.
• RAID - involves keeping several copies of a database on different disks at the same time.
• If a master file is lost or damaged and it is the only copy, the whole system will break down.

3.7 Self Assessment Questions (SAQ)
• What do you mean by Storage Capacity? How we measure the storage capacity of a computer system?
• List down the differences between:
  o RAM and ROM
  o Mega Byte and Giga Byte
• Explain what is meant by the term storage device? Give three examples of storage devices. Also give possible advantages and disadvantage of the same.
• Explain different types of files with the help of suitable examples.
• Explain what is meant by the term File Generations? Explain with the help of suitable example.
• List down some important rules for backing up files.
• Explain the process of taking backup of an online data base.
3.8 References/Suggested Readings

- Sams Teach Yourself COBOL in 24 Hours, Hubbell, Sams, Dec 1998
- Structured COBOL Methods, Noll P, Murach, Sep 1998
- ICT for you, Stephon Doyle, Nelson Thornes, 2003
- Information and Communication Technology, Denise Walmsley, Hodder Murray 2004
- Information Technology, P Evans, BPB Publications, 2000
1.0 Objectives

1. To understand the basic behavior of the COBOL language.
2. To know the various segments of a COBOL program.
3. To be able to understand the purpose of DIVISIONS, SECTIONS and paragraphs used in a COBOL program.
4. To learn the coding styles of the COBOL program.
5. To understand the concepts of data names, COBOL words, literals and constants.

1.1 Introduction

In contrast to administrative data processing, scientific computing generally involves a lower volume and diversity of input data, small or nonexistent files, less complex processing logic but more extensive mathematical manipulation, and more limited report production needs. Because administrative data processing has characteristics different from those of scientific computing, a special programming language i.e. COBOL (Common Business Oriented Language) has been developed to fulfill the particular needs associated with such processing of data. Now, the COBOL has persisted as the most widely used language for administrative data processing.

1.2 Presentation of Contents

1.2.1 HISTORY OF COBOL
In the 1950s there was a growing need for a high-level programming language suitable for business data processing. To meet this need, the Dept. of Defense (DoD) of USA (in 1958) formed a “short-term work” group. In 1959, the short-term committee gave the idea of a new language named COBOL (CCommon Business Oriented Language).

In 1960, the board of directorate of the short-term group, known as CODASYL (Conference on DATA System Language) established a COBOL maintenance committee to keep the COBOL up-to-date. On May 5, 1961, COBOL-61 was published with some revisions. The users started writing programs in COBOL when the first COBOL compiler became available in early 1962. In 1965, the next version with some new additions was published. In August 1968 a standard version of the language was approved by the American National Standards Institute (ANSI) known as ANSI-68 COBOL or COBOL-68. COBOL-74, the next revised official standard was introduced in 1974. This version is currently implemented in almost every machine. However, in 1985 a revised standard was introduced known as COBOL-85 that is the latest version of COBOL.

COBOL is self-documenting language. One of the design goals for COBOL was to make it possible for non-programmers such as supervisors, managers and users, to read and understand COBOL code. As a result, COBOL contains such English-like structural elements as verbs, clauses, sentences, sections and divisions. As it happens, this design goal was not realized. Managers and users nowadays do not read COBOL programs. Computer programs are just too complex for most laymen to understand them, however familiar the syntactic elements. But the design goal and its effect on COBOL syntax has had one important side effect. It has made COBOL the most readable, understandable and self-documenting programming language in use today. It has also made it the most verbose.

When programs are new, both the in-program comments and the external documentation accurately reflect the program code. But over the time, as
more and more revisions are applied to the code, it gets out of the step with the documentation. Ultimately, the documentation actually becomes a hindrance to maintenance rather than help. The self-documenting nature of COBOL means that this problem is not as severe with COBOL programs as it is with other languages.

Readers who are familiar with C or C++ or Java might want to consider how difficult it becomes to maintain programs written in these languages. C programs that you have written yourself are difficult enough to understand when you come back to them six months later. Consider how much more difficult it would be to understand a program that had been written fifteen years previously by someone else, and which had since been amended and added to by so many others that the documentation no longer accurately reflects the program code. This is a nightmare still awaiting maintenance programmers of the future.

COBOL is a simple language (no pointers, no user defined functions, no user defined types) with a limited scope of function. It encourages a simple straightforward programming style. Curiously enough though, despite its limitations, COBOL has proven itself to be well suited to its targeted problem domain (business computing). Most COBOL programs operate in a domain where the program complexity lies in the business rules that have to be encoded rather than in the sophistication of the data structures or algorithms required. And in cases where sophisticated algorithms are required COBOL usually meets the need with an appropriate verb such as the SORT and the SEARCH.

### 1.2.2 Advantages of COBOL

1) Its main advantage is advancement of communication i.e. it reduces the communication gap between the programmers and decision makers.

2) No need of any symbolic and machine instructions by the programmers.
3) Pre-tested modules of input and outputs are included in the COBOL processor. Hence it reduces the tedious job of writing and test them.

4) The programmer is writing in a language that is familiar to him/her and hence reduces the documentation.

5) While COBOL is not completely portable but with a little modification in a program you can make a COBOL program portable.

6) A COBOL program is a set of different DIVISIONS therefore different divisions can handle using the modular programming approach.

7) During the completion phase, a COBOL processor generates a list of diagnostics (list of errors other than logical)

1.2.3 Structure of a COBOL Program

A COBOL program is made up of the hierarchy shown in Fig 4.1.

![Hierarchy of COBOL program](image)

Fig 4.1 Hierarchy of COBOL program

1.2.3.1 Divisions

A division is a block of code, usually containing one or more sections or paragraphs. Division starts from the point where the division name is encountered and ends with the beginning of the next division or with the end of the program text. A division name is followed by the word DIVISION and a period.
There are four divisions in a COBOL program – identification division, environment division, data division and procedure division. These divisions can appear in the program in this order only.

1.2.3.2 Sections

A section is a block of code usually containing one or more paragraphs. A section begins with the section name and ends where the next section name is encountered or where the program text ends.

Section names are devised by the programmer, or defined by the language. A section name is followed by the word SECTION and a period.

1.2.3.3 Paragraphs

A paragraph is a block of code made up of one or more sentences. A paragraph begins with the paragraph name and ends with the next paragraph or section name or the end of the program text. A paragraph name is devised by the programmer or defined by the language, and is followed by a period.

1.2.3.4 Sentences and statements

A sentence consists of one or more statements and is terminated by a period. Following are few examples of valid sentences:

MOVE .21 TO VatRate; MOVE 1235.76 TO ProductCost.

COMPUTE VatAmount = ProductCost * VatRate.

A statement consists of a COBOL verb and an operand or operands. For example:
SUBTRACT Tax FROM GrossPay GIVING NetPay

The statement of a COBOL program must follow the hierarchy of units.

Character: It is the lowest and indivisible unit of the COBOL program structure.

Word: It is formed with the string of characters.

Clause: It consists of either characters or words to specify the attributes w.r.t. an entry.

1.2.4 COBOL PROGRAM

At the highest level a COBOL program consists of the following four divisions:

a) IDENTIFICATION DIVISION
b) ENVIRONMENT DEVISION
c) DATA DEVISION
d) PROCEDURE DEVISION
1.2.4.1 IDENTIFICATION DIVISION

The purpose of IDENTIFICATION DIVISION is to provide the program and programmer related information to the outer world. It contains a number of paragraphs with the name of the program, author’s name, date on which program was written or compiled and some more program related information. The following program code gives the identification division and its paragraphs. PROGRAM-ID paragraph is compulsory and remaining paragraphs are optional. All the paragraphs are self explanatory.

IDENTIFICATION DIVISION.
PROGRAM-ID. SalaryBill.
AUTHOR. Nath R.
INSTALLATION. Dept of Computer Science.
SECURITY. Departmental Level.

Program-id is the name of the program for the identification of the program. This name must start with alphabetic character with the restricted size (depending on the compiler’s limit)
1.2.5.2 ENVIRONMENT DIVISION

does the second division of the COBOL program. It identifies the environment of the program. The portability of the COBOL program can be obtained by the modification of the ENVIRONMENT DIVISION because the device specifications are given in this division i.e. If you shift from one type of system to another then you must update this division as per the new system specifications. The CONFIGURATION SECTION and INPUT-OUTPUT SECTION are two sections of this division. The CONFIGURATION deals with the system specifications and the INPUT-OUTPUT refers to input/output devices used in the program.

The following program segment shows the two sections and their paragraphs:

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM PC.
OBJECT-COMPUTER. IBM PC.
SPECIAL-NAMES. CONSOLE IS CRT.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT IN-FILE ASSIGN TO DISK.
   SELECT REPORT-FILE ASSIGN TO PRINTER.
IO-CONTROL.

SOURCE-COMPUTER paragraph identifies the name of the computer where source program is developed and compiled. OBJECT-COMPUTER paragraph identifies the name of the computer where program is executed. SPECIAL-NAMES paragraph can be used to relate some hardware names to user-defined names. For example, DECIMAL POINT IS COMA, CONSOLE IS CRT etc.
FILE-CONTROL paragraph is used to select input/output files and assign them to hardware devices. IO-CONTROL paragraph is used for advanced I/O system and will be discussed later.

1.2.5.3 DATA DIVISION

The DATA DIVISION refers to all the data field names with their data type, size etc. those are used in the PROCEDURE DIVISION. DATA DIVISION is an entry rather than a statement because it is only a declaration and not an instruction for the compiler. In this division you have mainly two sections: - FILE SECTION, refer to the input-output data and WORKING-STORAGE SECTION, to hold the intermediate results. Here both the sections are optional but if used then the FILE SECTION must be the first one.

```
DATA DIVISION.
FILE SECTION.
[File description, Record description ]
WORKING-STORAGE SECTION.
[Data item or record description ]
```

Note that DATA DIVISION contains sections only. It does not contain any paragraph. You will learn the usage of these sections in the ensuing chapters.

1.2.5.4 PROCEDURE DIVISION

The PROCEDURE DIVISION refers to the instructions given by the programmer to communicate the logic of the program and to handle the various elements those are defined in the DATA DIVISION. DATA DIVISION is consisting with the SECTIONS, those are consisting with sentences terminated by a period(.). Each sentence is composed from statements (valid instructions starting with COBOL verb) . Here you can concatenate more than one statements in one line but in that case those must be separated by the comma(,) and terminated this group by the period.
1.3 CHARACTER SET AND WORDS

Character is a basic unit that can’t be subdivided further into parts or you can say that characters are the alphabets of the COBOL language. The COBOL character set is shown in Table 4.1. Lowe case alphabets are converted into the uppercase by the COBOL compiler. That means COBOL is not case-sensitive language. That is why lowercase letters are not listed in the Table 4.1. COBOL programs are designed using these characters.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>CHARACTER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>0-9</td>
<td>Numerals/Digits</td>
</tr>
<tr>
<td>2.</td>
<td>A – Z</td>
<td>Alphabets</td>
</tr>
<tr>
<td>3.</td>
<td>” ” ; $</td>
<td>Blank or Space</td>
</tr>
<tr>
<td>4.</td>
<td>&lt; &gt; ( ) .</td>
<td>Special Character</td>
</tr>
</tbody>
</table>

Table 4.1: Character Set in COBOL

A group of characters form a word, which can be further categorized as user-defined words (defined by the user itself) and reserved words (defined by the language itself). A user cannot use a reserved word as user-defined word.

To coin the user-defined words, the following rules must be following:

a) Only 0-9 A – Z and -(hyphen) can be used to form a user defined word.

b) The maximum word length can be 30 characters. (This restriction is compiler dependent).

c) The first letter should be an alphabet, remaining can be alphanumeric or hyphen.

d) There must be at least one alphabet in the word.
e) The hyphen (if used) must be sandwiched between alphanumeric characters.

f) Only hyphen is allowed as a special symbol, no other special symbol is allowed.

Some valid examples: ROLL-NO STUDENT-ID DATE-23
Some Invalid examples -ROLLNO (Hyphen can not be first character), STUDENT/ID (no other special symbol is allowed), DATE 23 (blank is not allowed)

1.3.1 DATA NAMES

In COBOL, memory locations are directly accessed through the data names by the programmer whenever they want to access them. The memory locations are referenced by their respective data-names. The data names must be a user-defined word and must not be a reserved word of COBOL.

1.3.2 LITERALS /CONSTANTS

Literals are the actual values of the data in an operation. A data name can have different values of it at different execution points in a program but the value of a literal remains constant throughout the program execution. Therefore, these are also known as constants. Literals are self defined; they don’t require any data-name to define them and hence are not defined in the DATA DEVISION of the COBOL program.

There are three types of literals in COBOL as shown in Fig 4.2:
1.3.2.1 Numeric Literals

Numeric literals are consist of numerals and sign (plus or minus). It can be a whole number (i.e. integer) or a fractional number. If it is a fractional number then the decimal point should not be at the right most position of the literal. The sign (if present) must be at the leftmost position without any blank between the first digit and the sign of the literal. The size of the literal is compiler-dependent.

Some examples of the valid numeric literals:

23.7 .1973 -25.2 2007

1.3.2.2 Nonnumeric Literal

The basic use of the nonnumeric literals is for the messages and headings in the program to increase the readability. The nonnumeric literals are string of characters enclosed within a pair of quotation marks. There is only one restriction that, if a quotation mark is include in a nonnumeric literal, then it must be followed by another quotation mark with in the pair of quotation marks. The size of the non-numeric literal is again compiler dependent.

Some examples of the valid nonnumeric literals:
1.3.2.3 Figurative Constant

The frequently used constant values can be treated as figurative constants. These are referred by some well-defined fixed names. When the compiler encounters these names (figurative constants), it sets a predefined value(s) for these names in the object program.

Following are the figurative constants provided by the COBOL:

ZERO, ZEROS, ZEROES: These specify value 0.
SPACE, SPACES: These specify one or more blanks.
QUOTE, QUOTES: These specify single character “.”
HIGH-VALUE, HIGH-VALUES: These specify the highest value in the collating sequence.
LOW-VALUE, LOW-VALUES: These specify the lowest value in the collating sequence.

1.4 Coding Rules for COBOL program

COBOL program must be encoded in a format required by its compiler. Formally COBOL programs are written on a coding sheet specifically meant for this purpose. There are 80 character positions in one line on the coding sheet. One line is divided into the following field positions:

<table>
<thead>
<tr>
<th>Positions</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-6</td>
<td>Sequence</td>
</tr>
<tr>
<td>7</td>
<td>Indicator</td>
</tr>
<tr>
<td>8-11</td>
<td>Margin A or Area A</td>
</tr>
<tr>
<td>12-72</td>
<td>Margin B or Area B</td>
</tr>
<tr>
<td>73-80</td>
<td>Identification</td>
</tr>
</tbody>
</table>
**Sequence Field**: Each coding lines can optionally be assigned a sequence number. Sequence number must be in the ascending order. Positions 1-3 can be used for page numbers and positions 4-6 can be used for line numbers.

**Indicator Field**: This field can be used in the following ways:

* or / in this field indicates comment line. Comments lines are ignored by the compiler. They appear in the listing of the program only. / also starts the printing of the list from the new page.

- in this field indicates the continuation of the non-numeric literal.

**Margin A**: COBOL requires some entries be started from margin A. Division names, section names, paragraph names, FD, level no 01 are started from margin A and remaining entries are started from **Margin B**. All the entries of the COBOL program are written in the area 12-72.

**Identification Field**: This area is ignored by the COBOL compiler and can be used for the purpose of identifying lines in the program. In this line any thing can be written for the comment purpose.

**1.5 Notations used**

To describe the COBOL language following notations have been used:

1. The words consisting with upper case letters forms key words.
2. The operands are in lower case letter word.
3. The part of the statement with in the square brackets [ ] is optional at the user end.
4. Consider at least one of the statements with in the curly brackets {}.
5. The comma (,) and semi comma (;) are optional at the user end.
6. The blank or space is used as the separator between the two statements.
1.6 Summary

- The COBOL language is English like language and used throughout the world for programming the business data processing applications. The reason for its popularity is continuously standardization and improvements by the committees.

- COBOL has many self-documenting features so it is easily understood by nonprogrammers also. As a result, the additional documentation required by the COBOL program is very less.

- COBOL is high-level language and uses pre-tested input/output modules. It can be easily ported to other systems.

- COBOL programs can easily implement the modular approach of the system design. Debugging is very simple with the help of diagnostic messages.

- Every COBOL program contains four divisions in the following order: IDENTIFICATION DIVISION, ENVIRONMENT DEVISION, DATA DEVISION, PROCEDURE DEVISION. These divisions further contains sections and/or paragraphs.

- Sentence consists of one or more statements terminated by a period. One sentence can be encoded into more than one line or more than one sentences can be encoded into one line.

- COBOL words are of two types - user-defined words and reserved words. Reserved words can not be used as user-defined words as they have some pre-defined meaning to the compiler.
- COBOL has three types of literals – numeric, non-numeric and figurative.

1.7 Key Words

division, section, paragraph, sentence, word, reserved, key, margin

1.8 Self Assessment Questions (SAQ)

1. When did the first CODASYL committee meet and what were their major objectives?
2. Give a brief history of COBOL Language?
3. List the advantages of COBOL over other programming languages.
4. List the divisions of COBOL program and discuss the purpose of each division?
5. Discuss the basic structure of the COBOL program with proper.
6. Discuss the coding rules for the COBOL program.
7. List the paragraphs used in the identification division and discuss the purpose of each paragraph.
8. List the sections and their paragraphs used in the environment division and discuss the purpose of each paragraph.
9. List the sections used in the data division and discuss the purpose of each section.

1.9 References/Suggested Readings

1. COBOL Programming by M.K.Roy and D..Dastidar ; TMH
2. Schaum’s outline series Programming with Structured COBOL ; MGH
4. Comprehensive COBOL, vol-II , Advanced COBOL programming, 4/e by A.S.Philippakis and Leonard J. Kazmier ; TMH
5. Structured COBOL: Fundamentals and style, 4/e by Welburn; TMH
6. Computer Programming in COBOL by V.Rajaraman; PHI
7. Fundamentals of Structured COBOL Programming by Carl Feingold; Galgotia Booksourc.

1.0 Objectives

- To introduce you about the COBOL verbs.
- To discuss the Input/Output verbs such as ACCEPT, DISPLAY, OPEN, CLOSE, READ, WRITE.
- To explain compiler-directed COBOL verbs like ENTER, COPY, USE
- To discuss sequence control verbs like IF, GO TO, PERFORM, STOP

1.1 Introduction

COBOL verbs are building blocks of the PROCEDURE DIVISION in a COBOL program. On the basis of the operations the COBOL verbs do, they can be categorized into the following groups as shown in the table 5.1.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>CATEGORY</th>
<th>VERBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Input/Output</td>
<td>ACCEPT, DISPLAY, OPEN, CLOSE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ, WRITE</td>
</tr>
<tr>
<td>2.</td>
<td>Compiler- Directing</td>
<td>ENTER, COPY, USE</td>
</tr>
<tr>
<td>3.</td>
<td>Sequence Control</td>
<td>IF, GO TO, PERFORM, STOP</td>
</tr>
<tr>
<td>4.</td>
<td>Arithmetic</td>
<td>ADD, SUBTRACT, COMPUTE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXPONENT, DIVIDE</td>
</tr>
<tr>
<td>5.</td>
<td>Data Movement</td>
<td>MOVE</td>
</tr>
<tr>
<td>6.</td>
<td>String/Character</td>
<td>EXAMINE, INSPECT, STRING,</td>
</tr>
<tr>
<td></td>
<td>Manipulation</td>
<td>UNSTRING</td>
</tr>
</tbody>
</table>
Table 5.1 CBOL Verbs

In this chapter first three categories of the verbs will be discussed. Remaining verbs will be discussed in the next chapter.

1.2 Presentation of contents

1.2.1 Input/Output verbs

COBOL language provides number of verbs that can allow you to perform input/output operations with the various I/O devices. I/O operations can be with the files stored on secondary storage or through the keyboard or display unit. ACCEPT verb allows you to input your data through keyboard while DISPLAY verb can be sued to display your output on the screen (Visual Display Unit). OPEN, CLOSE, READ and WRITE verbs are associated with files handling. The following paragraphs discuss these verbs in detail.

1.2.1.1 ACCEPT verb

ACCEPT verb is used to supply a small-size data like date, time or control totals etc. to the specified data item. It is not used for the high-volume of the data like reading from files. There are two syntax for the ACCEPT statement, which are given below:

Syntax-1

    ACCEPT identifier [FROM mnemonic-name].

In this syntax, when the FROM option is omitted, then the data is read into the identifier through the user’s console. If some mnemonic name is assigned to the input device then that name is specified to read data from that device into the identifier.

Syntax-2
According to this format you can read the system's date and time into the identifier. The DATE option stores the six-digit (YYMMDD) current date of the system into the identifier. DAY option returns the five-digit (YYDDD) date into the identifier. Value of DDD can be any from 001 to 365 i.e. day of the year. The TIME option stores the eight-digit (HHMMSSTT) current time of the system into the identifier.

For example

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 STUDENT-RECORD.
   05 ROLL-NO PIC X(5).
   05 NAME PIC X(15).
   05 DOB PIC 999999.
.
.
.
PROCEDURE DIVISION.
INPUT-PARA.
   ACCEPT ROLL-NO FROM CONSOLE.
   ACCEPT NAME.
   ACCEPT DOB FROM DATE.
```

First ACCEPT statement takes Roll NO from console of the computer. Second ACCEPT statement takes NAME from the console of the computer as by default the FROM option is CONSOLE. Third ACCEPT statement takes DOB from the system DATE.

1.2.1.2 DISPLAY Verb
It is used to deal with small size (low volume) of the data like messages, control totals as output data on peripherals (printers, console, etc.). The output of a DISPLAY verb is without any blank in between two data values, if it is required we can use figurative constant SPACE or a blank is included in a nonnumeric literal.

Syntax: of the DISPLAY statement is given below:

\[
\text{DISPLAY } \{ \text{identifier-1 literal-1} \}[ \text{identifier-2 literal-2} ] \cdots [ \text{UPON mnemonic-} \]
\]

Syntactical Rules:

(i) Here you can use numeric or nonnumeric (must be unsigned) identifiers or literals.
(ii) In case of more then one operand with DISPLAY then the size of the sending data is the algebraic sum of all the operands.
(iii) The order of the data items at the hardware device is identical to their order in the DISPLAY verb.
(iv) The identifier-n may be either an elementary or group item.
(v) The figurative constant ALL is not allowed.
(vi) In the absence of UPON option, standard display device is used by default.

For example

```
DATA DIVISION.
WORKING-STORAGE SECTION.
02 STUDENT-RECORD.
   05 ROLL-NO PIC X(5).
   05 NAME PIC X(15).
   05 DOB PIC 999999.
```
PROCEDURE DIVISION.
INPUT-PARA.
   ACCEPT ROLL-NO FROM CONSOLE.
   ACCEPT NAME.
   ACCEPT DOB FROM DATE.
   DISPLAY ROLL-NO, SPACE, NAME, SPACE.

The DISPLAY and ACCEPT both are used for the proper handling of a COBOL program by its operator. Through these verbs a COBOL programmer can communicate with the operator of the programmer that at what places he/she must enter the data from console for the proper functioning of the program.

1.2.1.3 OPEN verb

When ever a file is operated with either READ or WRITE operations in COBOL program, then firstly it must be opened with the help of the OPEN verb. The OPEN verb describes that either the file is opened as input file or output file. If a file is opened with input file then only reading is possible. On the other hand if the file is opened as an output file then only writing is possible. After use, a file must be closed with the CLOSE verb. If the file has been closed during the processing, another OPEN statement must be executed prior to any other use. Each file that has been opened must be defined in the file description entry in the Data Division as well as the SELECT entry in the Environment Division.

Syntax:

OPEN { INPUT file-name-1 [, file-name-2]…
OUTPUT file-name-3 [, file-name-4]… }…
With one OPEN statement, more than one file can be opened in input or output mode.

Example: PROGRAM 5.1

ENVIRONMENT DIVISION.
....
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT IN-FILE ASSIGN TO DISK.
  SELECT OUT-FILE ASSIGN TO DISK.
DATA DIVISION.
FILE SECTION.
FD  IN-FILE
  LABEL RECORD IS STANDARD.
  01  IN-RECORD.
      05  ROLL-NO   PIC  X(5).
      05  NAME      PIC  X(15).
      05  CLASS     PIC  X(10).
      05  MARKS-OBT PIC 9999.
      05  TOTAL-MARKS PIC 9999.
....
WORKING-STORAGE SECTION.
  77  TEMP          PIC  99V99.
FD  OUT-FILE
  LABEL RECORD IS STANDARD.
  02  OUT-RECORD.
      05  FILLER     PIC XX.
      05  O-ROLL-NO  PIC  X(5).
      05  FILLER     PIC XX.
      05  O-NAME     PIC  X(15).
      05  FILLER     PIC XX.
      05  O-CLASS    PIC  X(10).
      05  FILLER     PIC XX.
PROCEDURE DIVISION.
OPEN-PARA.
  OPEN  INPUT  INFILE,
  OUTPUT OUTFILE.
READ-PARA.
  READ IN-RECORD
  AT END GOTO LAST-PARA.
PROCESS-PARA.
  COMPUTE TEMP = (MARKS-OBT / 1250) * 100.
  MOVE ROLL-NO TO O-ROLL-NO.
  MOVE NAME TO O-NAME.
  MOVE CLASS TO O-CLASS.
  MOVE MARKS-OBT TO O-MARKS-OBT.
  MOVE TEMP TO O-PERCENTAGE.
WRITE-PARA.
  WRITE OUT-RECORD.
  GO TO READ-PARA.
LAST-PARA.
  CLOSE IN-FILE, OUTFILE.
  STOP RUN.

1.2.1.4 CLOSE Verb

A CLOSE verb is used to close an open file in a COBOL program. Every file should be closed before the termination of the program. When a close statement is executed, the IOCS starts end of the file process. There must be
a CLOSE statement for every OPEN i.e. for both INPUT as well as OUTPUT file.

Syntax: of the CLOSE statement

```
CLOSE file-name [WITH LOCK] [ , file-name-2 [WITH LOCK] ]...
```

The files file-name1, file-name2,.., file-name-n must be defined in the FD entry of the Data Division. The option WITH LOCK restricts the opening of the same file within the same program. Always develop a habit to use the last CLOSE statement of a file having WITH LOCK option; otherwise it can be possible that after termination of the program the desired file can be lost from the disk. The CLOSE verb is illustrated in the Program 5.2.

### 1.2.1.5 READ Verb

The READ verb is used to make available the next logical record for processing from an input file. A READ statement must be executed before the data from a record can be processed. When a read operation for all the records of a file is complete i.e. after the “end-of-file”, the statement followed by the AT END clause will be executed. Hence a READ verb performs two operations, one it makes the data available for processing and secondly it also determines what to do when the end-of-file comes.

Syntax: of the READ verb:

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

If the INTO option is used, then the input record is moved to the identifier-1. When logical end of the file is reached, then the statement after AT END is executed. Statement after the AT END can be any imperative statement only. Use of the READ statement is illustrated in the Program 5.1.
Note: “AT END” clause must be included in the READ statement in case of sequential input file.

1.2.1.6 WRITE Verb

The WRITE verb is used to release a logical record for insertion into an output file. Some time it is also used for the vertical positioning of lines with in a logical page (similar to indent in word).

Syntax: of the WRITE verb.

```
WRITE     record-name   [FROM identifier-1]
          {BEFORE    ADVANCING { {Identifier-1
          AFTER   Integer-1
          } Line    Identifier-2}
          } Line
          } Lines
```

In case of WRITE verb record-name is required in place of file-name. When FROM option is used, then first identifier-1 is moved into the output record and then output record is written into the output file. ADVANCING option is used to control the vertical spacing between the records. Integer-1 or identifier-2 number of lines can be inserted before writing a record or after writing a record into the output file. Use of the WRITE statement is illustrated in the Program 5.1.

1.2.2 Compiler-Directing verbs

There are three compiler-directing statements in the COBOL language – ENTER, USE and COPY. These statements are used to direct the compiler and no object code is generated for these statements.
1.2.2.1 ENTER Verb

ENTER verb is used to support more than one languages in a COBOL program. In this case, the statements of the other language are executed in the object program as if they had been compiled in the object program with the ENTER verb. A programmer can refer any programming language name, which is specified by the implementer.

Syntax: of the ENTER verb:

```
ENTER     language-name [ routine-name ].
```

If the statement is not a single line statement then it must be included through a routine.

1.2.2.2 USE verb

The USE verb is behaved as an indirect verb i.e. USE verb itself is never executed. If some input-output errors (exceptions) occur, then the procedure followed by the USE statement is executed. The procedure can be for error handling or for the items monitored by the associated Debugging Section.

```
USE AFTER  STANDARD        
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
                          
```

File-name-1     [ , file-name-2]
INPUT
OUTPUT
I-O
EXTEND

Note: - After execution of the procedure referred by the USE verb the control returns to the invoking routine. When INPUT, OUTPUT, I-O or EXTEND option is used then the procedures referred by the USE are executed in response to any error or exception in any file opened in the declare mode.
1.2.2.3 COPY Verb

COBOL programmer’s library is a collection of COBOL source program elements accessible by reference to text-names. A text-name is a name of a member of a portioned data set contained in the programmer’s library. A well-organized library reduces the efforts to write routines common to a number of programs. The COPY verb is used to insert library data into the source program and treat it as a part of the source program by the COBOL compiler.

Syntax of the COPY verb:

```
COPY text-name [ { OF IN } library-name ]
```

```
[ REPLACING [ { ==pseudo-text-1== identifier-1 literal-1 word-1 } BY { ==pseudo-text-2== identifier-2 literal-2 word-2 } ] ] ...
```

Rules for COPY:

1. The text-name in reference to a programmer’s library must be unique in nature.
2. The COPY statement must be terminated by a period and preceded by a space.
3. If there is more than one library then the text-name must be qualified by the name of the respective library.
4. The COBOL compiler compile a program with COPY statement is similar to a program without COPY statement.
5. The comments from the library text are copied in to the source program without any change.
6. The text-name and the library names are the user defined names having at least one character in it.
(7) There is a restriction on pseudo-text-1, that it should not be either empty or consisting with only comments. On the other hand there should not be such restriction on the pseudo-text-2.

(8) The word-1 can be any valid COBOL word.

1.2.3 Sequence Control verbs

The verbs that control the execution sequence of the program are called as sequence control verbs. COBOL provides four sequence control verbs: IF, GO TO, PERFORM and STOP, which are discussed in the following paragraphs.

1.2.3.1 IF verb

This is a conditional sequence control statement. The syntax of IF is as shown below:

IF condition;
Statement1/NEXT SENTENCE
[ELSE Statement1/NEXT SENTENCE].

If condition is true then statment1 is executed. When condition is false the else part of the statement is executed. NEXT SENTENCE simply moves the control to the sentence next to the IF statement.

For example:

IF BALANCE IS LESS THAN MIN-BALANCE GO TO ERROR-PARA.

IF A IS GREATER THAN B MOVE A TO BIG
ELSE MOVE B TO BIG.

IF statement can be nested. That is IF within IF statement.
IF A IS GREATER THAN B
    IF A IS GREATER THAN C
        MOVE A TO BIG
    ELSE
        MOVE C TO BIG
    ELSE
        MOVE C TO BIG

1.2.3.1 GO TO verb

A GO TO verb is used for the control to be branched with or without any condition to the first statement of a predefined procedure-name. The execution is continued from the first statement of that procedure-name. The name of the procedure is given in the header entry of the procedure through which it is refer in the GO To statement. Therefore a programmer must take extra care while using this statement.

Some time a GO TO statement is a better solution for a problem as compared to other alternatives.

Syntax of the GO TO statement:

    GO TO procedure-name.

Rules for GO TO:

1) Always use the GO TO, to transfer the control in a COBOL program under the boundaries of a module.
2) Always use the GO TO statement is use to transfer the control only in the forward direction with in a module.
3) Always use the GO TO as an exit point of a paragraph of a sequence of paragraphs.

Example:

    PROCEDURE DIVISION.
1.2.3.2 PERFORM Verb

The PERFORM verb is used to specify the sequence of execution of a COBOL modular program known as range of the PERFORM statement. When ever a PERFORM statement is reached in a COBOL program, then a temporary departure from the normal sequential execution takes place. In COBOL, PERFORM is most flexible verb that is it has a number of uses in a COBOL program. PERFORM is used to control the execution of the loops.

PERFORM verb has many forms. Syntax of each form is described in the following paragraphs.

Simple PERFORM

Syntax-1 of PERFORM

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

Here the procedure-name is either a paragraph or a COBOL section name. It is important to note that a procedure-name must not contain any GO TO or a STOP RUN statement, however a procedure may itself contain another PERFORM instruction.

Rules for PERFORM statement:
1. The sequence of procedure-names in a PERFORM statement must be same as you desired at the time of execution.
2. Procedure-name1 through Procedure-name2 will contain all procedures between these two limits inclusive.
3. A procedure-name can be a paragraph or a section name.

The simplest form of the PERFORM is responsible for the single execution of the procedure, referred by the PERFORM.

Example: Program 5.2

DATA DIVISION.

WORKING-STORAGE SECTION.
77 COUNT PIC 9999 VALUE ZERO.
PROCEDURE DIVISION.
PARA1.
    PERFORM ADD-ONE-PARA.
    DISPLAY COUNT.
    STOP RUN.
ADD-ONE-PARA.
    ADD 1 TO COUNT.

This program displays 1.

Example: Program 5.3.

DATA DIVISION.

WORKING-STORAGE SECTION.
77 COUNT PIC 9999 VALUE ZERO.
PROCEDURE DIVISION.
PARA1.
    PERFORM ADD-ONE-PARA THRU THEN-ADD-FIVE.
    DISPLAY COUNT.
STOP RUN.
ADD-ONE-PARA.
   ADD 1 TO COUNT.
THEN-ADD-FIVE.
   ADD 5 TO COUNT

This program displays 6.

**PERFORM is with TIME option:**

Syntax2:

```
PERFORM procedure-name-1 [{THRU THROUGH} Procedure-name-2 ] identifier/literal TIMES
```

In this case the range of procedures from procedure-name1 thru procedure-name-2 will be executed literal or identifier times.

Example: Program 5.4

DATA DIVISION.

WORKING-STORAGE SECTION.
77 COUNT PIC 9999 VALUE ZERO.
PROCEDURE DIVISION.
PARA1.
   PERFORM ADD-ONE-PARA 10 TIMES.
   DISPLAY COUNT.
STOP RUN.
ADD-ONE-PARA.
   ADD 1 TO COUNT.
This program displays 10.

Example: Program 5.5.
DATA DIVISION.

WORKING-STORAGE SECTION.
77 COUNT PIC 9999 VALUE ZERO.
PROCEDURE DIVISION.
PARA1.

    PERFORM ADD-ONE-PARA THRU THEN-ADD-FIVE 5 TIMES.
    DISPLAY COUNT.
    STOP RUN.
ADD-ONE-PARA.
    ADD 1 TO COUNT.
THEN-ADD-FIVE.
    ADD 5 TO COUNT

This program displays 30.

PERFORM is with UNTIL option:

Syntax: 3

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

The PERFORM with the UNTIL option is the COBOL implementation of the DO-WHILE structure. As we know that a DO-WHILE structure terminates on a false statement, where as COBOL UNTIL terminates on a true statement. Therefore, the test condition must base upon the inverse of the desired logic.
**Fig 5.1:** Flow-chart of PERFORM with UNTIL

Notes:-
1. Here the condition-1 can be simple or a compound predicate (logical expression).
2. The decision statement must execute before the specified procedure.
3. Procedure is executed till the condition remains false.

Example: Program 5.6

DATA DIVISION.

WORKING-STORAGE SECTION.

77 COUNT PIC 9999 VALUE ZERO.
77 INDX PIC 99 VALUE 1.

PROCEDURE DIVISION.

PARA1.

PERFORM ADD-ONE-PARA UNTIL INDX > 10.
DISPLAY COUNT.
STOP RUN.
ADD-ONE-PARA.
   ADD 1 TO COUNT.
   ADD 1 TO INDX.

This program displays 10.

Example: Program 5.7.

DATA DIVISION.
...
WORKING-STORAGE SECTION.
77 COUNT PIC 9999 VALUE ZERO.
77 INDX PIC 99 VALUE 1.
PROCEDURE DIVISION.
PARA1.
   PERFORM ADD-ONE-PARA THRU THEN-ADD-FIVE
      UNTIL INDX > 5.
   DISPLAY COUNT.
   STOP RUN.
ADD-ONE-PARA.
   ADD 2 TO COUNT.
THEN-ADD-FIVE.
   ADD 3 TO COUNT.
   ADD 1 TO INDX.

PERFORM with VARYING ... AFTER option:

This form of PERFORM acts like nested loops in other programming languages.

Syntax:
PERFORM procedure-name-1 [THRU procedure-name-2]
VARYING identifier1/index1 FROM identifier2/index2/literal1
BY identifier3/literal2
UNTIL condition-1
AFTER identifier3/index3 FROM identifier4/index4/literal3
BY identifier5/literal4
UNTIL condition-2

The syntax of this form is explained through the following example.

Example: Program 5.6

DATA DIVISION.

WORKING-STORAGE SECTION.
77 COUNT PIC 9999 VALUE ZERO.
77 INDX1 PIC 99.
77 INDX2 PIC 99.

PROCEDURE DIVISION.
PARA1.
   PERFORM ADD-ONE-PARA
   VARYING INDX1 FROM 1 BY 1
   UNTIL INDX > 10
   AFTER INDX2 FROM 1 BY 1
   UNTIL INDX2 > 10.
   DISPLAY COUNT.
   STOP RUN.

ADD-ONE-PARA.
   ADD 1 TO COUNT.

This program displays 100.
Note: for each value of INDX1, the INDX2 will vary 10 times. Therefore, ADD-ONE-PARA will be executed 100 times.

1.2.3.3 STOP verb
STOP is another important verb of the COBOL language, which plays different role during the execution of a COBOL program. STOP statement marks the logical end of the program.

- STOP RUN is used to shift the control back to the operating system. Be ensure that all the files must be closed before using the STOP verb in a COBOL program, otherwise program can give some unexpected results during its execution. There must be at least one (in some versions of COBOL, exactly one stop statement) STOP statement in a COBOL program.

  Syntax: STOP RUN.

- STOP literal, option of STOP is used to display the value of literal to the COBOL operator’s monitor and terminate the processing of the program temporarily, so that the operator can interact with the peripheral devices for their proper functioning. In this case the program termination is released back by the operator’s signal via console terminal.

  For example:

  STOP "PLEASE SET THE PRINTER FOR THE INVOICE PRINT"

1.3 Summary

- COBOL supports different types of verbs like, Input/Output, Compiler-Directed, Sequence Control, Arithmetic and Data Manipulation.
- When ever a file is operated with either READ or WRITE operations in COBOL program, then firstly it must be opened with the help of the OPEN verb. Each file that has been opened must be defined in the
file description entry in the Data Division as well as the SELECT entry in the Environment Division.

- A CLOSE verb is used to close the opened file in a COBOL program, before termination of the program.

- The READ verb is used to make available the next logical record for processing from an input file. A READ statement must be executed before the data from a record can be processed.

- The WRITE verb is used to release a logical record for insertion in an output file. Some time it is also used for the vertical positioning of lines with in a logical page.

- The ACCEPT verb is used to supply a small-size data like date, time or control totals etc. to the specified data item.

- The output of a DISPLAY verb is without any blank in between two data values, if it is required we can use figurative constant SPACE or a blank is included in a nonnumeric literal. The DISPLAY and ACCEPT both are used for the proper handling of a COBOL program by its operator.

- ENTER verb is used to support more than one languages in a COBOL program. A GO TO verb is used for the control to be branched with or without any condition to the first statement of a predefined procedure and execution continue from that point.

- The COPY verb is used to insert library data into the source program and treat it as a part of the source program by the COBOL compiler.

- The PERFORM verb is used to specify the sequence of execution of a COBOL modular program known as range of the PERFORM statement. When ever a PERFORM statement is reached in a COBOL program, then a temporary departure from the normal sequential execution take place.

1.4 Key words

Perform, stop, copy, use, go to, accept, display.
1.5 Self Assessment Questions (SAQ)

1. What is the significance of COBOL verbs in the COBOL programming?
2. Differentiate between ACCEPT and DISPLAY statements with examples.
3. How CLOSE statement is differing from STOP statement in a COBOL program?
4. Discuss unconditional jump in COBOL.
5. Describe following with examples:
   (i) ENTER  (ii) COPY
   (iii) USE   (iv) DISPLAY
6. How the PERFORM verb can be used in a COBOL program?
7. Discuss syntax and purpose of different forms of PERFORM verb.
8. Distinguish between READ and ACCEPT, WRITE and DISPLAY statements.
9. Explain the usage of READ and WRITE verbs with suitable examples.

1.6 References/Suggested Readings

- COBOL Programming by M.K. Roy and D. Dastidar; TMH
- Schaum’s outline series Programming with Structured COBOL; MGH
- Comprehensive COBOL, vol I, Fundamentals of COBOL programming, 4/e by A.S. Philippakis and Leonard J. Kazmier; TMH
- Comprehensive COBOL, vol II, Advanced COBOL programming, 4/e by A.S. Philippakis and Leonard J. Kazmier; TMH
- Structured COBOL: Fundamentals and style, 4/e by Welburn; TMH
- Computer Programming in COBOL by V. Rajaraman; PHI
- Fundamentals of Structured COBOL Programming by Carl Feingold; Galgotia Booksource.
LESSON 5
COBOL Verbs-II

1.0 Objectives

- This chapter describes more COBOL verbs.
- Arithmetic verbs ADD, SUBTRACT, MULTIPLY, DIVIDE, COMPUTE, EXPONENT – would be described in this chapter.
- Data manipulation verb MOVE will be discussed.

1.1 Introduction

As you learnt in the last chapter, COBOL verbs are building blocks for the PROCEDURE DIVISION in the COBOL program. Every program requires some arithmetic calculations be performed. To do arithmetic calculations, COBOL provides many arithmetic verbs. In this chapter, you will learn the ways in which arithmetic may be performed in COBOL. Formats and options available with the arithmetic verbs will also be described.

Most of the time you need to move the content of one memory location to another location. COBOL provides MOVE verb for transfer of information from one location to another. This chapter will describe the use of this verb.

1.2 Presentation of contents

COBOL provides many arithmetic verbs - ADD, SUBTRACT, MULTIPLY, DIVIDE, COMPUTE and EXPONENT - to perform arithmetic calculations. Three options – GIVING, ROUNDED and ON SIZE ERROR – can be used
with these arithmetic verbs. The syntax and formats of these verbs along with their options are describe in the following paragraphs.

### 1.2.1 ADD verb

Add verb is used for the addition of the two or more numeric operands and finally stores the result in the predefined location. It must be noted that every identifier of the add verb must refer to an elementary numeric data item except in case of identifier following the word GIVING. With the ADD verb all the three options i.e. GIVING, ROUNDED and ON SIZE ERROR can be used. There are two syntaxes for ADD verb as given below.

**Syntax-1**

```plaintext
ADD { identifier-1 [ identifier-2 ] … TO identifier-m 
[identifier-n [ROUNDED]] … [ ON SIZE ERROR imperative-statement] 
```

**Syntax-2**

```plaintext
ADD { identifier-1 literal-1 }, { identifier-2 literal-2 }[ identifier-3 literal-3 ] …
GIVING identifier-m [ROUNDED] [identifier-n [ROUNDED]] …
[; ON SIZE ERROR imperative-statement ]
```

**Syntax rules for ADD verb:**

- It must consist of at least two operands.
- In case of syntax-1, the values of the operands preceding to the TO are added and the result must be stored/overwritten to the identifier-
m.
- In case of syntax-2, GIVING must be followed by the two operands.
- The decimal point is automatically aligned.
• The words TO and GIVING may be specified in the same statement if you are using a COBOL-85 compiler.
• The ROUNDED option is always used with the destination field.
• If ROUNDED option is not used than result is truncated, in case destination field cannot accommodate all the decimal positions in the result.
• When ROUNDED option is used, the compiler will always round the result to the PIC specification of the destination field.
• When ON SIZE ERROR option is used, imperative statement specified after the ON SIZE ERROR is executed whenever size error occurs. Size error occurs when destination field is smaller than the result to be stored in the destination.

Example 1: The following example illustrate the use of the ADD verb in various formats.

....

DATA DIVISION.
WORKING-STORAGE SECTION.
    77 DATA1 PIC 99V99 VALUE 5555.
    77 DATA2 PIC 99V99 VALUE 1111.
    77 DATA3 PIC 99V99 VALUE 9999.
    77 SUM1 PIC 99V99 VALUE ZERO.
    77 SUM2 PIC 99V99.
    77 SUM3 PIC 999V9.

PROCEDURE DIVISION.
ADD-PARA.
    ADD DATA1 TO SUM1.
    DISPLAY SUM1.  [displays 5555]
    ADD DATA1, DATA2 GIVING SUM2.
    DISPLAY SUM2.  [displays 6666]
    ADD DATA1, DATA3 GIVING SUM3
       ROUNDED.
    DISPLAY SUM3.  [displays 0667]
ADD DATA1, DATA3 GIVING SUM2
ON SIZE ERROR DISPLAY “SIZE ERROR”.
FINISH-PARA.
STOP RUN.

1.2.2 SUBTRACT Verb

SUBTRACT is used to subtract one or the sum of two or more numbers from one or more numbers and finally stores the results in the predefined location(s).

Syntax-1

\[
\text{SUBTRACT} \left\{ \text{identifier-1} \right\} \left[ \text{identifier-2} \right] \ldots \text{FROM} \text{identifier-m} \left[ \text{identifier-n} \right. \left[ \text{ROUNDED} \right] \ldots \right] \left[ \text{ON SIZE ERROR imperative-statement} \right]
\]

Syntax-2

\[
\text{SUBTRACT} \left\{ \text{identifier-1} \right\} \left[ \text{identifier-2} \right] \ldots \text{FROM} \left\{ \text{identifier-3} \right\} \left[ \text{identifier-n} \right. \left[ \text{ROUNDED} \right] \ldots \right] \left[ ; \text{ON SIZE ERROR imperative-statement} \right]
\]

Syntax rules for SUBTRACT verb:

- All the operands must be numeric in nature.
- In case of syntax-1, the sum of the values of the operands preceding the FROM are subtracted from the identifiers after the FROM and the result is stored/overwritten in the identifier-m.
- if GIVING option is used then destination fields will be after the word GIVING.
- The decimal point is automatically aligned.
• The ROUNDED option is always used with the destination field.
• If ROUNDED option is not used than result is truncated, in case destination field cannot accommodate all the decimal positions in the result.
• When ROUNDED option is used, the compiler will always round the result to the PIC specification of the destination field.
• When ON SIZE ERROR option is used, imperative statement specified after the ON SIZE ERROR is executed whenever size error occurs.

Example 2: The following example illustrate the use of the SUBTRACT verb in various formats.

....
DATA DIVISION.
WORKING-STORAGE SECTION.
    77 DATA1 PIC 99V99 VALUE 5555.
    77 DATA2 PIC 99V99 VALUE 1111.
    77 DATA3 PIC 99V99 VALUE 9999.
    77 DIFF1 PIC 99V99 VALUE ZERO.
    77 DIFF2 PIC 99V99.
    77 DIFF3 PIC 999V9.
PROCEDURE DIVISION.
SUBTRACT-PARA.
    SUBTRACT DATA2 FROM DATA1.
    DISPLAY DATA1. [displays 4444]
    SUBTRACT DATA1 DATA2 FROM DATA3.
    DISPLAY DATA3. [displays 3333]
    SUBTRACT DATA1 FROM DATA3 GIVING DIFF1.
    DISPLAY DIFF1. [displays 4444]
    SUBTRACT DATA1, DATA2 FROM DATA3 GIVING DIFF3 ROUNDED.
    DISPLAY DIFF3. [displays 0333]
    SUBTRACT DATA1, DATA2 FROM DATA3 GIVING DIFF3
1.2.3 MULTIPLY Verb

The MULTIPLY verb is used to multiply one or more values (known as multiplicands) by a multiplier and finally results are stored in the destination fields.

Syntax-1

\[
\text{MULTIPLY}\left\{\begin{array}{c}
\text{identifier-1} \\
\text{literal-1}
\end{array}\right\} \text{ BY identifier-2 [ROUNDED]} \\
\text{[identifier-3[ROUNDED]]} \ldots \ [; \text{ON SIZE ERROR imperative-statement}]
\]

Syntax-2

\[
\text{MULTIPLY}\left\{\begin{array}{c}
\text{identifier-1} \\
\text{literal-1}
\end{array}\right\} \text{ BY } \left\{\begin{array}{c}
\text{identifier-2} \\
\text{literal-2}
\end{array}\right\} \\
\text{GIVING identifier-3 [ROUNDED]} \ [\text{identifier-4[ROUNDED]}] \ldots \\
[; \text{ON SIZE ERROR imperative-statement}]
\]

Syntax rules for MULTIPLY verb:

- All the operands must be numeric in nature.
- In case of syntax-1, products of multiplier and multiplicands are stored/overwritten in the identifier-2, identifier3, ...
- In syntax2 the product is stored in the identifiers written after GIVING.
- The decimal point is automatically aligned.
- The ROUNDED option is always used with the destination field.
• If ROUNDED option is not used than result is truncated, in case destination field cannot accommodate all the decimal positions in the result.
• When ROUNDED option is used, the compiler will always round the result to the PIC specification of the destination field.
• When ON SIZE ERROR option is used, imperative statement specified after the ON SIZE ERROR is executed whenever size error occurs.

Illustrative statements:

MULTIPLY 0.5 BY TOTAL-LECT ROUNDED.

In this statement, the value TOTAL-LECT is multiplied by a factor 0.5 and the results are over-written in TOTAL-LECTURE and result will be rounded.

MULTIPLY A BY C D E .

In this statement, the value of A is multiplied by C and the product is stored in C; the value of A is multiplied by D and the product is stored in D; , the value of A is multiplied by E and the product is stored in E.

MULTIPLY A BY B GIVING D.

In this statement, the values of A and B are multiplied and the product is stored in different identifier i.e. D. Note that, in this computation the previous value of the D will be lost.

MULTIPLY A BY C D GIVING L M.

In this statement, the results of multiplication of A with C and A with D are stored in L and M respectively.
Example 3: The following example illustrate the use of the MULTIPLY verb in various formats.

....

DATA DIVISION.

WORKING-STORAGE SECTION.

77 DATA1 PIC 99V99 VALUE 1000.
77 DATA2 PIC 99V99 VALUE 1111.
77 DATA3 PIC 99V99 VALUE 9999.
77 PROD1 PIC 9999V99 VALUE 000100.
77 PROD2 PIC 9999V99.
77 PROD3 PIC 999V9.

PROCEDURE DIVISION.

MULTIPLY-PARA.

MULTIPLY DATA3 BY PROD1.
DISPLAY PROD1. [displays 001000]
MULTIPLY DATA1 BY DATA3 GIVING PROD2.
DISPLAY PROD2. [displays 099990]
MULTIPLY DATA1 BY DATA3 GIVING PROD3 ON SIZE ERROR DISPLAY “SIZE ERROR”.

FINISH-PARA.

STOP RUN.

1.2.4 DIVIDE verb

This verb is used to divide one numeric data item by another and finally stores the results in the destination fields. There are five different syntax of DIVIDE verb as given below:

Syntax-1 (DIVIDE ... INTO)

DIVIDE { identifier-1 literal-1 } INTO identifier-2 [ROUNDED]
[identifier-3[ROUNDED]]... [; ON SIZE ERROR imperative-statement]

Syntax-2 (DIVIDE...INTO...GIVING)

DIVIDE { identifier-1 literal-1 } INTO { identifier-2 literal-2 }
GIVING identifier-3 [ROUNDED] [identifier-4[ROUNDED]]...
[; ON SIZE ERROR imperative-statement ]

Syntax-3 (DIVIDE...BY...GIVING)

DIVIDE { identifier-1 literal-1 } BY { identifier-2 literal-2 }
GIVING identifier-3 [ROUNDED] [identifier-4[ROUNDED]]...
[; ON SIZE ERROR imperative-statement ]

Syntax-4 (DIVIDE...INTO...GIVING...REMAINDER)

DIVIDE { identifier-1 literal-1 } INTO { identifier-2 literal-2 }
GIVING identifier-3 [ROUNDED] REMAINDER [identifier-4[ROUNDED]]...
[; ON SIZE ERROR imperative-statement ]

Syntax-5 (DIVIDE...BY...GIVING...REMAINDER)

DIVIDE { identifier-1 literal-1 } BY { identifier-2 literal-2 }
GIVING identifier-3 [ROUNDED] REMAINDER [identifier-4[ROUNDED]]...
[; ON SIZE ERROR imperative-statement ]
Syntax rules for DIVIDE verb:

- In syntax1, identifier-2 is divided by identifier-1/literal-1 and result is stored in identifier-2.
- In syntax2, identifier-2/literal-2 is divided by identifier-1/literal-1 and result is stored in identifier-3, identifier-4, ...
- In syntax3, identifier-1/literal-1 is divided by identifier-2/literal-2 and result is stored in identifier-3, identifier-4, ...
- In syntax4, identifier-2/literal-2 is divided by identifier-1/literal-1 and quotient is stored in identifier-3 and remainder is stored in identifier-4.
- In syntax5, identifier-1/literal-1 is divided by identifier-2/literal-2 and quotient is stored in identifier-3 and remainder is stored in identifier-4.
- The ROUNDED option is always used with the destination field.
- If ROUNDED option is not used than result is truncated, in case destination field cannot accommodate all the decimal positions in the result.
- When ROUNDED option is used, the compiler will always round the result to the PIC specification of the destination field.
- When ON SIZE ERROR option is used, imperative statement specified after the ON SIZE ERROR is executed whenever size error occurs. Size error occurs when destination field is smaller than the result to be stored in the destination.

Illustrative statements for DIVIDE verb:

```
DIVIDE 8 INTO X.
```

In this statement, 8 divides the value of X and the result is overwritten in X i.e. destination field.

```
DIVIDE 8 INTO X GIVING Z.
```
In this statement, 8 divides the value of X and the result is overwritten in the identifier Z.

```
DIVIDE 8 BY X GIVING Z.
```

In this statement, X divides the value 8 and the result is overwritten in the identifier Z.

```
DIVIDE X INTO Y GIVING Z REMAINDER U.
```

In this statement Y will be divided by X, quotient is stored in Z and remainder is stored in U. Let the values of ‘X’, ‘Y’, ‘Z’ and ‘U’ are 04, 35, 12 and 10 respectively then after execution of the statement the values of ‘X’, ‘Y’, ‘Z’ and ‘U’ becomes 04, 35, 08 and 03 respectively.

```
DIVIDE X BY Y GIVING Z REMAINDER U.
```

In this statement X will be divided by Y, quotient is stored in Z and remainder is stored in U. Let the values of ‘X’, ‘Y’, ‘Z’ and ‘U’ are 04, 35, 12 and 10 respectively then after execution of the statement the values of ‘X’, ‘Y’, ‘Z’ and ‘U’ becomes 04, 35, 00 and 04 respectively.

Example 4: The following example illustrate the use of the MULTIPLY verb in various formats.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
  77 DIVIDEND PIC 99 VALUE 75.
  77 DVISOR PIC 99 VALUE 16.
  77 QUOTIENT PIC 99.
  77 REMAIN PIC 99.
PROCEDURE DIVISION.
  DIVIDE-PARA.
    DIVIDE DIVISOR INTO DIVIDEND
```
1.2.5 COMPUTE verb

In COBOL arithmetic operations also support another important verb known as COMPUTE. COMPUTE is used to specify a number of arithmetic operations (ADD, SUBTRACT, MULTIPLY and DIVIDE). Therefore, whenever you use more than one arithmetic operation in a computation, then you should use COMPUTE verb.

Syntax

```
COMPUTE identifier-1 [ROUNDED] [, identifier-2 [ROUNDED]]…
= arithmetic operation [: ON SIZE ERROR imperative-statement ]
```

<table>
<thead>
<tr>
<th>Arithmetic Operators</th>
<th>Function</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>+2</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>-3</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>5*3</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>6/2</td>
</tr>
<tr>
<td>**</td>
<td>Exponent</td>
<td>2**3</td>
</tr>
</tbody>
</table>
Table 6.1: Arithmetic operators with function and their use.

In arithmetic only COMPUTE support the exponent operation, but there are some limitations to use it. So, the following cases are not allowed in the COMPUTE verb because in these cases, you may get some unexpected results.

(i) A non-integer value as an exponent of a negative number.
(ii) A number zero as an exponent of a number zero.
(iii) A negative number as an exponent of a number zero.

Syntax rules for COMPUTE verb

(i) The arithmetic expression must be formed by the use of arithmetic operators and data names or literals.
(ii) At least one space must be there between arithmetic operator and its associated operands.
(iii) In the absence of the parentheses, the priority of the operators (from left to right) is in the following order:
   (a) Unary negation (-)
   (b) Exponentiation (**).
   (c) Multiplication (*) and Division (/).
   (d) Addition (+) and Subtraction (-).
(iv) If parentheses are present, then innermost parentheses are solved first, then outer. Within the parentheses, same order of precedence of operators is followed.
(v) No two arithmetic operators can appear together in an expression (** is considered as a single operator).
(vi) If the arithmetic expression is preceded by a +, then it is called unary + operator. If the sign is -, then it is called unary – operator.
Examples of some valid arithmetic expressions:

- $A + B$
- $A \times B$
- $A^4$
- $A - B$
- $A / B$
- $-B$
- $A + B / C$
- $A + (B / C)$
- $(A + B) / (C * D)^5$

Examples of some invalid arithmetic expressions:

- $A (B * D)$ is invalid because there is no operator between $A$ and $(B * D)$.
- $A * + B$ is invalid because there are two adjacent operators between operands $A$ and $B$.
- $A/B$ is invalid because operator is not preceded and followed by at least one blank.

Illustrative statements for COMPUTE verb:

Example 5: The following example illustrate the use of the COMPUTE verb.

This example computes simple interest and amount.

```
....
DATA DIVISION.
WORKING-STORAGE SECTION.
77 PRINCIPAL PIC 9999.
77 RATE PIC 99.
77 TIME PIC 99.
77 INTEREST PIC 9999.
77 AMOUNT PIC 99999
PROCEDURE DIVISION.
INPUT--PARA.
    DISPLAY “Enter principal : “.
    ACCEPT PRINCIPAL.
    DISPLAY “Enter rate : “.
    ACCEPT RATE.
    DISPLAY “Enter time : “.
```
ACCEPT TIME.

COMPUTE-PARA.
    COMPUTE INTEREST = (PRINCIPAL * RATE * TIME) / 100.
    COMPUTE AMOUNT = PRINCIPAL + INTEREST.

OUTPUT-PARA.
    DISPLAY "Interest = Rs. ", INTEREST.
    DISPLAY "Amount = Rs. ", AMOUNT.

FINISH-PARA.
    STOP RUN.

1.2.6 MOVE verb

In programming it is very frequent to transport the data from one memory location to another. In COBOL it is done with the help of MOVE verb. In result of a MOVE statement, the value of the variable is copied in to the output area. The variable retains its value but the value of the output area is updated by new value.

Syntax of MOVE:

MOVE {identifier-1} {literal-1} TO identifier-2 [, identifier-3]...

Syntax rules for MOVE verb:

- Value of identifier-1 or literal-1 is transferred to identifier-2, identifier-3, ...

On execution of MOVE statement, the Contents of identifier-1 (or literal-1) are transferred to the identifier-2, identifier-3 etc. Here the contents of all the receiving fields will be replaced by the contents of the sending field but the contents of the sending field remain unchanged.
MOVE statement can be used to send the source data to multiple destinations. MOVE can be of two types: elementary MOVE and GROUP move. When both the fields are elementary type, then data movement is called elementary move. When at least one of the item in the MOVE is group data item, then it is called group move.

MOVE source-field TO destination-field. The rules to move data from the source to the destination fields are summarized in the Table 6.2.

<table>
<thead>
<tr>
<th>Sending data Category</th>
<th>Receiving data Category</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alphabetic</td>
</tr>
<tr>
<td>Alphabetic</td>
<td>Valid</td>
</tr>
<tr>
<td>Alphanumeric</td>
<td>Valid</td>
</tr>
<tr>
<td>Integer</td>
<td>Invalid</td>
</tr>
<tr>
<td>Non integer</td>
<td>Invalid</td>
</tr>
</tbody>
</table>

**Table 6.2:** Transfer of data from source field to destination field

The effects of the different types of the MOVE statement can be summarized in the following table:
Table 6.3 Effects of transfers

<table>
<thead>
<tr>
<th>MOVE Type</th>
<th>Receiving item</th>
<th>Compiler action</th>
<th>Alignment</th>
<th>Padding</th>
<th>Truncation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphanumeric</td>
<td>Group</td>
<td>None</td>
<td>Left</td>
<td>Right</td>
<td>Right</td>
</tr>
<tr>
<td></td>
<td>Alphabetic</td>
<td>Conversion</td>
<td>Left</td>
<td>Right</td>
<td>Right</td>
</tr>
<tr>
<td></td>
<td>Alphanumeric</td>
<td>Conversion if required</td>
<td>At decimal point</td>
<td>Right</td>
<td>Right</td>
</tr>
<tr>
<td>Numeric</td>
<td>External Decimal/ Packed decimal</td>
<td>Conversion if required</td>
<td>At decimal point</td>
<td>Right &amp; Left with zeros</td>
<td>Left &amp; Right</td>
</tr>
<tr>
<td>Edit</td>
<td>Edited</td>
<td>Editing + conversion</td>
<td>At decimal point</td>
<td>Right &amp; Left with zeros</td>
<td>Left &amp; Right</td>
</tr>
</tbody>
</table>

The result produced by the computer may not be fit for users. Before it is printed or displayed, it must be edited. The MOVE statement of COBOL supports the editing of the data. It may allow you to insert, replace and delete characters w.r.t. a given data. Editing characters used in the PIC clause are described in chapter 8.

Example 6: This example illustrates the use of the MOVE verb and editing characters. This example computes simple interest and amount.

.....

DATA DIVISION.

WORKING-STORAGE SECTION.

01 INPUT-FIELDS.
  05 PRINCIPAL PIC 9999V99.
  05 RATE PIC 99V99.
  05 TIME PIC 99V99.

01 OUTPUT-FIELDS.
  05 OPRINCIPAL PIC ZZZZ.99.
  05 ORATE PIC ZZ.ZZ.
  05 OTIME PIC ZZ.ZZ.
  05 OINTEREST PIC ZZZZ.ZZ.
PROCEDURE DIVISION.

INPUT--PARA.
    DISPLAY "Enter principal : ".
    ACCEPT PRINCIPAL.
    DISPLAY "Enter rate : ".
    ACCEPT RATE.
    DISPLAY "Enter time : ".
    ACCEPT TIME.

COMPUTE-PARA.
    COMPUTE INTEREST = (PRINCIPAL * RATE * TIME) / 100.
    COMPUTE AMOUNT = PRINCIPAL + INTEREST.

MOVE-PARA.
    MOVE PRINCIPAL TO OPRINCIPAL.
    MOVE TIME TO OTIME.
    MOVE RATE TO ORATE.
    MOVE INTEREST TO OINTEREST.
    MOVE AMOUNT TO OAMOUNT.

OUTPUT-PARA.
    DISPLAY "Principal = Rs. ", PRINCIPAL.
    DISPLAY "Time = ", TIME, " Years".
    DISPLAY "Rate = ", RATE, " %".
    DISPLAY "Interest = Rs. ", OINTEREST.
    DISPLAY "Amount = Rs. ", OAMOUNT.

FINISH-PARA.
    STOP RUN.
1.3 Summary

- There are five arithmetic verbs in COBOL: ADD, SUBTRACT, MULTIPLY, DIVIDE and COMPUTE.

- There are three options – ROUNDED, GIVING, ON SIZE ERROR - that can be used with most of these arithmetic verbs.

- ROUNDED option is used with destination field in these arithmetic statements. This option rounds the value according to the PIC size of the destination field.

- When GIVING option is used, destination field is the identifier(s) after the GIVING. This option is not available with the COMPUTE verb.

- When ON SIZE ERROR option is used, then the imperative statement specified after this is executed only when size error occurs.

- The five arithmetic verbs form imperative statements when ON SIZE ERROR option is not used. If ON SIZE ERROR option is used then they form conditional statement.

- MOVE statement can be sued to send the source data to multiple destinations. MOVE can be of two types: elementary MOVE and GROUP move. When both the fields are elementary type, then data movement is called elementary move. When at least one of the item in the MOVE is group data item, then it is called group move.

1.4 Key words

add, subtract, multiply, divide, compute, remainder, move, error, giving, rounded, size.
1.5 Self Assessment Questions (SAQ)

1. Write the following algebraic expression using the COMPUTE verb:-
   (a) \( \frac{A \cdot (B + C - D)^3}{A + D} \)
   (b) \( A^4 - 5 \cdot C \cdot D \)
   (c) \( \frac{B + C - D}{A} \cdot C+B \)

2. Calculate the following expressions, if the data items are described in the WORKING-STORAGE section as:
   77 A PIC 7(2).
   77 B PIC 7(3)V99.
   Expressions are :-
   (i) COMPUTE A = 9 + 5.
   (ii) COMPUTE A = 5.3 / 2.0 – 1.0.

3. Find out the incorrect statement and give reasons for being incorrect.
   (i) COMPUTE A= 3*X + Z ROUNDED.
   (ii) COMPUTE X,Y ROUNDED = 2 * A – C/D.
   (iii) COMPUTE X = L- M + K/N.
   (iv) SUBTRACT X FROM 245, B.
   (v) SUBTRACT X,Y FROM P,L GIVING M,N.
   (vi) MULTIPLY C BY 35.
   (vii) MULTIPLY -8.5 BY A.
   (viii) DIVIDE A INTO 5.
   (ix) DIVIDE C INTO D GIVING L,M.

4. What are different types of arithmetic verbs in COBOL? Give their syntax and explain with examples.

5. Discuss the different types of options that can be used with the arithmetic verbs.

6. What is purpose of MOVE verb? Discuss elementary and group move of data with examples.

1.6 References/Suggested Readings

1. COBOL Programming by M.K.Roy and D..Dastidar ; TMH
2. Schaum’s outline series Programming with Structured COBOL ; MGH
4. Comprehensive COBOL, vol-II, Advanced COBOL programming, 4/e by A.S.Philippakis and Leonard J. Kazmier; TMH
5. Structured COBOL: Fundamentals and style, 4/e by Welburn; TMH
6. Computer Programming in COBOL by V. Rajaraman; PHI
7. Fundamentals of Structured COBOL Programming by Carl Feingold; Galgotia Booksource.
LESSON 7
ADVANCED COBOL VERBS

1.0 Objectives

- This chapter discusses advanced COBOL verbs
- To describe ADD with TO option.
- To introduce INITIALIZE verb.
- To know the impact of INSPECT with CONVERTING option.
- To introduce CONTINUE and compare it with EXIT.
- To discuss PERFORM WITH TEST AFTER verb.

1.1 Introduction

In 1985, the American National Standards Institute (ANSI) introduces some advanced features to the COBOL and the revised version is known as COBOL-85. The COBOL-85 is a revised version of the COBOL-74. COBOL-85 comes with a number of new features that also includes some of the features of COBOL-74 as it is or with some modifications. At the same time some of the unwanted features of COBOL-74 have been deleted. COBOL-85 supports the structured programming in true sense.

1.2 Presentation of contents

1.2.1 ADD with GIVING

COBOL-85 introduces the ADD verb with GIVING phrase using an optional word TO in the syntax of the ADD verb as shown below:

Syntax of ADD with TO and GIVING:
ADD \{ identifier-1 \} \ldots \text{To} \\{ identifier-2 \} \text{GIVING identifier-3}

In this syntax, identifier-1/literal-1, \ldots and identifier-2/literal-2 are added and result can be stored in new location i.e. identifier-3.

Therefore, according to the syntax given above, 

ADD X TO Y GIVING Z.

is a valid COBOL statement.

1.2.2 INITIALIZE verb

COBOL-85 introduces a new verb - INITIALIZE, which provides an easy way to move data to the selected fields. In advanced versions of COBOL mostly INITIALIZE verb is used for setting numeric fields to zero and nonnumeric fields to spaces.

Syntax of INITIALIZE

\[
\text{INITIALIZE} \quad \{\text{identifier-1}\} \ldots \\
\begin{bmatrix}
\text{REPLACING} & \{\{\text{ALPHABETIC, ALPHANUMERIC, NUMERIC, ALPHANUMERIC-EDITED, NUMERIC-EDITED}\}\} \\
\{\text{identifier-2} \} & \{\text{literal-1} \} \\
\end{bmatrix} \text{DATA BY} \ldots
\]

This verb is used to initialize identifier-1 (a group item or an elementary item). If the identifier-1 refers to a group item then only those items that belong to the category defined by the REPLACING phrase will be initialized by the value denoted by the identifier-2/literal-1.

Illustrating INITIALIZE using MOVE verb:
Consider the following segment of the data division.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 X.
   02 L PIC A(2).
   02 M PIC X(2).
   02 N PIC 9V9.
   02 O PIC X/X.
   02 P PIC $9.99.
   02 Q PIC X(2).
01 A PIC 9V9 VALUE 1.5.
01 B PIC XX VALUE ‘15’.

On the basis of the DATA DIVISION specifications given above, we can compare the INITIALIZE statement with the MOVE statement. INITIALIZE statement equivalent MOVE statements are given in Table 7.1.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>INITIALIZE Statement</th>
<th>Equivalent MOVE statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>INITIALIZE X</td>
<td>MOVE SPACES TO L, M, O, Q</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MOVE ZERO TO N, P</td>
</tr>
<tr>
<td>2.</td>
<td>INITIALIZE L</td>
<td>MOVE SPACES TO L</td>
</tr>
<tr>
<td>3.</td>
<td>INITIALIZE X REPLACING NUMERIC BY ZERO</td>
<td>MOVE ZERO TO N</td>
</tr>
</tbody>
</table>

Table 7.1 Comparing INITIALIZE with MOVE verb

From the Table 7.1, it is clear that an INITIALIZE statement is equivalent to a number of MOVE statements. The function of the INITIALIZE verb is similar to the VALUE clause but in case of VALUE initialization take place only once at
the starting of the program. Thus INITIALIZE statement is more powerful statement as compared to MOVE statement.

1.2.3 INSPECT with CONVERTING

INSPECT verb is discussed in chapter 14. but in this chapter it is discussed with a new option i.e. CONVERTING.

Syntax of INSPECT verb

```
INSPECT CONVERTING
    identifier-1 { identifier-2 literal-1 } TO { identifier-3 literal-2 }
    [{ BEFORE AFTER }] INITIAL { identifier-4 literal-3 }
```

The INSPECT with CONVERTING verb is used to replace the matched characters in identifier-1 by some other characters and the identifier-2/literal-1, identifier-3/literal-2 gives the matching/replacement criteria. The identifier-2/literal-1 and identifier-3/literal-2 are known as subject and object field respectively; these fields must be identical in nature.

For example:

```
INSPECT FIELD-B CONVERTING "LMNOP" BY "PONML".
```

If before execution FIELD-B having value “COMPUTERSCIENCE” then after the execution of the above statement FIELD-B contains the value “CMONUTERSCIENCE”

On the other side:

```
INSPECT X CONVERTING Y TO Z
```

Here this INSPECT is equivalent to:
INSPECT X REPLACING ALL Y(1:1) BY Z(1:1)
   Y(2:1) BY Z(2:1)
   Y(3:1) BY Z(3:1)
   ...
   Y(n:1) BY Z(n:1)

Where ‘n’ be a literal denoted the size of Y and Z.

1.2.4 CONTINUE verb

The CONTINUE verb is another new facility provided by the COBOL’s advanced version to its programmers. Whenever a COBOL compiler faces a CONTINUE statement in a COBOL program it means that no operation.

Syntax of CONTINUE verb:

CONTINUE

The syntax of CONTINUE statement does not require any operand. The programmer can use it any where in a COBOL program with a conditional or an imperative statement.

For example, when it is confirmed that the end-of-file has not occurred then CONTINUE can be used as given below:

READ STUDENT-FILE RECORD
   AT END CONTINUE.

The CONTINUE and the EXIT statements are similar in their operation but differ in their objectives. In COBOL, CONTINUE is an alternate to a null path, on the other side, the EXIT is used for as a common end point for a sequence of paragraphs. The another implementation of CONTINUE is as a NEXT SENTENCE phrase in the IF statement.
1.2.5 USAGE clause

The USAGE clause is used to specify how a data item is to be stored in the computer's memory. It must be noted that every variable declared in a COBOL program has a USAGE clause - even when no explicit clause is specified. By default - USAGE IS DISPLAY - is applied.

For text items, or for numeric items that are not going to be used in a computation (Roll-numbers, Phone Numbers etc.), the default of USAGE IS DISPLAY presents no problems. In case of numeric items those are involved in some calculation, the default usage is not the most efficient way to store the data. When calculations are done with numeric data items with USAGE IS DISPLAY, the compiler has to convert the non-numeric values to their binary equivalents before the calculation can be done. When the result has been computed the computer has to reconvert it to ASCII digits. Hence conversion to and from ASCII digits slows down computations. Due to this reason, data that is heavily involved in computation is often declared using one of the usages optimized for computation such as USAGE IS COMPUTATIONAL. There are two new types of USAGE clause supported by the advanced version of the COBOL namely BINARY and PACKED-DECIMAL.

The syntax for USAGE clause:

```
USAGE IS DISPLAY/DISP

USAGE IS COMPUTATIONAL/COMP

USAGE IS PACKED-DECIMAL
```

The USAGE IS DISPLAY clause means that the standard data format is used to represent the data item. That is, a single position of storage is used to store one character of the data.
USAGE IS COMPUTATIONAL/COMP

COMP items are held in memory as pure binary 2's complement numbers. The storage requirements for fields described as COMP are as follows:

<table>
<thead>
<tr>
<th>Number of Digits</th>
<th>Storage Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC 9(1 to 4)</td>
<td>1 Word (2 Bytes)</td>
</tr>
<tr>
<td>PIC 9(5 to 9)</td>
<td>1 LongWord (4 Bytes)</td>
</tr>
<tr>
<td>PIC 9(10 to 18)</td>
<td>1 QuadWord (8 Bytes)</td>
</tr>
</tbody>
</table>

DATA DIVISION.

WORKING-STORAGE SECTION.

01 TABLE1 USAGE IS COMPUTATIONAL.

05 ITEM1 PIC S9(10).

05 ITEM2 PIC S9(5).

USAGE IS PACKED-DECIMAL

This usage is used to conserve storage space when defining numeric WORKING-STORAGE item as it enables numeric items to be stored as compactly as possible. Data-items declared as PACKED-DECIMAL are held in binary-coded-decimal (BCD) form. Instead of representing the value as a single binary number, the binary value of each digit is held in a nibble (half a byte). The sign is held in a separate nibble in the least significant position of the item.

Consider the example:

DATA DIVISION.
WORKING-STORAGE SECTION.

77 AMOUNT-DISP   PIC 9(7) USAGE IS DISPLAY.
77 AMOUNT-PACK   PIC 9(7) USAGE IS PACKED-DECIMAL.

......

PROCEDURE DIVISION.

PARA1.

....

MOVE 1234567 TO AMOUNT-DISP.

MOVE 1234567 TO AMOUNT-DISP.

....

AMOUNT-DISP takes seven positions of storage as shown below:

1 2 3 4 5 6 7

AMOUNT-PACK takes four positions of storage as shown below:

12 34 56 7+

These examples show that considerable amount of storage space can be saved by using USAGE is PACKED-DECIMAL.

1.2.5 Advanced DISPLAY VERB

The DISPLAY verb was discussed in chapter 5. this has been modified to include some advanced features. The syntax of modified DISPLAY verb is as given below:
Syntax of DISPLAY verb:

```
DISPLAY { identifier-1 literal-1 } ... [upon mnemonic-name] [WITH NO ADVANCING]
```

In this syntax WITH NO ADVANCING phrase is used in case of interactive terminals. In case of normal DISPLAY verb execution the cursor blinks at the very first position of the next line on the screen. But in case of DISPLAY with ‘WITH NO ADVANCING phrase’ then the cursor moves after the last character displayed.

### 1.2.6 IF verb

When a COBOL program runs, the program statements are executed one after another in a sequence unless a statement is encountered that alters the order of execution. An IF statement is one of these types of statements that can alter the order of execution in the program. An IF statement allows the programmer to specify that the block of code is to be executed only if the condition attached to the IF statement is satisfied. The syntax of IF statement is given below:

Syntax of IF verb:

```
IF condition THEN { statement-1 }... NEXT SENTENCE

ELSE { statement-2 }... END-IF
ELSE NEXT SENTENCE END-IF
```

When an IF statement is encountered in a program, the block of statements following the THEN is executed when the condition specified is true, and the block of statements following the ELSE (if used) is executed when the
condition specified is false. The block of statements can include any valid COBOL statement including further IF constructs, PERFORM, etc.

The END-IF makes explicit the scope of the IF statement. Using a full stop to delimit the scope of the IF can lead to problems. For instance, the two IF statements below are supposed to perform the same task. But the scope of the one on the left is delimited by the END-IF, while that on the right is delimited by a full stop.

```
Statement1
Statement2
IF VarX > VarY THEN
  Statement3
  Statement4
END-IF
Statement5
Statement6.
```

Unfortunately, in the IF on the right, the programmer has forgotten to follow Statement4 by a delimiting full stop. This means that Statement5 and 6 will be included in the scope of the IF (that means these statements will only be executed if the condition is true) by mistake. If you use full stop to delimit the scope of an IF statement, this is an easy mistake to make and, once made, it is difficult to spot. A full stop is small and unobtrusive compared to an END-IF.

### 1.2.7 EVALUATE verb

The EVALUATE performs the same task which was done by the CASE, but the EVALUATE verb has more powerful features as compared to CASE. The syntax of EVALUATE verb is as given below:

Syntax of EVALUATE
EVALUATE subject-1 [ ALSO subject-2 ] ...

{{ WHEN object-1 [ ALSO object-2] ...}...} imperative-statement-1}...

[ WHEN OTHER imperative-statement-1] [ END-EVALUATE ]

In the syntax of EVALUATE verb the subject can be as:

{ identifier
  literal
  expression
  TRUE
  FALSE
}

In the syntax of EVALUATE verb the object can be as:

{ ANY
  Condition
  TRUE
  FALSE
  [NOT] } { identifier-1
  literal-1
  arithmetic-expression-1 } { THROUGH
  THRU
}

{ identifier-1
  literal-1
  arithmetic-expression-1
}

Consider an input data item NUMBER-OF-YEARS is used to perform the type of processing to be performed. The following code shows the type of processing performed:

IF NUMBER-OF-YEARS = 1
    PERFORM FIRST-YEAR.

IF NUMBER-OF-YEARS = 2
    PERFORM SECOND-YEAR.

IF NUMBER-OF-YEARS = 3
PERFORM  THIRD-YEAR.

IF     NUMBER-OF-YEARS = 4
      PERFORM  FOURTH-YEAR.

To ensure correct processing let us add fifth condition to check the error:

IF     NUMBER-OF-YEARS     IS NOT = 1    AND
      IS NOT = 2    AND
      IS NOT = 3    AND
      IS NOT = 4    AND

      PERFORM  ERROR-ROUTINE.

These statements can be encoded by using EVALUATE statement more easily, clearly and efficiently as given below:

EVALUATE  NUMBER-OF-YEARS

   WHEN     1     PERFORM  FIRST-YEAR
   WHEN     2     PERFORM  SECOND-YEAR
   WHEN     3     PERFORM  THIRD-YEAR
   WHEN     4     PERFORM  FOURTH-YEAR
   WHEN     OTHER  PERFORM  ERROR-ROUTINE

END-EVALUATE

The WHEN OTHER clause is executed when NUMBER-OF-YEARS is not 1, 2, 3, or 4.
The another way to write the preceding EVALUATE is as given below:

EVALUATE TRUE

   WHEN NUMBER-OF-YEARS = 1
       PERFORM FIRST-YEAR
   WHEN NUMBER-OF-YEARS = 2
       PERFORM SECOND-YEAR
   WHEN NUMBER-OF-YEARS = 3
       PERFORM THIRD-YEAR
   WHEN NUMBER-OF-YEARS = 4
       PERFORM FOURTH-YEAR
   WHEN OTHER
       PERFORM ERROR ROUTINE

END-EVALUATE

1.2.8 PERFORM with TEST AFTER option

In COBOL-85, a PERFORM...UNTIL can be made equivalent to a Repeat...Until with the use of a TEST FTER clause. The syntax of PERFORM with TEST AFTER is given below:

Syntax:

PERFORM [paragraph-name]
WITH TEST {BEFORE/AFTER} UNTIL condition

Example:

PERFORM WITH TEST AFTER

UNTIL NUMBER < 1

PERFORM DISPLAY-NUMBER

SUBTRACT 1 FROM NUMBER

END-PERFORM

In this example, DISPLAY-NUMBER will be performed at least once even if NUMBER is less than 1 in the beginning.

1.3 SUMMARY

- COBOL-85 comes with a number of new features added to COBOL-74. At the same time some of the unwanted features are deleted from COBOL-74. COBOL-85 supports the structured programming in true sense.

- COBOL-85 introduces a new verb INITIALIZE, which provides an easy way to move data to the selected fields only. In advanced versions of COBOL, the INITIALIZE verb is used for setting numeric fields to zero and nonnumeric fields to spaces. An INITIALIZE statement is equivalent to a number of MOVE statements.

- The INSPECT statement can be sued to count the number of occurrences of a given character in a field. It can also be used to replace occurrences of a given character with another character.
• Whenever a COBOL compiler encounters a CONTINUE statement in a COBOL program it replaces it with no operation instruction. The CONTINUE and the EXIT statements are similar in their operation but differ in their objectives.

• In COBOL, CONTINUE is an alternate to a null path, on the other side, the EXIT is used for as a common end point for a sequence of paragraphs.

• DISPLAY statement WITH NO ADVANCING phrase is used for interactive terminals output. In case of normal DISPLAY verb, the cursor blinks at the very first position of the next line on the screen. But in case of DISPLAY with ‘WITH NO ADVANCING phrase' the cursor is placed after the last character displayed.

• IF statement is one of those statements that can alter the order of execution of a program. Number of IF statements can easily and efficiently can be encoded by using EVALUATE statement.

1.4 Key words

COBOL-85, initialize, continue, exit, perform, inspect, test

1.5 Self Assessment Questions (SAQ)

1. Explain the difference between the INITIALIZE and MOVE statement in COBOL with example.
2. What is the significance of a INSPECT verb with CONVERTING phrase?
3. What is the significance of CONTINUE statement in COBOL? How is it different from EXIT statement?
4. Which advanced verb of the COBOL is used in place of IF statements? Give some suitable examples for it.
5. Which special features are included in USAGE verb to handle the computations more efficiently?
6. What is the impact of END-IF phrase in IF statement?

1.6 References/Suggested Readings

1. COBOL Programming by M.K.Roy and D..Dastidar ; TMH
2. Schaum’s outline series Programming with Structured COBOL ; MGH
4. Comprehensive COBOL, vol-II , Advanced COBOL programming, 4/e by A.S.Philippakis and Leonard J. Kazmier ; TMH
5. Structured COBOL: Fundamentals and style, 4/e by Welburn ; TMH
6. Computer Programming in COBOL by V. Rajaraman; PHI
7. Fundamentals of Structured COBOL Programming by Carl Feingold; Galgotia Booksource.
1.0 Objectives

- To cover the various clauses used in the DATA DIVISION of COBOL.
- To describe specification of data items by using PICTURE clause.
- How to initialize data names at the time of compilation using VALUE clause.
- How can you specify internal formats to data names by using USAGE clause?
- How can you give many descriptions to the same storage area by using REDIFINE clause?
- How can you regroup data names by using RENAME clause?
- How can you justify data to the right by using JUSTIFIED clause?
- How can you specify unnamed data names by using FILLER clause?
- How can you specify data names with same names?

1.1 Introduction

COBOL clauses are used for different purposes. Some clauses are used to describe the data items in the data division while others can be used to increase the efficiency of the COBOL program. Data description clauses are used in the DATA DIVISION of a COBOL program. COBOL has a number of clauses such as PICTURE, VALUE, REDIFINE, RENAME, SIGN, JUSTIFIED, USAGE, FILLER etc. This chapter would describe these clauses in sufficient detail.

1.2 Presentation of contents
### 1.2.1 PICTURE Clause

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Picture Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>A</td>
<td>The corresponding character position in the data item contains only a letter or space character.</td>
</tr>
<tr>
<td>2.</td>
<td>B</td>
<td>Each B in a picture string represents one byte into which a blank space will be inserted when data are moved in to field ‘B’.</td>
</tr>
<tr>
<td>3.</td>
<td>P</td>
<td>Indicates the position of the assumed decimal point when the point lies outside the data item.</td>
</tr>
<tr>
<td>4.</td>
<td>S</td>
<td>Indicates that the data item is signed.</td>
</tr>
<tr>
<td>5.</td>
<td>V</td>
<td>Indicates the position of the assumed decimal point.</td>
</tr>
<tr>
<td>6.</td>
<td>X</td>
<td>Indicates that corresponding data position contains any allowable character of COBOL character set.</td>
</tr>
<tr>
<td>7.</td>
<td>Z</td>
<td>To represent the position of a decimal digit which is to be replaced with a blank space if that digit is a leading zero?</td>
</tr>
<tr>
<td>8.</td>
<td>9</td>
<td>Indicates that the corresponding character position in that data item contains a numeral.</td>
</tr>
<tr>
<td>9.</td>
<td>/</td>
<td>To reserve a byte in the edited result which will always hold a slash(/), it is useful when editing numbers are dates.</td>
</tr>
<tr>
<td>10.</td>
<td>,</td>
<td>To represents one byte of storage in which code for comma will be placed.</td>
</tr>
<tr>
<td>11.</td>
<td>.</td>
<td>Its represents the actual position of the decimal point in a field.</td>
</tr>
<tr>
<td>12.</td>
<td>+ - CR DB</td>
<td>Used in the editing of negative numerical values. CR, DB are negative credit and debit respectively</td>
</tr>
<tr>
<td>13.</td>
<td>*</td>
<td>It is used as Check protection, so that the amount field on a check can be protected from tampering.</td>
</tr>
<tr>
<td>14.</td>
<td>$</td>
<td>To insert a fixed dollar sign that prints immediately to the left of the first digit position.</td>
</tr>
</tbody>
</table>

**Table 8.1** Characters used in the character string of PICTURE clause.
The PICTURE clause describes the format of an elementary item. It may not be specified for a group item. A character string is used to specify the format of the data item. The syntax of PICTURE clause is as given below:

\{PICTURE/PIC\} IS character-string.

The ‘character-string’ of the PICTURE may involve the following code characters:

A  B  P  S  V  X  Z  9  /  ,  +  -  CR  DB  *  $  

The character string may contain 1 to 30 code characters. These code characters can be specified in two ways as shown below:

PIC IS AAAAA.  Or  PIC IS A(5).

Table 8.1 describes the characters used in the PIC clause.

Elementary data items can be classified into three categories – alphabetic, numeric and alphanumeric.

(i) In case of alphabetic data the picture clause may contain only the symbol A.

(ii) In case of numeric data the allowable symbols are 9,V,P and S. The symbols S and V can appears only once and S must be the leftmost character of the picture string. The symbol P can be repeated and a numeric data must contains at least one 9.

(iii) In case of an alphanumeric data, picture may contain all Xs or a combination of 9, A and X but not all 9 or all A.

Examples:

DATA DIVISION.
WORKING-STORAGE SECTION.
01 GROUP-ITEM.
  05 ROLL-NUMBER   PIC 9999.
  05 REG-NO        PIC 99AA9999.
  05 NAME          PIC A(20).
  05 CLASS         PIC X(10).
  05 PERCENTAGE    PIC 99V99.
  77 SUM           PIC 9999.

1.2.2 THE VALUE CLAUSE

The VALUE clause defines the initial value of a data item. The value of the
data item specified by the VALUE clause is used to initialize at the time of
compilation. The syntax for VALUE clause is given below:

```
VALUE IS literal
```

Where the literal can be any numeric value or figurative constant. If it is a
nonnumeric string then it must be included within the quote ("."). The class of
the data item as specified through PICTURE clause must be compatible w.r.t.
its corresponding literal.

For example:

DATA DIVISION.
WORKING-STORAGE SECTION.
01 GROUP-ITEM.
  05 ROLL-NUMBER   PIC 9999
      VALUE IS 1111.
  05 REG-NO        PIC 99AA9999
      VALUE IS "95HR2345".
  05 NAME          PIC A(20)
      VALUE IS "RADHA KRISHAN".
  05 CLASS         PIC X(10)
1.2.3 THE USAGE CLAUSE

Internally data can be stored in different ways. Most of the time, it is done by the system itself. But in case of COBOL, a programmer can control it for the efficient use of the data items. Mainly there are two methods of internal representation i.e. computational (for the numeric data or any other data which can take part in any arithmetic operation) and display (for any data item). The syntax of the USAGE clause is given below:

```cobol
USAGE IS {
   COMPUTATIONAL
   COMP
   DISPLAY
} [integer]
```

Table 8.2 gives the different forms of USAGE clause.

<table>
<thead>
<tr>
<th>S.NO.</th>
<th>USAGE TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>DISPLAY</td>
<td>Every character of the data is represented in one byte and stored at contiguous bytes in memory.</td>
</tr>
<tr>
<td>2.</td>
<td>COMPUTATIONAL</td>
<td>When the numeric data is of pure binary form.</td>
</tr>
<tr>
<td></td>
<td>(COMP)</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>COMP-1</td>
<td>When the numeric data is represented in one word in the floating-point form.</td>
</tr>
</tbody>
</table>
For example:

DATA DIVISION.
WORKING-STORAGE SECTION.
01 GROUP-ITEM.
   05 DATA1 PIC 9999 USAGE IS COMP.
   05 DATA2 USAGE IS COMP-2.
   05 DATA3 PIC A(20) USAGE IS DISPLAY.
   05 DATA4 PIC 9(7) USAGE IS COMP-3.
   05 DATA5 USAGE IS COMP-1.

Note that PIC clause cannot be specified with data items having usage COMP-1 and COMP-2.

1.2.4 The REDIFINE Clause

The REDIFINE clause can be used to allow the same storage location to be referenced by different data-names or allow a regrouping or description of the data in a particular storage location. The syntax of this clause is:-

   Level-number       data-name       REDIFINES       data-name2
Under the following conditions the REDIFINES clause cannot be used:

- It cannot be used at the 01 level in the FILE SECTION.
- It cannot be used when the levels of data-name-1 and data-name-2 are different.
- Further the level-number must not be 66; it is reserved for the RENAME clause.
- There can be as many redefinitions of an item as desired. However, all the redefinitions refer to the first item description.

For example:

DATA DIVISION.
WORKING-STORAGE SECTION.
01 EXAM.
  02 STUDENT.
    10 REGION-ID PIC X(4).
    10 COLLEGE-ID PIC X(5).
    10 STUDENT-ID PIC X(10).
  02 STD-RECORD REDIFINES STUDENT.
    10 REGION-NAME PIC X(4).
    10 COLLEGE-NAME PIC X(5).
    10 STUDENT-NAME PIC X(10).

Here the REDIFINES allow the data-names STUDENT and STD-RECORD to refer to the same 19 positions in the internal storage as shown below.

| STUDENT | | 
|---------|---------|---------|
| REGION-ID | COLLEGE-ID | STUDENT-ID |
| | | |

| STD-RECORD | | 
|-------------|---------|---------|
| REGION-NAME | COLLEGE-NAME | STUDENT-NAME |
| | | |
Through redefinition, you can change the format of the data-item but the overall size of the item remains same. The REDIFINES applies to the storage area involved and not to the data which is stored there.

1.2.5 RENAME Clause

It is used by the programmer for regrouping the elementary data-items. It is similar to REDIFINES except it can form a new grouping of data items that combine several contiguous items. The RENAME clause must be used with the level number-66, its syntax is:

66   data-name-1 RENAMES data-name-2 [THRU data-name-3]

For example:

DATA DIVISION.
WORKING-STORAGE SECTION.
01   EXAMPLE.
   02   STUDENT.
       10   REGION-ID   PIC   X(4).
       10   COLLEGE-ID  PIC   X(5).
       10   STUDENT-ID  PIC   X(10).
   02   RESULT
       10   SEMESTER-1  PIC   X(4).
       10   SEMESTER-2  PIC   X(4).
       10   SEMESTER-3  PIC   X(4).
66   FINAL-RESULT RENAMES STUDENT-ID THRU SEMESTER-3.

This example forms a new group of elementary data items called FINAL-RESULT. The new group consists of STUDENT-ID, SEMESTER-1, SEMESTER-2, SEMESTER-3 as shown below:
1.2.6 THE SIGN CLAUSE

The PICTURE character S specifies that the field is signed. The SIGN clause represents the position and the mode of representation of the operational sign (if it is necessary to represent).

The syntax of SIGN clause:

\[
\text{SIGN IS} \begin{cases} \text{LEADING} \\ \text{TRAILING} \end{cases} [\text{SEPARATE CHARACTER}] 
\]

When the SEPARATE CHARACTER option is used, then the operational sign is actually represented as a separate leading or trailing character i.e. it requires a storage space position. If this clause is not used then sign is stored as a zone bit along with the data.

For example:

DATA DIVISION.
WORKING-STORAGE SECTION.
77 NUM1 PIC S9999
   SIGN IS LEADING SEPARATE CHARACTER.
77 NUM1 PIC S9999
   SIGN IS TRAILING SEPARATE CHARACTER.
In NUM1 sign is stored as a separate character and will be before the data value. In NUM2 sign is stored as a separate character and will be after the data value.

### 1.2.7 THE JUSTIFIED CLAUSE

This clause is used with the elementary alphabetic or alphanumeric items only and its effect is to nullify the by-default left justification of the nonnumeric data. Without the JUSTIFIED RIGHT clause truncation will take place from the right in case of alphanumeric and alphabetic data, but when the JUSTIFIED RIGHT clause is used, truncation takes place from the left. The syntax of JUSTIFIED clause is given below:

Syntax:

```
JUSTIFIED {RIGHT/LEFT}
```

By default, data is justified left. If you want to justify the data to the right then you should use this clause.

For example:

```plaintext
DATA DIVISION.
WORKING-STORAGE SECTION.
77 TITLE PIC X(10)
   VALUE "DISHANT".
```

The value of the TITLE field will be stored as shown below:

```
D I X H A N T
```

```plaintext
DATA DIVISION.
WORKING-STORAGE SECTION.
77 TITLE PIC X(10)
```
The value of the TITLE field will be stored as shown below:

```
D I X H A N T
```

1.2.8 FILLER CLAUSE

When you do not want to assign any name to the storage area that can be specified with the FILLER clause. Syntax of FILLER clause is as shown below:

```
Level-no FILLER PIC character-string
```

FILLER clause is required for COBOL-74 and in COBOL-85; you can leave the field name blank. FILLER clause is generally used to control the spacing between the output fields.

For example:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 IN-REC.
     05 ROLLNO PIC 9999.
     05 NAME PIC X(20).
     05 CLASS PIC X(10).
     05 MARKS PIC 9999.
  01 OUT-REC.
     05 ROLLNO PIC 9999.
     05 FILLER PIC X(5)
        VALUE SPACES.
     05 NAME PIC X(20).
     05 FILLER PIC X(5)
```
In the OUT-REC, five spaces will be introduced between two successive fields.

1.2.9 QUALIFICATION OF DATA NAMES

Data names need not be unique in a COBOL program. They can have the same name. The duplicate data names when used in the procedure division need to be qualified. A qualified data name is followed by the words IN or OF. Syntax for qualified data names is as shown below:

Syntax-1
Data-name1 {OF/IN} Data-name2 [{OF/IN} Data-name3]….

Syntax-2
Data-name1 {OF/IN} File-name

- File name or 01 level data items are the highest-level qualifiers.
- Record name or data record in FILE SECTION can also be qualified by a file name as in syntax-2.
- The same data name cannot appear at different levels in a hierarchy.
- Qualification is normally required in PROCEDURE DIVISION.

For example:

DATA DIVISION.
WORKING-STORAGE SECTION.
The PICTURE clause describes the format of an elementary item. It may not be used with a group item.

The VALUE clause defines the initial value of a data item. The value specified by the VALUE clause is used by the compiler to initialize the data name at the time of compilation.
• The REDIFINE clause can be used to allow the same storage location to be referenced by different data-names or allow a regrouping or description of the data in a particular storage location. There can be as many redefinitions of an item as desired.

• On the other hand RENAMES clause is used to regroup the data items. It is similar to REDIFINES except it can form a new grouping of data items that combine several contiguous items.

• In RENAMES you can not change the PIC of any data item while in REDEFINES you can change the PIC of any data item. That means in REDEFINES you can give a completely new description to the storage.

• The PICTURE character S specifies that the field is signed. Sign is stored as a zone bit along the value of the data item. You can assign a separate storage to the sign by using SIGN clause.

• By default, non-numeric values are justified left. If you want to justify it to the right; it can be done by using JUSTIFIED clause.

• You can store values of data items in many different formats. COBOL provides you USAGE clause to specify the internal storage format.

• In COBOL, every data name need not be unique. But when duplicate data names are used then they need to be qualified.

1.4 Key words

Picture, redefines, renames, usage, display, qualify, justify, filler

1.5 Self-assessment Questions (SAQ)
1. What is the significance of PICTURE clause in a COBOL program? Discuss its use with examples.
2. What character codes can be used in the PICTURE clause? Explain each with suitable examples.
3. What is VALUE clause? Explain its use with example.
4. What are the different formats of internal storage you can specify by using USAGE clause? Explain each with suitable examples.
5. Compare and contrast REDEFINES and RENAMES clauses.
6. What is the importance of FILLER clause? Explain with suitable examples.
7. When do you need qualifiers to qualify a data name? Give example.
8. What is the need of JUSTIFIED clause? Explain with example.

1.6 References/Suggested Readings

- COBOL Programming by M. K. Roy and D. Dastidar; TMH
- Schaum’s outline series Programming with Structured COBOL; MGH
- Comprehensive COBOL, vol-I, Fundamentals of COBOL programming, 4/e by A.S. Philippakis and Leonard J. Kazmier; TMH
- Comprehensive COBOL, vol-II, Advanced COBOL programming, 4/e by A.S. Philippakis and Leonard J. Kazmier; TMH
- Structured COBOL: Fundamentals and style, 4/e by Welburn; TMH
- Computer Programming in COBOL by V. Rajaraman; PHI
- Fundamentals of Structured COBOL Programming by Carl Feingold; Galgotia Booksource.
1.0 Objectives

- To introduce you the concept of Tables in COBOL.
- To learn how to declare one-dimensional and two-dimensional and multi-dimensional tables. How the contents are entered into a table.
- To discuss the COBOL verbs and clauses related to Table handling.

1.1 INTRODUCTION

A table is a group of similar (or logically related) data items i.e. a table is a collection of homogeneous items. Examples are student names, Timetable and Salary-Table. Most of the programming languages use the term "array" to describe repeated, or multiple- occurrences of data-items. COBOL uses the term "table". The repeated components of a table are referred to as its elements.

In the program text, a table is declared by specifying two things 1) the type, or structure, of a single data-item (element), 2) the number of times the data-item (element) is repeated.

Tables have the following attributes

- A single name is used to identify all the elements of a table.
- Individual elements can be identified using an “index” or “subscript”.
- All elements of a table have the same type or structure.

Unlike other programming languages, the index in COBOL tables always starts from 1 (not from 0) and go on to the maximum size of the table. Using the element name followed by the index/subscript in parentheses can reference the particular element in the table.

A table is stored in memory as a contiguous block of bytes. The elements of a table must follow a sorting order, so that their retrieval becomes easy during their processing. As per the number of columns in a table these can be classified into many categories such as One dimensional, two dimensional and so on. If the dimension of the table is 2 or more then they are called as multi dimensional table.

1.2 Presentation of Contents

1.2.1 THE OCCUR CLAUSE AND SUBSCRIPTING

Consider an example that a student of PGDCA has four different subjects. His marks scored in an examination are to be stored in a table named PGDCA-RESULT. The table PGDCA-RESULT can be described in the data division as given below:

DATA DIVISION.
WORKING-STORAGE SECTION.
01 PGDCA-RESULT.
   02 SUB-1   PIC 99.
   02 SUB-2   PIC 99.
   02 SUB-3   PIC 99.
   02 SUB-4   PIC 99.

In the above example, each element of the table are identical in description i.e. they have PIC 99. In such a situation where similar structure of data items
occur, we can alternatively define these items by using OCCURS clause.

Syntax of OCCURS clause:

\[
\text{OCCURS integer TIMES}
\]

Rules for the OCCURS clause

- The integer in the occur clause must be positive.
- This clause can be specified for elementary as well as group data names.
- The OCCURS clause cannot be specified for data items whose level number is 01, 66, 77, or 88.
- Data name used as a subscript cannot be another subscripted data name.
- VALUE clause cannot be used with OCCURS clause.
- Any data-item whose description includes occurs clause must be subscripted when referred to.
- Any data-item, which is subordinate to a group item whose description contains, occurs clause must be subscripted when referred to.

For example, the example given above can be rewritten as:

DATA DIVISION.
WORKING-STORAGE SECTION.
01 PGDCA-RESULT.
   02 SUB PIC 99 OCCUR 4 TIMES.

This style of description is very simple and efficient if used for large number of elements in a table. The elements of a table can be referenced in the PROCEDURE DIVISION by a special method called subscripting.

Rules for subscripts
Each subscript must be a positive integer, a data name which represents one, or a simple expression, which evaluates to one.

The subscript must contain a value between 1 and the number of elements in the table/array inclusive.

When more than one subscript is used they must be separated from one another by commas.

One subscript must be specified for each dimension of the table. There must be 1 for a one-dimension table, 2 subscripts for a two-dimension table and 3 for a three-dimension table and so on.

The first subscript applies to the first OCCURS clause, the second applies to the second OCCURS clause, and so on.

Subscripts must be enclosed in parentheses.

For example for one dimensional Table:

DATA DIVISION.
WORKING-STORAGE SECTION.
01 PGDCA-RESULT.
   02 SUB PIC 99 OCCUR 4 TIMES.
PROCEDURE DIVISION.
PUT-VALUE-PARA.
   MOVE 78 TO SUB (1).
   MOVE 88 TO SUB (2).
   MOVE 95 TO SUB (3).
   MOVE 87 TO SUB (4).

Another example of one-dimensional table: Suppose you want to store records of 20 students.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 STUDENT-RECORD.
  05 RECORD-TABLE OCCURS 20 TIMES.
    10 ROLL-NUMBER PIC 9999.
    10 REG-NO PIC 99AA9999.
    10 NAME PIC A(20).
    10 CLASS PIC X(10).

PROCEDURE DIVISION.
PUT-VALUE-PARA.
  MOVE 1111 TO ROLL-NUMBER (1).
  MOVE 88KL2345 TO REG-NO (1).
  MOVE "ABC" TO NAME (1).
  MOVE "PGDCA" TO CLASS (1).

In the above example, subscript indicates the student number. All MOVE statements store the values for the first student.

A table in such a format that every entry of it is a one-dimensional table itself, is known as two-dimensional table.

For example for two-dimensional Table: Suppose a person wants to store commission received from 20 branches for 12 months.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 TWO-DIMENSIONAL-TABLE.
  05 BRANCHES OCCURS 20 TIMES.
    10 MONTHLY-COMM PIC 9999V99 OCCURS 12 TIMES.

PROCEDURE DIVISION.
PUT-VALUE-PARA.
MOVE 1278 TO MONTHLY-COMM (5, 10).
DISPLAY BRANCHES (5).

In this example, MONTHLY-COMM (5, 10) refers to fifth branch and tenth month. On the other hand, BRANCHES (5) refers to all monthly commissions of fifth branch. That means BRANCHES (5) is an array of 12 elements.

Similarly, we can define multidimensional tables as shown in example below:

DATA DIVISION.
WORKING-STORGAE SECTION.
 01 MULTI-TABLE.
    05 FIRST-DIM OCCURS 10 TIMES.
    10 SECOND-DIM OCCUR 5 TIMES.
    15 DATA PIC 99 OCCURS 5 TIMES.

PROCEDURE DIVISION.

PUT-VALUE-PARA.

    MOVE 12 TO DATA (5, 2, 4).

1.2.2 INSERTING VALUES INTO A TABLE

The values to table elements can be assigned via two different methods:

In the first method, to insert the values in a table is to assign initial values to table elements in the DATA DIVISION through REDIFINE clause. The following example illustrates this method:

DATA DIVISION.
WORKING-STORGAE SECTION.
 01 MONTHS-TABLE.
      02 FILLER PIC X(10) VALUE IS "January".
      02 FILLER PIC X(10) VALUE IS "February".
      02 FILLER PIC X(10) VALUE IS "March".
02 FILLER PIC X(10) VALUE IS “April”.
02 FILLER PIC X(10) VALUE IS “May”.
02 FILLER PIC X(10) VALUE IS “June”.
02 FILLER PIC X(10) VALUE IS “July”.
02 FILLER PIC X(10) VALUE IS “August”.
02 FILLER PIC X(10) VALUE IS “September”.
02 FILLER PIC X(10) VALUE IS “October”.
02 FILLER PIC X(10) VALUE IS “November”.
02 FILLER PIC X(10) VALUE IS “December”.

01 MONTH-NAME REDEFINES MONTH-TABLE.
02 MONTH PIC X(10) OCCURS 12 TIMES.

77 I PIC 99.

PROCEDURE DIVISION.
MAIN-PARA.
    PERFORM DISPLAY PARA VARYING I FROM 1 BY 1
         UNTIL I > 12.
    STOP RUN.
DISPLAY-PARA.
    DISPLAY MONTH (I).

In this example, DISPLAY statement displays names of 12 months.

In the second method storing of the data is through the PROCEDURE DIVISION. Here the values may be obtained from a file or from some calculations or from terminals. The following example illustrates this method.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 PGDCA-RESULT.
    05 STU-ID OCCURS 50 TIMES.
        10 SUB-CODE PIC X(5).
        10 SUB-MARKSPIC 99.
77 I PIC 99.
Suppose, the values are stored in a file with the name PGDCA-FILE, whose FD entry and record descriptions are as shown below:

FILE SECTION.
FD PGDCA-FILE.
01 PGDCA-REC.
   05 TITLE-CODE PIC X(5).
   05 TITLE-MARKS PIC 99.

The PROCEDURE DIVISION statements to store the values of table from the said file written as:-

PROCEDURE DIVISION.
READ-PARA:
   MOVE 1 TO I.
   READ PARA.
   READ PGDCA-RECORD AT END GO TO END-OF-STORING.
   MOVE TITLE-CODE TO SUB-CODE (I).
   MOVE TITLE-MARKS TO SUB-MARKS (I).
   ADD 1 TO I.
   IF I NOT > 50 GOTO READ-PARA.
   STOP RUN.

The data name "I" has been used as subscript and it is described in the WORKING-STORAGE section.

1.2.3 USAGE IS INDEX CLAUSE

An INDEX data item is an elementary item, which is defined in the DATA DIVISION with the USAGE IS INDEX clause. It has the following syntax:
Rules:

- There must not be a picture clause with the index data item.
- If this clause is specified for a group item then it applies to all elementary data items of it, but remember that the group itself is not a data index item.
- The index item can be set by using SET verb.

Syntax of SET verb is given below:

```plaintext
SET index-name-1, [index-name-2, …] TO {integer-1/identifier-1/index-name-1}
```

You can increase or decrease values of an index by using SET verb as shown by the syntax below:

```plaintext
SET index-name-1, [index-name-2, …] {UP BY/ DOWN BY} {integer-1/identifier-1}
```

For example:

```plaintext
DATA DIVISION.
WORKING-STORAGE SECTION.
77 I USAGE IS INDEX.
77 J USAGE IS INDEX.
77 K USAGE IS INDEX.
.....
PROCEDURE DIVISION.
INDEX-PARA.
    SET I TO 5.
    SET J TO 1.
    SET K TO J.
```
SET I UP BY 1.
SET I DOWN BY 1.

Second way to specify index is with OCCURS clause that has been discussed in the next paragraph.

1.2.4 TABLE HANDLING WITH PERFORM VERB

1.2.4.1 TIMES option

The format for the PERFORM with TIMES option is

```
PERFORM  procedure-name-1 [THRU procedure-name-2]
{ identifier integer } TIMES
```

For example: - PERFORM activity-A 3 TIMES

Here the range of the procedure is controlled by the identifier or the integer and then control shift to very next statement. If the value of identifier or integer is zero then the procedure is not executed.

1.2.4.2 UNTIL option

The format for the PERFORM with UNTIL option is

```
PERFORM procedure-name-1 [THRU procedure-name-2]
UNTIL condition.
```

Here the range is executed until the predefined condition is not true.

1.2.4.3 VARYING option
PERFORM procedure-name-1 [THRU procedure-name-2]

{ identifier -1
  index-name }
FROM
{ identifier -2
  index-name-2 }

BY
{ identifier -3
  index-name-3 }
UNTIL Condition

Example: This example illustrate the use of index with OCCURS clause and manipulation by the PERFORM verb.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 STUDENT-RECORD.
   05 RECORD-TABLE OCCURS 20 TIMES INDEXED BY I.
      10 ROLL-NUMBER PIC 9999.
      10 REG-NO PIC 99AA9999.
      10 NAME PIC A(20).
      10 CLASS PIC X(10).

PROCEDURE DIVISION.
MAIN-PARA.
   PERFORM READ-PARA VARYING I FROM 1 BY 1
         UNTIL I > 20.
   PERFORM READ-PARA VARYING I FROM 1 BY 1
         UNTIL I > 20.
   STOP RUN.
READ-PARA.
   ACCEPT ROLL-NUMBER (I).
   ACCEPT REG-NO (I).
1.3 Summary

- A table is a collection of homogeneous items. It is declared by specifying the type, or structure, of a single data-item and the number of times the data-item (element) is repeated.

- A table is stored in memory as a contiguous block of bytes. As per the number of columns in a table these can be classified into mainly two categories: One-dimensional and Multi-dimensional table.

- In COBOL, OCCURS clause is used to define tables. OCCURS clause can also be used to define index items.

- The elements of the table can be referred in the PROCEDURE DIVISION by a special method known as subscripting. The subscript is enclosed in parentheses and follows the table name.

- The values to a table can be assigned in two different ways – through REDEFINES clause in DATA DIVISION and by using verbs in the PROCEDURE DIVISION.

- Index can be declared by two ways - one through OCCURS clause and second by using USAGE clause. Indexes can be manipulated by using SET verb.
Tables can easily be handled by using PERFORM verb.

1.4 Key words

Table, occurs, redefines, set, index, subscript, up, down.

1.5 Self-Assessment Questions (SAQ)

1. Discuss the PERFORM verb in table handling with all its options.
2. Write all the DATA DIVISION statements to define a table having 10 different courses and to initialize the table to contain the number of students enrolled limited by 50.
3. Differentiate the following:
   (i) Subscript and Index.
   (ii) Subscript and Index data item.
   (iii) Index and Index data item.
4. What is the significance of OCCUR clause in table handling? Give an example.
5. Write all the statements of DATA DIVISION to form a table consisting all the names of the months so that the names of the months are referenced by the subscript.
6. How can you initialize a table during compilation?
7. Discuss two different ways to declare an index.
8. Discuss two different ways to initialize a table.
9. Discuss the COBOL verb to manipulate index item.

1.6 References/Suggested Readings

- COBOL Programming by M. K. Roy and D. Dastidar; TMH
- Schaum’s outline series Programming with Structured COBOL; MGH
- Comprehensive COBOL, vol-I, Fundamentals of COBOL programming, 4/e by A.S. Philippakis and Leonard J. Kazmier; TMH
- Comprehensive COBOL, vol-II, Advanced COBOL programming, 4/e by A.S. Philippakis and Leonard J. Kazmier; TMH
• Structured COBOL: Fundamentals and style, 4/e by Welburn; TMH
• Computer Programming in COBOL by V. Rajaraman; PHI
• Fundamentals of Structured COBOL Programming by Carl Feingold; Galgotia Booksource.
I.0 Objectives

- To discuss the concept of index and different ways to define it.
- SET verb and its different formats will be discussed.
- Linear search operation by using SEARCH verb will be described.
- Binary search verb syntax and its use on sorted tables will be presented.

1.1 Introduction

In the last chapter, you learnt the concept of tables. You also learnt simple clauses and verbs associated with table handling. In this chapter you will learn more advanced clauses and verbs to handle tables. In COBOL, index is a data item, which can be used in place of the subscript. Searching a particular in the table is very common in business applications. To search a particular value in the table, COBOL provides two search techniques - linear search and binary search. The binary search is more efficient and fast technique as compare to linear search. But binary search can be applied only if table is sorted.

1.2 PRESENTATION OF CONTENTS

1.2.1 INDEXING BY clause

In COBOL, index is a data item, which can be used in place of subscript so that the machine language address calculation can be made more efficient. The index value is a displacement in the table, which is added in to the first address of the table to
generate the address of the desired data item in the table. An index can be defined by using USAGE clause as you learnt in the last chapter. The more elegant way to define an index is with OCCURS clause.

The format of the INDEXED BY phrase is as shown below:

```
OCCURS integer TIMES
[INDEXED BY index-name-1 [, index-name-2] …]
```

The following rules must be keep in mind by the programmer:

- The indexing must be equally distributed over the table that is if indexing is done over one level then it must be implemented over the all levels of the table.
- The index name should not be used with the subscript in combination.
- Indexes are valid only in their respective table.
- Indexes are manipulated only by the SET, SEARCH and PERFORM statements.
- The indexes name must be unique.
- You can use more than one index for one level.

Examples:

```
DATA DIVISION.
WORKING-STORGAE SECTION.
01 TWO-DIMENSIONAL-TABLE.
   05 BRANCHES OCCURS 20 TIMES
      INDEXED BY I K.
   10 MONTHLY-COMM PIC 9999V99 OCCURS 12 TIMES INDEXED BY J.

PROCEDURE DIVISION.
MAIN-PARA.
   PERFORM READ-PARA
```
VARYING I FROM 1 BY 1 UNTIL I > 20
AFTER J FROM 1 BY 1 UNTIL J > 12.
PERFORM DISPLAY-PARA
VARYING I FROM 1 BY 1 UNTIL I > 20
AFTER J FROM 1 BY 1 UNTIL J > 12.
PERFORM STOP-PARA.
READ-PARA.
DISPLAY “ENTER COMMISSION: “.
ACCEPT MONTHLY-COMM (I, J);
DISPLAY –PARA.
STOP-PARA.
STOP RUN.

Difference between Index and Subscript::

Subscript is a data item that refers to the number of the table entry you want to reference. The value of the subscript can be changed by PERFORM …VARYING, MOVE, ADD, SUBTRACT.

Index can be defined by INDEX BY clause on the OCCURS level. Indexes are more efficient as compared to subscript. Computer actually uses displacement values to actually access indexed table entries. The displacement values used by an index depend upon the number of bytes in each table entry.

Because an index refers to a displacement and not just an occurrence value , its contents can not be modified with MOVE ADD or SUBTRACT like a subscript can.

Index can be modified either by SET verb or by PERFORM…VARYING.

1.2.2 SET VERB
Index can be manipulated by using SET verb. The SET verb can be used to increase/decreases the values of the indexes. The SET verb can have many formats.

One of the formats of SET verb allows you to set a particular value to a number of indexes so that different index names are set to the same value. The syntax of this format is given below:

**Syntax-1**

```
SET index-name-1 [ , index-name-2] … TO { identifier -1 integer-1 index-data-item index-name-3 }
```

Integer value can be positive only.

For examples:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 A-TABLE.
  05 MARKS PIC 99 OCCURS 20 TIMES
      INDEXED BY F1, F2, F3.
PROCEDURE DIVISION.
TEST-para.
  SET F1, F2, F3 TO 5.
  MOVE 78 TO MARKS (F1).
  DISPLAY MARKS (F2). //displays 78
  …..
  STOP RUN.
```

(ii) Current value of an index can be stored in one or more identifiers. The syntax for
this format of SET verb is as given below:

**Syntax-2:**

```plaintext
SET identifier-1 [, identifier-3]... TO index-name-1
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```plaintext
01 A-TABLE.
  05 MARKS PIC 99 OCCURS 20 TIMES
      INDEXED BY F1, F2, F3.

77 TEMP1 PIC 99.
77 TEMP1 PIC 99.
```

PROCEDURE DIVISION.

```plaintext
TEST-PARA.

SET F1, F2, F3 TO 5.
SET TEMP1, TEMP2, TO F3.
// identifiers TEMP1 SNF TEMP2 are set to 5
```

(iii) When it is necessary to increment or decrement one or more indexes by a positive integer value then the following format of the SET verb can be used:

**Syntax-3**

```plaintext
SET index-name-1 [, index-name-6]... \{ UP BY \} \{ \{ DOWN BY \} identifier-4 integer-2 \}
```

In this format, ‘UP BY’ phrase is used to increment the index value by integer-2/identifier-4 and ‘DOWN BY’ phrase is used to decrement the value of the index by integer-2/identifier-4.
Examples:

Let A1 and X1 are two indexes defined in the data division. A1 is initialized with 10 and X1 is initialized with 5 by the SET verb as shown below:

```
SET A1 TO 10.
SET X1 TO 5.
```

Now you want to increment A1 by 3. it can be done by writing the statement as given below:

```
SET A1 UP BY 3.
```

Now A1 contains the value 13. You can decrease the value of an index by specified value as:

```
SET A1 DOWN BY 4.
```

Now A1 contains the value 9. You can also decrease the value of an index by another index value as:

```
SET A1 DOWN BY X1.
```

Now A1 contains the value 4.

**1.2.3 SEARCH VERB**

Whenever your target is to search an element from a one-dimensional table then the SEARCH verb is an excellent option for you. Searching an element means whether the desired element (element which satisfy the predefined condition) is present in the table or not. COBOL language provides SEARCH verb in two different formats – 1)
for linear search and 2) binary search. The syntax for the linear SEARCH verb is given below:

**Syntax of Linear SEARCH verb:**

```
SEARCH Identifier-1 [ VARYING { identifier-2 index-name-1 } ]

; AT END imperative-statement-

; WHEN Condition-1 { Imperative-statement-2 NEXT SENTENCE }

; WHEN Condition-2 { Imperative-statement-3 NEXT SENTENCE }
```

**Syntax Rules for SEARCH verb:**

1. Identifier-1 (Table-Name) must identify a data-item in the table hierarchy with both OCCURS and INDEXED BY clauses. The index specified in the INDEXED BY clause of Table-name is the controlling index of the SEARCH.

2. The index must have some initial value before execution of a SEARCH verb. When the search terminates without finding the particular element then the index of the table has no predictable value.

3. The SEARCH can only be used if the table to be searched has an index item associated with it. An index item is associated with a table by using the INDEXED BY phrase in the table declaration. The index
item is known as the table index. The table index is the subscript, which the SEARCH uses to access the table.

Working of SEARCH verb:

- The SEARCH searches a table sequentially starting at the element pointed to by the table index.

- The starting value of the table index is under the control of the programmer. The programmer must ensure that, when the SEARCH executes, the table index points to some element in the table (for instance, it cannot have a value of 0 or be greater than the size of the table).

- The VARYING phrase is only required when we require data-item to mirror the values of the table index. When the VARYING phrase is used, and the associated data-item is not the table index, then the data-item is varied along with the index.

- The AT END phrase allows the programmer to specify an action to be taken if the searched for item is not found in the table.

- When the AT END is specified, and the index is incremented beyond the highest legal occurrence for the table (i.e. the item has not been found), then the statements following the AT END will be executed and the SEARCH will terminate.

- The conditions attached to the SEARCH are evaluated in turn and as soon as one is true the statements following the WHEN phrase are executed and the SEARCH ends.
The flowchart given in Fig 10.1 explains the working of the SEARCH verb.

**Fig 10.1 Flowchart for Sequential Search Verb**

Example of linear binary search:

```plaintext
... DATA DIVISION. WORKING-STORAGE SECTION. 01 MONTHS-TABLE. 02 FILLER PIC X(10) VALUE IS "January". 02 FILLER PIC X(10) VALUE IS "February".
```
This program takes month’s name as input and searches that name in the table. If name is found then displays number equivalent of that month name. If name is not found then it displays, month’s name is ILLEGAL.

1.2.4 Binary search
The SEARCH verb discussed in the previous section is linear search, which is applicable to an unsorted table or sorted table both. Linear search is slow specially when table size is very large. In the table of n elements linear search requires n number of comparisons.

If the values of the table are sorted, then there is another approach called Binary Search for fast searching the elements in the table. In binary search, first given element is matched with the middle element of the table. If match occurs search is successful else the given element is in the first half or in the later half. This procedure is repeated on halve expected to contain the given element till the element is found or table is exhausted. COBOL supports the binary search directly through the SEARCH verb. The syntax of the binary search verb is given below:

Syntax of binary SEARCH verb:

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

In case of binary search, SET verb is not required to initialize the index, but the OCCURS clause of the table must include an ASCENDING/DESCENDING KEY. On the basis of this key, the field of the table is decided on which sorting of the table is done. Syntax of the OCCURS clause is given below:

```
OCCURS integer TIMES
[(ASCENDING/DESCENDING) KEY IS data-name-1 [, data-name-2] ]
[INDEXED BY index-1 [, index-2] ]
```

When the ASCENDING/DESCENDING option is used it is assumed that at the time of search table is arranged either in ascending or in descending
order, which ever is mentioned. If more than one data name is used the first is major key, second one is the next major key and so on.

Example:

... 

DATA DIVISION.
WORKING-STORAGAE SECTION.
  01 SAVING-BANK-ACCOUNT.
     05 SB-TABLE OCCURS 100 TIMES
        ASCENDING KEY IS AC-NO
        INDEXED BY I.
     10 AC-NO PIC 999999.
     10 NAME PIC X(20).
     10 BALANCE PIC 9(8).99.
    77 ACCOUNT-NO PIC 999999.
PROCEDURE DIVISION.
READ-PARA.
  DISPLAY “Enter your Account Number:”.
  ACCEPT ACCOUNT-NO.
SEARCH-PARA.
  SEARCH ALL SB-TABLE
  AT END DISPLAY “ILLEGAL ACCOUNT”
  WHEN ACCOUNT-NO = AC-NO (I)
    DISPLAY “ACCOUNT NUMBER = “, AC-NO.
    DISPLAY “NAME = “, NAME.
    DISPLAY “BALANCE = “, BALANCE.
STOP-PARA.
STOP RUN.

1.3 Summary

- Subscript is a data item that refers to the number of the table entry you want to reference. The value of the subscript can be changed by PERFORM
The index value is a displacement within a table, which is added in to the first address of the table to generate the desired data item from the table. Index can be defined by INDEX BY clause on the OCCURS level. Indexes are more efficient as compared to subscript. Computer actually uses displacement values to actually access indexed table entries. The displacement values used by an index depend upon the number of bytes in each table entry.

Because an index refers to a displacement and not just an occurrence value, its contents cannot be modified with MOVE ADD or SUBTRACT like a subscript can. Index can be modified either by SET verb or by PERFORM...VARYING verb.

The index name cannot be used with the subscript in the combinations. Indexes are valid only in their respective table.

COBOL supports two types of search operations on tables – linear search and binary search. Linear search can be applied on both sorted and unsorted tables while binary search can be applied on sorted tables only. Binary search is much faster than linear search.

To apply SEARCH verb on tables, the table should be associated with an index item(s). The index item is known as the table index. The table index is used by the SEARCH verb to access the table.

A table can be sorted on more than one key. The most important key on which sorting is done is known as major key and the key with least importance is known as minor key.
1.4 Key words

Index, search, perform, set, binary, linear, occurs, key

1.5 Self-Assessment Questions (SAQ)

1. What is index? How is it defined in COBOL? Explain with example.
2. Differentiate between index and subscript.
3. What are different types of search supported by COBOL?
4. Explain the meaning of the following COBOL verbs with examples:
   (a) SET TO and SET UP BY or SET DOWN BY.
   (b) SEARCH with its options.
5. Explain the method of searching in an unsorted table of COBOL.
6. Write a COBOL program for linear search on STUDENT table.
7. Write a COBOL program for binary search on EMPLOYEE table where major key is EMPLOYEE-ID.

1.6 References/Suggested Readings

- COBOL Programming by M. K.Roy and D. Dastidar ; TMH
- Schaum’s outline series Programming with Structured COBOL ; MGH
- Comprehensive COBOL, vol-I, Fundamentals of COBOL programming, 4/e by A.S.Philippakis and Leonard J. Kazmier ; TMH
- Comprehensive COBOL, vol-II, Advanced COBOL programming, 4/e by A.S.Philippakis and Leonard J. Kazmier ; TMH
- Structured COBOL: Fundamentals and style, 4/e by Welburn ; TMH
- Computer Programming in COBOL by V.Rajaraman; PHI
• Fundamentals of Structured COBOL Programming by Carl Feingold; Galgotia Booksource.
1.0 Objectives

- To introduce structured programming techniques.
- To discuss Top-down approach and Bottom-up approach
- To introduce GO-TO less programming
- To describe single entry single exit constructs.

1.1 Introduction

The programming design refers to a process to describe the logic of a problem in a non-programming language such as flow charts, decision tables, structured English etc. Through the programming design you can fragment a program into logical modules so that the problem can be handled easily. The programming design must follow a well-defined pattern. Structured Programming is a strategy that encompasses a number of methodologies to achieve certain objectives. E.W. Dijkstra first introduced the concept of structured programming. He introduced this concept with a number of objectives in his mind such as ease of coding, program development by modules, less development time, less error rate, more readability and more independability. This chapter will discuss the objectives and methodologies of structured programming.

1.2 PRESENTATION OF CONTENTS

1.2.1 Structured Programming
The structured programming design partitions a program into smaller and independent modules. These modules are arranged in a hierarchy in a top down manner with increasing details. Thus a structured design attempts to minimize complexity of a problem and make the problem manageable by subdividing it into segments of smaller size.

The advantages offered by structured programming are:

- Program has more nearly self-documentation.
- Program is easy to modify.
- Maintenance of the program becomes easier.
- A large program can be handled with ease by using modular approach.
- Number of errors is reduced drastically.

The basic objectives of the structured programming design are: Modular Programming, Top-down/Bottom-up programming and Structured flow of control.

1.2.2 Modular Approach

Here a program is decomposed into a number of well-defined subprograms (modules), which follows all the characteristics of a program. That means a module is a portion of a program that also satisfies the definition of a program. A module can further be decomposed into subordinate modules or conversely subordinate modules can further be combined to form a superior module. Superior module reuses the codes of subordinate modules; it does not include the physical copies of the code of subordinate modules. After successful designing of modules, these are integrated to obtain a complete program from them. That means a superior module through a reference can call a subordinate module from any part of the program without bothering its location in the program.
The subordinate module is called as called module and superior module is called as calling module. Fig 11 illustrates the concepts of calling module and called module. In Fig 11.1, A is a superior module while B and C are subordinate modules. Module A is a calling module and modules B and C are called modules.

![Fig 11.1 Modular Design](image)

The advantage of modular programming will depend upon the effectiveness of the design of a module. Generally, one module should be designed for one function in the system. This leads to easier modification of the program. If there is any change in that function then that function can be identified and modified easily without affecting the rest of the program.

To implement the modular programming, a language should support the facilities for definition and calling of modules. In COBOL, one section or one paragraph in procedure division can be equivalent to a module. To call a paragraph or section, COBOL provides PERFORM statement. PERFORM statement transfers the control to the called paragraph or section (module). After the execution of the called module, control is transferred back to the next sequential statement in the calling module.

### 1.2.3 Top Down/Bottom Up Approach

The modular programming as discussed above consists of a hierarchical structure. This hierarchical structure can be perceived in two different ways - top-down and bottom-up.
The top-down approach starts with the specification of whole problem to be solved and then breaks it down progressively into smaller and lesser complex sub-problems. The decomposition of the problem progresses with increasing level of details. Each sub-problem at each level is organized into modules. In the top down approach, calling module is always designed before its called modules. The broad functions of called modules are considered in the calling module. The details of these functions are not considered until the calling modules are taken up for design. Therefore, top down approach is a successive refinement approach. The process of refinement of functions is continued until the lowest level module is designed.
For example, an author wants to write a book. In the top down approach of writing a book, the author first decides the title of the book. Then the chapters of the book will be planned. After deciding the chapters, author will take up chapters for writing. In the chapter, topics to be covered will be decided. Then these topics will actually be written down. Fig 11.2 illustrates the writing of a book.

As it is obvious from Fig 11.2, top-down design can be viewed as a hierarchical structure. Each box in the figure represents a module.

Another example of top-down design, consider the design of an interactive system. The top-level program will be the part of the system, which ties together the key system components. One of these components might be the part of the system, which reads a command; another component might evaluate the command just entered. Still another component might be the part of the system, which displays the results of executing the command just entered. The overall structure of such a system is shown from the top down in the following diagram:

The top down approach offers the following advantages:
• It provides a natural way to solve a problem.
• Modules at lower level can be designed without knowing the details at the higher level.
• Different programmers can develop modules independently.

In summary, we can say that this approach demands careful planning and coordination and a clear vision at the main objective of the program. The interface between modules can be defined before the functions are actually coded. The superior module must be designed before the designing of the subordinate module, so that later can be called by the first when it is needed.

On the other hand, bottom-up approach is just opposite of top-down. In this approach, modules at the lowest lever are either already available or designed first. Then these modules are combined to form the higher-level module. The process of combing the modules is continued till the entire program is designed. In component based software engineering, bottom-up approach is used to develop software. Software components already exist in the component repository. First the developer searches the components from the repository and then they are integrated to realize the software under development. The main disadvantage of this technique is dependence on readymade modules. Many times, the situation occurs that the desired modules are not available.

A bottom-up development approach directly addresses the need for a rapid solution of the business problem, at low cost and low risk. A typical requirement is to develop an operational data mart for a specific business area in 90 days, and develop subsequent data marts in 60 to 90 days each. The bottom-up approach meets these requirements without compromising the technical integrity of the data warehousing solution. Data marts are constructed within a long-term enterprise data warehousing architecture, and the development effort is strictly controlled through the use of logical data modeling techniques and integration of all components of the architecture with central metadata.
Top-down approach is more popular among the programmers due to its parallel development, proper connectivity with the subordinate modules and consideration of the major objective of the program in the beginning of the design.

**1.2.4 STRUCTURES USED IN STRUCTURED PROGRAMMING**

Structured programming is a technique for organizing and coding computer programs in which a hierarchy of modules is used, each having a single entry and a single exit point, and in which control is passed downward through the structure without unconditional branches to higher levels of the structure.

The Fundamental Principle of Structured Programming is that at all times and under all circumstances, the programmer must keep the program within his intellectual grasp. The well-known methods for achieving this can be briefly summarized as follows: 1) top-down design and construction, 2) limited control structures, and 3) limited scope of data structures.

The Step-by-Step Method helps you create the "right" systems by uncovering their true needs, but it doesn't ensure that the resulting systems are reliable and maintainable. "Structured programming" is a discipline that helps you avoid convoluted logic in your programs, but that doesn't scale up to large systems. What is needed is a way to treat software as "components," just the way engineers think of silicon chips as black boxes whose insides can be largely ignored.

The structured programming uses the following four forms of constructs:

a) Sequence
b) Decision
c) Iteration
d) Case
1.2.4.1 Sequence Structure

In this structure the sequential execution of instructions or imperative statements, one after the other i.e. once the control enters the paragraph, then it goes out only after completion of all the statements of it. Here the physical ordering of the statements must follows the logical ordering. The sequence is represented by one statement after another as shown in the diagram. There is only a single entry and single exit.

![Sequence Program Diagram](image)

1.2.4.2 Decision Structure

In this structure depending upon the decision condition value (True/False or Yes/No) only one of the two branches is selected. Decision is a selection between two actions based upon a condition, which is always either true/false or yes/no known as predicate (In Fig 11.4 and Fig 11.5 decision is represented by P in decision box, S, A, B represents a statement or a group of statements.).

![Decision Structure Diagram](image)
The decision constructs used in programming languages is IF statement. There are two different forms of IF as shown in Fig 11.4 and 11.5.

In Fig 11.4, if P is true then S is executed and then the statement next to the IF statement is executed. If P is false then the statement next to the IF statement is executed.

In Fig 11.5, if P is true then B is executed and then the statement next to the IF statement is executed else A is executed and then the statement next to the IF statement is executed.

1.2.4.3 Iteration Structure

In this structure, a process or a group of processes is to be repeated for a predefined number of times to obtain the desired results. There are two types of iterations:

- Pre-test iteration
- Post-test iteration
In case of Pre-test iteration, first condition is checked and then iteration takes place, e.g. \textit{Do} \ldots \textit{while}. On the other side in case of Post-test iteration condition is checked after the iteration has taken place, e.g. \textit{Do} \ldots \textit{until}.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{Fig11_6.png}
\caption{Pre-test iteration}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{Fig11_7.png}
\caption{Post-test iteration}
\end{figure}

\subsection*{1.2.4.4 CASE Structure}

The case structure is used when there is a set of multiple alternative paths (branches) in the program logic. Therefore, some time it is called as multi-branch decision structure. Decision structure as discussed above, is a special
type of case having two paths only. In case structure we follow one path out of many paths available depending upon case value. Fig 11.8 shows a typical CASE structure.

![CASE Structure Diagram](image)

**Fig 11.8** CASE Structure

### 1.2.5 GO-TO-LESS PROGRAMMING

GO-TO-Less programming is also associated with the structured programming. Writing a program without using GO TO instructions, an important rule in structured programming. A GO TO instruction points to a different part of the program without a guarantee of returning. Instead of using GO TO's, structures called "subroutines" or "functions" are used, which automatically return to the next instruction after the calling instruction when completed.

Nearly six years after publication of Dijkstra's letter, the subject of GOTO-less programming still stirs considerable controversy. Dijkstra and his supporters claim that the GOTO statement leads to difficulty in debugging, modifying, understanding and proving programs. GOTO advocates argue that this statement, used correctly, need not lead to problems, and that it provides a natural straightforward solution to common programming procedures.
A good program must have a set of sequence of statements without skipping any statements. The ability of sequencing or the top-down approach of programming is useful because these are very near to the human behavior of problem solving.

The quality of programmers is a decreasing function of the density of GO TO statements in the programs they produce. More recently it has been discovered why the use of the GO TO statement has such disastrous effects, and in my opinion the GO TO statement should be abolished from all "higher level" programming languages.

Although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to accomplish the desired effect; it is this process that in its dynamic behavior has to satisfy the desired specifications. Yet, once the program has been made, the "making" of the corresponding process is delegated to the machine.

Our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.

The GO TO statement as it stands is just too primitive; it is too much an invitation to make a mess of one's program. "Like the conditional, one entry one exit structures mirror the dynamic structure of a program more clearly than GO TO statements and these eliminate the need for introducing a large number of labels in the program."

1.2.6 Structured Programming in COBOL
COBOL supports all features to write structured program or GO-TO less programs. While writing COBOL programs following points can be followed.

- Do GO-TO-LESS programming.
- A single programming module per page for better modular size.
- Single entry and single exit of a module.
- Data names must be data related.
- Use of Minimum number of comments.
- Use of restricted number of statement types.
- Nested functions must be carefully used.

1.3 Summary

- Structured programming is a way to design, write and test a program using interdependent sections (modules). Structured programming uses mainly three basic structures - sequence, decision and iteration.

- Structured programming can be seen as subset or subdiscipline of procedural programming. A structured program is easier to understand as compare to other methods of designing programs.

- *Top-down* and *bottom-up* and GO-TO-LESS programming is associated with structured programming.

- The top-down approach starts with the specification of whole problem to be solved and then breaks it down progressively into smaller and lesser complex sub-problems. The decomposition of the problem progresses with increasing level of details. Each sub-problem at each level is organized into modules.

- In the top down approach, calling module is always designed before its called modules. The broad functions of called modules are considered
in the calling module. The details of these functions are not considered until the calling modules are taken up for design.

- Therefore, top down approach is a successive refinement approach. The process of refinement of functions is continued until the lowest level module is designed.

- Writing a program without using GO TO instructions, an important rule in structured programming. A GO TO instruction points to a different part of the program without a guarantee of returning. Instead of using GO TO's, structures called "subroutines" or "functions" are used, which automatically return to the next instruction after the calling instruction when completed.

- A bottom-up development approach directly addresses the need for a rapid solution of the business problem, at low cost and low risk.

- Single programming module per page, single entry and single exit, data related data names, minimum number of comments, restricted number of statement types, nesting with care are some of the points, which can be kept in mind while writing programs in COBOL.

1.4 Key Words

Structured Programming, Readability, Dependability, Top-down/Bottom-up, module, called-module

1.5 Self-Assessment Question (SAQ)

1. List and briefly explain the characteristics of a good program.
2. Define structured programming.
3. What are the advantages of a structured program?
4. What are the objectives of the structured programming?
5. What do you mean by the iteration control structure? Discuss its implementation in COBOL?

6. What do you mean by the decision control structure and its implementation in COBOL?

7. Explain top-down approach of design. Discuss its advantages.

8. Explain bottom-up approach of design. Discuss its advantages.

9. write a short note on GO-To less programming.

1.6 Reference/Suggested Readings:

1. COBOL Programming by M.K.Roy and D..Dastidar ; TMH
2. Schaum’s outline series Programming with Structured COBOL ; MGH
4. Comprehensive COBOL, vol-II , Advanced COBOL programming, 4/e by A.S.Philippakis and Leonard J. Kazmier ; TMH
5. Structured COBOL: Fundamentals and style, 4/e by Welburn ; TMH
6. Computer Programming in COBOL by V.Rajaraman; PHI
7. Fundamentals of Structured COBOL Programming by Carl Feingold; Galgotia Booksource.
LESSON 12

FILES IN COBOL

1.0 Objectives

- To understand concepts and terminology like field, record, file, record buffer etc.
- To know different types of organizations of files
- To study the COBOL file description FD clause along with its various options.
- To study the COBOL verbs relating to file operations such as OPEN, CLOSE, READ, WRITE AND REWRITE etc.

1.1 Introduction

Here our main objective is how to create and read a tape or disk file. A magnetic tape can have only a sequential access; on the other hand disk files can have a number of access methods. The file organization means the method in which data records are arranged on a file storage medium for data manipulations and computations. The different types of file accessing methods are sequential, relative and indexed. In case of sequential method, records are accessed from the file one after the other, in case of relative method, each record has an identifier through which it can directly accessed from the file and in case of index method the records are associated with the index number through which the records are directly accessed from the file. The IOCS (input-output
control system) is responsible for the file handling tasks during the access of the records from the file.

1.2 PRESENTATION OF CONTENTS

1.2.1 CONCEPTS AND TERMINOLOGY OF FILES

In a COBOL program, a file is a collection of related units of information within a data category. A file might contain all the information (related units of information) about customers (a data category) for a company. This usually is called a data file or a logical file.

Within a data file, the information about one unit is called a record. If a data file contains the information pertaining to all customers, for example, the information about one customer is a record.

A field or data field is one piece of data contained in a record. COBOL data files are organized as one or more records containing the same fields in each record. For example, a record for a personal phone book might contain fields for a last name, a first name, and a phone number. These fields would exist in each individual record.

For the file to exist, there must be a physical file on the disk. When the bytes in this file are arranged logically so that a COBOL program can access the information, it becomes a data file to a COBOL program.

The use of storing devices determines the method of data accessing. Mainly there are two devices for data recording one is magnetic tape and the other one is magnetic disk. The magnetic tapes support only
the sequential method of accessing the data. The magnetic tapes are processing on the magnetic tape unit. There are two reels for the processing of the magnetic tape. One is known as machine reel used for storing that portion of the tape, which has already been processed and other, is file reel contains the tape to be read or written on. The tape is passing through a read/write head for the processing, as similar to a tape-recorder at our home.

The read/write speed of a magnetic device depends upon (i) Recording density
Of the magnetic tape (ii) Linear speed of the tape drive.

Therefore a tape

- Recoding density = 1000 bpi.
- Linear speed = 100 ips.
- Read/write speed = Record density x Linear speed
  = 1000 x 100 = 1,00,000 bps.

The magnetic disk is physically similar to a phonograph record. Here data is recorded on tracks having the capacity to accommodate thousands of characters. Every track having the same capacity, either it is placed as the outer one or inner one on the disk. The similar capacity of the racks is achieved by adopting the different packing density of the tracks, Therefore in a disk having N tracks, the inner most track (track0) having the highest packing density and the outer most (trackN-1) having the minimum.

1.2.1.2 File Parameters
Here are some of the important parameters of the files those are important for the programmers point of view.

1) Record Size  
2) Block Size  
3) Buffer  
4) Label

1.2.1.2.1 Record Size

The size of the record is directly associated with the storage media, which is controlled by the programmer through the field size declaration. The total sizes of the fields in a record are the algebraic sum of their fields, with minimum and maximum limits of the record. The record can be fixed or variable size, if the record is of variable size, then the size of each record of the file is stored with it (first four characters are used for the length of the record and remaining are for the data values).

1.2.1.2.2 Block Size

Block is a number of consecutive records from the storage media, through which file handling becomes easier. Some time block is also known as “physical- record” and the number of records in a block is known as “blocking –factor”. On the other side the records defined in the program are known as “logical-record”. By using the blocking during the accessing of the data from the storage media we can reduce the input-output time and increase the storage utilization factor. There must be proper trade-off for the block size, so that accessing must optimize.
1.2.1.2.3 Buffer

Now a day’s data-channels are used in the systems, so that the CPU oriented tasks and input-output oriented tasks can handle simultaneously. Hence the IOCS require more than one buffer for smooth functioning of the system. We can use a number of buffers to increase the performance of the system but there should be an upper bound.

1.2.1.2.4 Label

Every block must be preceded and followed by records known as header and the trailer respectively; these are helpful for the correct file handling by the IOCS. File-title is the main information stored in the header used for the file identification, file-title is just a physical name of the file used by the IOCS, and so that proper file is assigned to the program. In normal practice two files with the same titles cannot be resides in the same storage media. Now days the concept of generation numbers is used in place of file-title to avoid the ambiguity in case of same file-title for two files in a same storage media.

1.2.3 File Organizations in COBOL
Files can be organized in many different ways. Using only COBOL syntax, COBOL programs can create, update and read files of four different organizations:

- **Line sequential**

  Line Sequential files are a special type of sequential file. They correspond to simple text files as produced by the standard editor provided with your operating system.

- **Record sequential**

  Sequential files are the simplest form of COBOL file. Records are placed in the file in the order they are written, and can only be read back in the same order.

- **Relative Files**

  Every record in a relative file can be accessed directly without having to read through any other records. Each record is identified by a unique ordinal number both when it is written and when it is read back.

- **Indexed**

  Indexed files are the most complex form of COBOL file, which can be handled directly by COBOL syntax. A unique user-defined key when written identifies records in an indexed file. Each record can contain any number of user-defined keys, which can be used to read the record, either directly or in key sequence.

### 1.2.3.1 Sequential File organization

A sequential file is a file in which the records can only be accessed sequentially. Here the records are stored in the serial order and read in the same order in which they reside on the storage device. Records are
always added to the end of the file. COBOL supports two different types of sequential files:

- Line Sequential
- Record Sequential

### 1.2.3.1.1 Line Sequential Files

In line sequential files, each record in the file is separated from the next by a record delimiter. On DOS, Windows and OS/2 this is a carriage return (x"0D") and a line feed (x"0A") character. On UNIX it is just the line feed (x"0A") character. These characters are inserted after the last non-space character in each record so line sequential files always contain variable-length records.

Report files are line sequential, since most PC printers require the carriage return and/or line feed characters at the end of each record. Most PC editors produce line sequential files, and these files can therefore be edited with almost any PC editor. The primary use of line sequential files is for display-only data. Line sequential files are also known as text files, or flat ASCII files.

When you declare a file as line sequential in COBOL, you do so through the SELECT clause.

**For Example:** Creating a line sequential file

```cobol
ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT LINESEQ-FILE
      ASSIGN TO "DATAFILE.TXT"
    ORGANIZATION IS LINE SEQUENTIAL.

... DATA DIVISION.
  FILE SECTION.
  FD LINESEQ-FILE
```
1.2.3.1.2 Record Sequential Files

Record sequential files are simply called sequential files, since record sequential is the default for a sequential file. Records in a record sequential file can be either fixed or variable in length.

Variable-length records save disk space. There are many applications that can benefit from the use of variable-length records. A common example is where your application generates many small records, with occasional large ones. If you make the record length as long as the largest record, you waste a lot of disk space. The way to prevent this waste is to use variable-length records.

When you declare a file as record sequential in COBOL, you do so through the SELECT clause.

For Example: Creating a record sequential file with fixed-length records.

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.

FILE-CONTROL.
SELECT RECSEQ-FILE
    ASSIGN TO "STUDENT.DAT"
    ORGANIZATION IS RECORD SEQUENTIAL.
....
DATA DIVISION.
```
FILE SECTION.
FD RECSEQ-FILE
   RECORD CONTAINS 80 CHARACTERS.
01 FILE-REC        PIC X(80).

In place of the ORGANIZATION clause above, you could use:

ORGANIZATION IS SEQUENTIAL.

Or, you could simply omit the ORGANIZATION clause, as record sequential is
the default file organization (if the SEQUENTIAL directive is not set).

**For Example:** Creating a record sequential file with variable-length records.

    ENVIRONMENT DIVISION.
    INPUT-OUTPUT SECTION.
    FILE CONTROL.
       SELECT IN-FILE
          ASSIGN TO "STUDENT.DAT"
          ORGANIZATION IS SEQUENTIAL.
          ...
    DATA DIVISION.
    FILE SECTION.
    FD IN-FILE
       RECORDING MODE IS V
       RECORD VARYING FROM 3 TO 80 CHARACTERS.
01 IN-REC        PIC X
     OCCURS 3 TO 80 TIMES
     DEPENDING ON WS-RECORD-LENGTH.
    WORKING-STORAGE SECTION.
01 WS-RECORD-LENGTH    PIC 99.
1.2.3.2 Indexed File Organization

Whenever you need to provide users with many different views of a file, you need indexed files. In your programs, this implies the need for random access, keyed on one or more fields in the records.

In indexed file, an index is created on records so that the records can be accessed directly without referring them in a sequence. The indexed organization having the best feature of other two file organizations that is it permits sequential storing but supports random processing of records. The indexed file not only stores the data records but also stores the index that has the location information of records.

Indexed file access enables you to access records either randomly or sequentially, using one or more key fields in the individual records. Key comparisons are made on a byte-by-byte basis from right to left using the ASCII collating sequence.

COBOL indexed files are actually made up of two physical files: a data file and an index file. The index file is created automatically, and has an extension of .IDX; the data file can have any other extension, although .DAT is very common. Records in indexed files can be either fixed or variable in length.

**For Example:** Creating an indexed file with fixed-length 80-byte records keyed on the first five bytes of each record:

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT IN-FILE ASSIGN TO "STUDENT.DAT"
       ORGANIZATION IS INDEXED
       ACCESS MODE IS DYNAMIC
```
RECORD KEY IS KEY-FIELD.

....

DATA DIVISION.
FILE SECTION.
FD IN-FILE
    RECORD CONTAINS 80 CHARACTERS.
01 IN-RECORD.
    05 KEY-FIELD           PIC X(5).
    05 REST-FIELD         PIC X(75).

For Example: Creating an indexed file with variable-length records, varying in length from 5 to 80 bytes. The keys defined for the file must all lie in the fixed part of the record.

IDENTIFICATION DIVISION.
PROGRAM-ID. FILESDEMO.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MYFILE ASSIGN TO "FILE.DAT"
    ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS KEY-FIELD.

DATA DIVISION.
FILE SECTION.
FD MYFILE
    RECORD IS VARYING IN SIZE
    FROM 5 TO 80 CHARACTERS
    DEPENDING ON WS-RECORD-COUNT.
01 FD-RECORD.
    05 KEY-FIELD               PIC X(5).
    05 REST-DATA             PIC X(75).
1.2.3.3 Relative File Organization

In relative file organizations, each record is referred by a unique identifier, which is a relative displacement reference in the file e.g. if a file consists of 10 records then the first record’s relative key value is 1 and the last record relative key is 10.

With relative file organization, you can access records sequentially or randomly. For sequential access, you simply do a sequential READ to get the next record in the file. For random access, you specify the ordinal number of the record in the file.

Relative files have a fixed-length file format. You can declare that you want the records to have a recording mode of "variable" but even if you do this, the system assumes the maximum record length for all WRITE statements to the file, and pads the unused character positions. So, when you are in a situation where you have a lot to gain by using variable-length records, you should avoid relative files because they are always fixed format.

Relative files have the fastest access time of all the file types used by this COBOL system so, if speed of access is the most important consideration, you should consider using relative files.

With relative files, you can have numeric keys, but you cannot key on fields. If you need to access data randomly based on certain fields, you must use indexed files.

For Example: Creating a relative file with a record length of 80 characters.
IDENTIFICATION DIVISION.
PROGRAM-ID. FILESDEMO.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT RELFILE ASSIGN TO "MYFILE.DAT"
    ORGANIZATION IS RELATIVE
    ACCESS MODE IS RANDOM
    RELATIVE KEY IS REL-KEY.

DATA DIVISION.
FILE SECTION.
FD RELFILE
    RECORD CONTAINS 80 CHARACTERS.
  01 REL-RECORD PIC X(80).

WORKING-STORAGE SECTION.
  01 REL-KEY PIC 9(8) COMP.

The relative key field is REL-KEY. When you are randomly accessing this file, there is no KEY IS field on the READ statement. The number in REL-KEY determines which record is read. (For sequential access, a simple READ statement gets the next record.)

1.2.4 FILE CONTROL SPECIFICATION

The file control specifications are used for the smooth handling of a COBOL file. SELECT [OPTIONAL ] file-name ASSIGN TO hardware-name

    [RESERVE integer {AREA/S}]
1.2.4.1 RESERVE CLAUSE

It specifies the numbers (integer-1) of buffers to be used for the file handling. If the integer value is 1 i.e. there is one area used as buffer. By default there are two buffers in the system.

1.2.4.2 ORGANIZATION/ACCESS CLAUSE

In the above format file is organized in the sequential manner and its access is also sequential one, both these clauses are optional and the by default the organization and access both are sequential.

1.2.4.3 FILE STATUS CLAUSE

This clause is used to determine the status of the file; the data-name should be defined as a two character alphanumeric field like 00, 30, 9x etc. some of them are listed in the table on next page.

<table>
<thead>
<tr>
<th>Data-name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Successful execution</td>
</tr>
<tr>
<td>05</td>
<td>A file is opened which is not present</td>
</tr>
<tr>
<td>10</td>
<td>End of file condition</td>
</tr>
<tr>
<td>30</td>
<td>A permanent error exits; no further information is available</td>
</tr>
<tr>
<td>9X</td>
<td>An error condition defined by the particular system in use</td>
</tr>
</tbody>
</table>
1.2.5 FILE DESCRIPTION

The file description (FD) of the DATA DIVISION is used to describe the general behavior of the file. Firstly here we describe FD w.r.t. records of fixed length.

FD file-name

[ ; BLOCKS CONTAINS integer-1 { RECORDS CHARACTER } ]

[ ; RECORD CONTAINS integer-2 CHARACTER ]

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

[ ; VALUE OF implementor-name-1 IS { data-name-1 literal-1 } ]

[ ; implementor-name-2 IS { data-name-2 literal-2 } ] ....

[ ; DATA { RECORD IS data-name-3 [ data-name-4 ]... RECORDS ARE literal-3 [ data-name-4 ]... ] ]

[ ; CODE-SET is alphabet-name ]

1.2.5.1 BLOCK CONTAINS

Integer-1 of this clause determines the size of the block in terms of records (or character), if the block size is calculated in terms of the
records than it should be a multiple of the record size. By default one block is consisted with one record.

1.2.5.2 RECORD CONTAINS

In this clause integer-2 specifies the record size i.e. the numbers of characters in a record. This clause is used only for the documentation purpose.

1.2.5.3 LABEL RECORD

This clause is related with the header and trailer of the file as a label. Here the word STANDARD means that the file is associated with a header and trailer and OMITTED means file is unlabeled.

1.2.5.4 VALUE OF

The VALUE OF has been marked as being obsolete in the revised versions of COBOL. This clause is implementation dependent, most of the time it is used to specify the title of the file.

1.2.5.5 DATA RECORD

This clause is used to identify the name of the record(s) in the file, so that better documentation can be achieved.

1.2.5.6 CODE-SET
It specifies that in which code data is stored on the external medium. It is normally used in case of magnetic tapes.

1.2.6 OPEN AND CLOSE VERBS FOR SEQUENTIAL FILES

The processing of a file is initiated with the OPEN verb. There are four different open modes of a file:

1) **INPUT**  
2) **OUTPUT**  
3) **EXTEND**  
4) **I-O**

Whenever data is to be input in a file it must be in the INPUT mode and when a new file is created first time, it should be opened in the OUTPUT mode. On the other hand EXTEND mode also open a file for output, but the file positioning is following the last record on the existing file. In case of I-O mode records can be read through the READ statement and can be write through the REWRITE statement (write statement can’t be used in case of I-O mode), this mode is available only with the disk files.

Combinations of OPEN mode and INPUT-OUTPUT verbs:-

<table>
<thead>
<tr>
<th>OPEN MODE</th>
<th>INPUT</th>
<th>OUTPUT</th>
<th>I-O</th>
<th>EXTEND</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STATEMENT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Syntax for the OPEN statement:

\[
\text{OP} \left\{ \begin{array}{c}
\text{INPUT} \\
\text{OUTPUT} \\
\text{EXTEND I-O}
\end{array} \right\} \text{ File-name-1 [ , file-name-2 ] ...}
\]

Syntax for the CLOSE statement:

\[
\text{CLOSE file-name-1 [ WITH LOCK] [ , file-name-2 [WITH LOCK] ] ...}
\]

CLOSE terminates the processing of the file through the IOCS end of file operation. Whenever a CLOSE statement is executed for a file then that file must be in the open mode.

For example:

.....

PROCEDURE DIVISION.
OPEN-PARA.

OPEN INPUT INFILE,
OUTPUT PRINTFILE
EXTEND TRANSFILE
I-O MASTERFILE.

.....

CLOSE-PARA.
CLOSE INFILE, PRINTFILE, TRANSFILE, MASTERFILE.

1.2.7 READ, WRITE, AND REWRITE VERBS

To manipulate files, COBOL provides the following verbs: READ, WRITE and REWRITE. These verbs are described in the following paragraphs.

1.2.7.1 READ Verb

READ verb is used to make available the next logical record for processing from an input file. A READ statement must be executed before the data from a record can be processed. When a read operation for all the records of a file is complete i.e. after the “end-of-file”, the statement followed by the AT END clause will be executed. Hence a READ verb performs two operations, one makes available the data for processing and secondly it also determines what to do as the end-of-file comes.

Note: An “AT END” must be included in READ statement in case of sequential input file.

Syntax for READ VERB

```
READ       file-name [ NEXT] RECORD [ INTO  identifier ]

[ AT END   imperative-statement-1 ]

[ NOT AT END imperative-statement-2 ]

[ END-READ]
```

For example:

```
PROCEDURE DIVISION.
```
1.2.7.2 WRITE Verb

The WRITE verb is used to release a logical record for insertion in an output file. Some time it is also used for the vertical positioning of lines within a logical page (similar to indent in word).

Syntax of WRITE verb

```
WRITE record-name [FROM identifier-1]
{ BEFORE } ADVANCING { { Integer-1 Identifier-2 } [ Line Lines ] } 
{ AFTER }
{ mnemonic-name hardware-name }
```

For example:

```
PROCEDURE DIVISION.

WRITE-Para.

WRITE OUTREC.
WRITE OUTREC FROM HEADING1.
WRITE OUTREC FROM DETAILREC.

....
```
1.2.7.3 REWRITE verb

In case of disk files, REWRITE is used to update the existing records, after the REWRITE statement the record is no longer available. The REWRITE statement is used as a special case when the file is opened in the I/O mode and must be preceded by a READ statement.

Syntax of REWRITE

REWRITE record-name [FROM identifier]

1.2.8 Some Sample Programs for File Handling:

Program1: This program demonstrates how to use data files. It calls PRINTFILE to write some records to a data file and INFILE to read the same records back (without opening or closing the file between calls INFILE displays the output.

IDENTIFICATION DIVISION.
PROGRAM-ID. FILE-HANDLING.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT FINFILE ASSIGN TO "ISAMFIL.DAT"
ORGANIZATION IS INDEXED
RECORD KEY IS FD-TRAN-DATE
ACCESS MODE IS DYNAMIC.

DATA DIVISION.
FILE SECTION.
FD FINFILE
RECORD CONTAINS 50 CHARACTERS.

01 FD-FINFILE-RECORD.
   05 FD-TRAN-DATE   PIC X(4).
   05 FD-WITH-OR-DEP PIC X(2).
   05 FD-AMOUNT      PIC 9(5)V99.

PROCEDURE DIVISION.
MAIN-LINE.
   PERFORM OPEN-FILE
   PERFORM WRITE-TO-THE-FILE
   PERFORM START-FILE
   PERFORM READ-THE-FILE
   PERFORM CLOSE-FILE
   STOP RUN.

OPEN-FILE.
   OPEN I-O FINFILE.

START-FILE.
   MOVE 1111 TO FD-TRAN-DATE
   START FINFILE KEY = FD-TRAN-DATE.

WRITE-TO-THE-FILE.
   CALL "PRINTFILE".

READ-THE-FILE.
   CALL "INFILE".

CLOSE-FILE.
   CLOSE FINFILE.

IDENTIFICATION DIVISION.
PROGRAM-ID. INFILE.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT FINFILE ASSIGN TO "ISAMFIL.DAT"
    ORGANIZATION IS INDEXED
    RECORD KEY IS FD-TRAN-DATE
    ACCESS MODE IS DYNAMIC.

DATA DIVISION.
FILE SECTION.
FD FINFILE
  IS EXTERNAL
  RECORD CONTAINS 50 CHARACTERS.
01 FD-FINFILE-RECORD.
   05 FD-TRAN-DATE PIC X(4).
   05 FD-WITH-OR-DEP PIC X(2).
   05 FD-AMOUNT PIC 9(5)V99.

WORKING-STORAGE SECTION.
01 WS-END-OF-FILE PIC 9 VALUE 0.
01 WS-SUBTOTAL PIC S9(5)V99 VALUE 0.
01 WS-TOTAL PIC -(4)9.99.

PROCEDURE DIVISION.
MAIN-LINE.
   PERFORM READ-THE-FILE.
   PERFORM UNTIL WS-END-OF-FILE = 1
   PERFORM CALCULATE-TOTALS
      READ-THE-FILE
   END-PERFORM.
   PERFORM DISPLAY-OUTPUT.
EXIT PROGRAM.
STOP RUN.
READ-THE-FILE.
   READ FINFILE NEXT RECORD AT END
   MOVE 1 TO WS-END-OF-FILE.

CALCULATE-TOTALS.
   EVALUATE FD-WITH-OR-DEP
      WHEN "WI"
         SUBTRACT FD-AMOUNT FROM WS-SUBTOTAL
      WHEN "DE"
         ADD FD-AMOUNT TO WS-SUBTOTAL
   END-EVALUATE.

DISPLAY-OUTPUT.
   MOVE WS-SUBTOTAL TO WS-TOTAL
   DISPLAY "ACCOUNT BALANCE = ", WS-TOTAL.

END PROGRAM INFILE.
***********************************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. PRINTFILE.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT FINFILE ASSIGN TO "ISAMFIL.DAT"
      ORGANIZATION IS INDEXED
      RECORD KEY IS FD-TRAN-DATE
      ACCESS MODE IS DYNAMIC.
DATA DIVISION.
FILE SECTION.
FD FINFILE
   IS EXTERNAL
   RECORD CONTAINS 50 CHARACTERS.
01 FD-FINFILE-RECORD.
   05 FD-TRAN-DATE   PIC X(4).
PROCEDURE DIVISION.
MAIN-LINE.
   PERFORM WRITE-RECORDS
   EXIT PROGRAM
   STOP RUN.

WRITE-RECORDS.

WRITE A WITHDRAWAL RECORD
   MOVE 1111 TO FD-TRAN-DATE.
   MOVE 'WI' TO FD-WITH-OR-DEP.
   MOVE 23.55 TO FD-AMOUNT.
   WRITE FD-FINFILE-RECORD.

WRITE A DEPOSIT RECORD
   MOVE 2222 TO FD-TRAN-DATE.
   MOVE 'DE' TO FD-WITH-OR-DEP.
   MOVE 123.55 TO FD-AMOUNT.
   WRITE FD-FINFILE-RECORD.

END PROGRAM PRINTFILE.

In this program, a sequence number has been assigned to each line.
* in the seventh column indicates a comment statement.

000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. PHONEPROG.
000300*======================================
000400* This program creates a new data file if necessary
000500* and adds records to that file from entered from keyboard
000600*======================================
000700*
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT OPTIONAL PHONE-FILE
SELECT PHONE-FILE
ASSIGN TO "phone.dat"
ASSIGN TO "phone"
ORGANIZATION IS SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD  PHONE-FILE
LABEL RECORDS ARE STANDARD.
01  PHONE-RECORD.
05  PHONE-LAST-NAME PIC X(20).
05  PHONE-FIRST-NAME PIC X(20).
05  PHONE-NUMBER PIC X(15).

WORKING-STORAGE SECTION.
* Variables for SCREEN ENTRY
01  MESSAGE-1 PIC X(9) VALUE "Last Name".
01  MESSAGE-2 PIC X(10) VALUE "First Name".
01  MESSAGE-3 PIC X(6) VALUE "Number".
01  YES-NO PIC X.
01  ENTRY-OK PIC X.

PROCEDURE DIVISION.
MAIN-LOGIC SECTION.
PROGRAM-BEGIN.
PERFORM OPENING-PROCEDURE.
MOVE "Y" TO YES-NO.
PERFORM ADD-RECORDS
UNTIL YES-NO = "N".
PERFORM CLOSING-PROCEDURE.
PROGRAM-DONE.
STOP RUN.

OPENING AND CLOSING

OPENING-PROCEDURE.

OPEN EXTEND PHONE-FILE.

CLOSING-PROCEDURE.

CLOSE PHONE-FILE.

ADD-RECORDS.

MOVE "N" TO ENTRY-OK.

PERFORM GET-FIELDS UNTIL ENTRY-OK = "Y".

PERFORM ADD-THIS-RECORD.

PERFORM GO-AGAIN.

GET-FIELDS.

MOVE SPACE TO PHONE-RECORD.

DISPLAY MESSAGE-1 " ? ".

ACCEPT PHONE-LAST-NAME.

DISPLAY MESSAGE-2 " ? ".

ACCEPT PHONE-FIRST-NAME.

DISPLAY MESSAGE-3 " ? ".

ACCEPT PHONE-NUMBER.

PERFORM VALIDATE-FIELDS.

VALIDATE-FIELDS.

MOVE "Y" TO ENTRY-OK.

IF PHONE-LAST-NAME = SPACE

   DISPLAY "LAST NAME MUST BE ENTERED"

   MOVE "N" TO ENTRY-OK.

ADD-THIS-RECORD.

WRITE PHONE-RECORD.

GO-AGAIN.

DISPLAY "GO AGAIN?".

ACCEPT YES-NO.

IF YES-NO = "y"

   MOVE "Y" TO YES-NO.
1.3 Summary

- A physical file is a named area of a disk containing some sort of data.

- A logical file in COBOL is a physical file that is organized into fields and records.

- Accessing a file in COBOL requires both a logical and a physical definition of the file.

- The physical definition of the file is created with a SELECT statement in the I-O CONTROL paragraph of the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION.

- The logical definition of a file is created with an FD in the FILE SECTION of the DATA DIVISION and includes the record layout.

- A file can be opened in four modes: EXTEND, OUTPUT, I-O, and INPUT. EXTEND creates a new file, or opens an existing one, and allows records to be added to the end of the file. OUTPUT creates a new file--or destroys an existing file and creates a new version of it--and allows records to be added to the file. INPUT opens an existing file for reading only and returns an error if the file does not exist. I-O mode opens a file for reading and writing and causes an error if the file does not exist.

- The errors caused by INPUT mode and I-O mode when you attempt to open a file that does not exist can be changed by including the OPTIONAL clause in the SELECT statement for the file, if your compiler allows it.
• Use CLOSE with a filename to close an open file, regardless of the open mode.

• Use WRITE with a file-record to write a record to a file.

• Read the next record in a file by using READ filename NEXT RECORD. The READ NEXT command includes syntax to allow you to set a flag when the file has reached the end or last record.

• These are the three parts to processing a file sequentially and organizing the logic:

  • Set a flag to reflect a "not-at-end" condition and read the first record.
  • Perform the processing loop until the file is at end.
  • Read the next record at the bottom of the processing loop.

1.4 Key Words

OPEN, CLOSE, READ, WRITE AND REWRITE, FD ETC.

1.5 Self Assessment Questions (SAQ)

1. What are the file parameters in COBOL? Explain them with examples.
2. What is the need of file control specifications? How these are implemented in COBOL?
3. What is the significance of the COBOL file description? Discuss it with syntax.
4. Explain the role of the OPEN and CLOSE verbs in COBOL.
5. Write short notes on: READ, WRITE and REWRITE verbs.
6. What are different types of file organizations supported by COBOL.
7. Discuss the COBOL verbs needed to create and manipulate sequential files.
8. Discuss the COBOL verbs needed to create and manipulate Indexed files.
9. Discuss the COBOL verbs needed to create and manipulate relative files.
10. Write a COBOL program to create line sequential file.
11. Write a COBOL program to create sequential file.
12. Write a COBOL program to create indexed file.
13. Write a COBOL program to create relative file.

1.6 References/Suggested Readings

- COBOL Programming by M.K.Roy and D..Dastidar ; TMH
- Schaum’s outline series Programming with Structured COBOL ; MGH
- Comprehensive COBOL, vol-I ,Fundamentals of COBOL programming, 4/e by A.S.Philippakis and Leonard J. Kazmier ; TMH
- Comprehensive COBOL, vol-II , Advanced COBOL programming, 4/e by A.S.Philippakis and Leonard J. Kazmier ; TMH
- Structured COBOL: Fundamentals and style, 4/e by Welburn ; TMH
- Computer Programming in COBOL by V.Rajaraman; PHI
- Fundamentals of Structured COBOL Programming by Carl Feingold; Galgotia Booksource.
1.0 Objectives

- Understand why you might want to sort a file as part of your solution to a programming problem.
- Understand the role of the temporary work file and the USING and GIVING files.
- Be able to apply the SORT to sort a file on ascending or descending or multiple keys.
- Understand why you might want to use an INPUT PROCEDURE or an OUTPUT PROCEDURE to filter or alter records.
- Know the difference between an INPUT PROCEDURE and an OUTPUT PROCEDURE and know when to use one, and when the other.
- Be able to use the MERGE verb to merge two or more files.
- Understand the significance of the merge keys.

1.1 Introduction

The sequential files maintenance requires arranging the contents of the file in some predefined sequence. The process of sequencing the data in a predefined manner is known as sorting. The sorting can be done either in ascending or descending order on the key data item(s) of the records. Some
COBOL versions allow sorting of a file on up to 12 different keys, in any combination of ascending or descending sequence.

When you sort a file on more than one key, the most important key is called major key, the least important is called minor key and the rest of the keys are called intermediate keys. Table 13.1 shows different types of keys.

<table>
<thead>
<tr>
<th>Department# (Major key)</th>
<th>Section# (Intermediate key)</th>
<th>Student-id# (Minor key)</th>
<th>Stud-name# (Not a key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Sc.</td>
<td>A</td>
<td>1234567547</td>
<td>Dixhant</td>
</tr>
<tr>
<td>Electrical</td>
<td>B</td>
<td>4857519849</td>
<td>Kashis</td>
</tr>
<tr>
<td>Information Tech.</td>
<td>A</td>
<td>4535987600</td>
<td>Sorabh</td>
</tr>
<tr>
<td>Computer Sc.</td>
<td>C</td>
<td>4759814773</td>
<td>Saksh</td>
</tr>
<tr>
<td>Mechanical</td>
<td>B</td>
<td>7834646688</td>
<td>Rajash</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>C</td>
<td>8327358435</td>
<td>Deepika</td>
</tr>
</tbody>
</table>

Table 13.1 Sorting Keys

The sorting process involves three files – INPUT-FILE, SORT-FILE and SORTED FILE as shown in Fig 13.1.

![Fig 13.1 Sorting Process](image)

The SORT-FILE represents a programmed file whose description is embedded in the sorting routine. As shown in Fig 13.1 data from INPUT–FILE are submitted to the SORT-FILE where they are sorted and the sorted data is then sent to the SORTED-FILE i.e. the input file is not disturbed; instead a new output file (SORTED-FILE) containing the records in a sorted order is created.
As you have seen in the last chapter, while processing sequential files, it is possible to apply processing to an ordered sequential file that is difficult, or impossible, when the file is unordered. When this kind of processing is required, and the data file you have to work with is an unordered Sequential file, then part of the solution to the problem must be to sort the file. COBOL provides the SORT verb for this purpose.

Sometimes, when two or more files are ordered on the same key field or fields, you may want to combine them into one single ordered file. COBOL provides the MERGE verb for this purpose.

This chapter discusses the syntax, semantics, and use of the SORT and MERGE verbs.

1.2 Presentation of contents

1.2.1 Sorting Files using SORT verb

In COBOL programs, the SORT verb is usually used to sort Sequential files. Some programmers say that SORT verb is unnecessary. But one major advantage of using the SORT verb is that it enhances the portability of COBOL programs. Because the SORT verb is available in every COBOL compiler, when a program that uses the SORT verb has to be moved to a different computer system, it can make the transition without requiring any changes to the SORT.

Sometimes, it is difficult to apply processing if the file is unordered and it becomes easier if the file is ordered. In these situations, an obvious part of the solution is to sort the file.
1.2.1.1 SORT VERB

The SORT verb, syntax of which is given Fig 13.2 is used to sort a file and is written in the PROCEDURE DIVISION.

```
SORT SDWorkFileName
  [ON {ASCENDING DESCENDING} ] KEY {SortKeyIdentifier} ...
  {WITHDUPLICATESINORDER }
  {COLLATINGSEQUENCE IS AlphabetName}
  USING {InFileName} ...
  GIVING {OutFileName} ...
```

**Fig 13.2 Syntax of SORT verb**

- The SORT can be used anywhere in the PROCEDURE DIVISION except in an INPUT or OUTPUT PROCEDURE, or another SORT, or a MERGE, or in the DEclaratives SECTION.

- The records described for the input file (USING) must be able to fit into the records described for the **SDWorkFileName**.

- The records described for the **SDWorkFileName** must be able to fit into the records described for the output file (GIVING).
• The **SortKeyIdentifer** description cannot contain an OCCURS clause (i.e., it can't be a table/array) nor can it be subordinate to an entry that does contain one.

• **InFileName** and **OutFileName** files are automatically opened by the SORT. When the SORT executes they must not be open already.

• The **SDWorkFileName** identifies a temporary work file that the SORT process uses for the sort. It is defined in the FILE SECTION using an SD (Stream/Sort Description) entry. Even though the work file is a temporary file, it must still have an associated SELECT and ASSIGN clause in the ENVIRONMENT DIVISION.

• The **SDWorkFileName** file is a Sequential file with an organization of RECORD SEQUENTIAL. Since this is the default organization it is usually omitted.

• Each **SortKeyIdentifer** identifies a field in the record of the work file. The sorted file will be in sequence on this key field(s).

• When more than one **SortKeyIdentifer** is specified, the keys decrease in significance from left to right (leftmost key is most significant, rightmost is least significant).
• *InFileName* and *OutFileName*, are the names of the input and output files respectively.

• If the DUPLICATES clause is used then, when the file has been sorted, the final order of records with the duplicate keys is the same as that in the unsorted file. If no DUPLICATES clause is used, the order of records with duplicate keys is undefined.

• *AlphabetName* is an alphabet-name defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION. This clause is used to select the character set the SORT verb uses for collating the records in the file. The character set may be ASCII (8 or 7 bit), EBCDIC, or user-defined.

The syntax of sort description (SD) entry (written in the DATA DIVISION) is shown in Fig 13.3.

\[
\text{SD} \quad \text{file-name} \\
\left[; \text{RECORD CONTAINS} \left[ \text{integer-1 TO integer-2 CHARACTER} \right] \right] \\
\left[; \text{DATA} \left\{ \text{RECORD IS } \text{RECORDS ARE} \right\} \text{data-name-1} [,\text{data-name-2}] \ldots \right]
\]

**Fig 13.3** Sort Description file entry

There can be any number of SORT statements in a COBOL program and sorting can be done on any number of keys (limit is put by the
compiler). All the sorting keys must appear according to their descriptions in the record description of the input file. If there are records with identical keys, then their relative order within the input-file may not be retained.

The efficiency of a multi key sort can be improved by grouping the key fields together and sorting on the group item as shown below in the program code where SORT-KEYS is the group of keys.

```plaintext
DATA DIVISION.
FILE SECTION.
SD SORT-FILE
   ASSIGN TO "WORK.TMP".
01 SORT-RECORD.
   05 SORT-KEYS.
      10 REGION-ID PIC X(4).
      10 COLLEGE-ID PIC X(5).
      10 STUDENT-ID PIC X(10).
      05 REGION-NAME PIC X(15).
      05 COLLEGE-NAME PIC X(20).
      05 STUDENT-NAME PIC X(15).
```

The grouping can be used when any of the following conditions occur:

- When the keys are mutually adjacent within the records.
- When all the keys are alphanumeric or unsigned numeric, with USAGE DISPLAY
- When arrangement of keys in the file is from major (most important) to minor key (least important).

Example: The following program illustrates the use of SORT verb
IDENTIFICATION DIVISION.
PROGRAM-ID. ABC.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT IN-FILE
    ASSIGN TO "IN.DAT"
    ORGANIZATION IS LINE SEQUENTIAL.

  SELECT SORT-FILE
    ASSIGN TO "SORTED.DAT"
    ORGANIZATION IS LINE SEQUENTIAL.

  SELECT WORK-FILE
    ASSIGN TO "WORK.TMP".

DATA DIVISION.
FILE SECTION.
FD IN-FILE.
  01 IN-REC.
    02 SubscriberNumCF   PIC 9(8).
    02 UnitsUsedCF       PIC 9(5).

FD SORT-FILE.
  01 SORT-REC.
    02 SubscriberNumSF   PIC 9(8).
    02 UnitsUsedSF       PIC 9(5).

SD WORK-FILE.
  01 WORK-REC.
    02 SubscriberNumWF   PIC 9(8).
    02 UnitsUsedWF       PIC 9(5).
PROCEDURE DIVISION.
SORT-PARA.
  SORT WORK-FILE ON ASCENDING SubscriberNumWF
    USING IN-FILE
    GIVING SORT-FILE.

1.2.1.2 SORTING with an INPUT PROCEDURE

Some times, not all the records in an unsorted file are required in the sorted file. Other times, it may be that the sorted file records require additional, modified, or fewer fields, than the unsorted records. In these cases, an INPUT PROCEDURE can be used to eliminate unwanted records, or to change the format of the records, before they are submitted to the sort process.

Since sorting is a disk-based process, and thus comparatively slow, every effort should be made to reduce the amount of data that has to be sorted. The syntax of INPUT PROCEDURE is as given below:

\[
\text{SORT } SDWorkFileName \left\{ \text{ON} \begin{cases} \text{ASCENDING} \\
  \text{DESCENDING} \end{cases}, \text{KEY} \{\text{SortKeyIdentifiers}\} \ldots \right\} \ldots
\quad \left[\text{WITH Duplicates IN ORDER} \right]
\quad \left[\text{COLLATING SEQUENCE IS AlphabetName} \right]
\quad \left\{ \text{INPUT PROCEDURE IS ProcName} \left\{ \text{THRU through} \right\} \text{ProcName} \right\}
\quad \text{USING} \{\text{InFileName}\} \ldots
\quad \text{GIVING} \{\text{OutFileName}\} \ldots
\]\n
• When an INPUT PROCEDURE is used, it replaces the USING phrase. The **ProcName** in the INPUT PROCEDURE phrase identifies a block of code, that uses the RELEASE verb to supply records to the sort process.

• The INPUT PROCEDURE must finish before the sort process sorts the records supplied to it by the procedure. That's why the records are released to the work file. They are stored there until the INPUT PROCEDURE finishes and then they are sorted.

• An INPUT PROCEDURE allows us to select which records, and what type of records, will be submitted to the sort process. Because an INPUT PROCEDURE executes before the sort process sorts the records, only the data that is actually required in the sorted file will be sorted.

• The INPUT PROCEDURE must contain at least one RELEASE statement to transfer the records to the **SDWorkFileName**.

• The old COBOL rules for the SORT verb stated that the INPUT and OUTPUT procedures had to be self-contained sections of code, and could not be entered from elsewhere in the program.
In COBOL '85, INPUT and OUTPUT procedures can be any contiguous group of paragraphs or sections. The only restriction is that the range of paragraphs or sections used must not overlap.

Examples of INPUT PROCEDURE:

```
SORT WorkFile ON ASCENDING Dept-No, RollNo
   INPUT PROCEDURE RejectDropOut
   GIVING SortedFile.
```

```
SORT WorkFile ON ASCENDING DeptNo
   INPUT PROCEDURE IS SelectForeignStud
   GIVING SortedForeignStudFile.
```

```
SORT WorkFile ON ASCENDING Dept-No, RollNo
   INPUT PROCEDURE IS ComputerRecords
   GIVING SortedFile.
```

### 1.2.1.3 SORTING with an OUTPUT PROCEDURE

An OUTPUT PROCEDURE is used to retrieve sorted records from the work file using the RETURN verb. An OUTPUT PROCEDURE only executes after the file has been sorted.

The advantage of an INPUT PROCEDURE (as discussed in the previous section) is that it allows us to filter, or alter, records before they are supplied to the sort process and this can substantially reduce the amount of data that has to be sorted.
An OUTPUT PROCEDURE has no such advantage. An OUTPUT PROCEDURE only executes when the sort process has already sorted the file. The syntax of OUTPUT PROCEDURE is as shown below.

```
SORT SDWorkFileName
  ON {ASCENDING | DESCENDING} KEY {SortKeyIdentifier} ...
  [WITH DUPLICATES IN ORDER]
  [COLLATING SEQUENCE IS AlphabetName]
  INPUT PROCEDURE IS ProcName {THRU THROUGH ProcName} }
  USING {inFileName} ...
  OUTPUT PROCEDURE IS ProcName {THRU THROUGH ProcName} }
  GIVING {outFileName} ...
```

- An OUTPUT PROCEDURE uses the RETURN verb to retrieve sorted records from the work file. An OUTPUT PROCEDURE must contain at least one RETURN statement to get the records from the SortFile.

- The SORT...GIVING phrase cannot be used if an OUTPUT PROCEDURE is used.

- An OUTPUT PROCEDURE can perform anything you like with the records it gets from work file. For example, it can put them into an array, display them on the screen, or send them to an output file.
• When the OUTPUT PROCEDURE sends records to an output file, it can control which records, and what type of records, appear in the file.

• An OUTPUT PROCEDURE is used because, until the records have been sorted into some order, the records cannot be summed.

• An OUTPUT PROCEDURE uses the RETURN verb to read sorted records from the work file declared in the Sort's SD entry. The syntax of the RETURN verb is as shown below:

```
RETURN SDFileName RECORD [INTO Identifier]
    AT END StatementBlock
END-RETURN
```

where SDFileName is the name of the file declared in the SD entry.

• An operational template for an OUTPUT PROCEDURE, which gets records from the work file and writes them to an output file, is shown in the table below. Notice that the work file is not opened by the code in the OUTPUT PROCEDURE. The work file is automatically opened by the SORT verb.

• Nevertheless, an OUTPUT PROCEDURE is useful when you don't need to preserve the sorted file. For example, if you are sorting records to produce a once-off report, you can use an OUTPUT PROCEDURE
to create the report directly, without first having to create a file containing the sorted records.

- An OUTPUT PROCEDURE is also useful when you want to change the structure of the records written to the sorted file. For instance, in the first example program below, we use an OUTPUT PROCEDURE to summarize the sorted records. The resulting sorted file contains summary records, rather than the detail records, contained in the unsorted file.

Examples to illustrate OUTPUT PROCEDURE:

```cobol
SORT WorkFile ON ASCENDING CustName
   INPUT PROCEDURE IS SelectEssentialCommodity
   OUTPUT PROCEDURE IS SummariseSortReport.

SORT WorkFile ON ASCENDING KEY Dept-No
   USING DeptFile
   OUTPUT PROCEDURE IS SummariseRep.
```

A complete COBOL program on SORT verb - This program analyses the a web site IndiaTourism file and uses an OUTPUT PROCEDURE to print a report showing the number of Tourists to the web site from the different countries.

IDENTIFICATION DIVISION.
PROGRAM-ID. SampleProg.
AUTHOR. ABC.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

SELECT IndiaTourismFile
    ASSIGN TO "IndiaTourism.Dat"
    ORGANIZATION IS LINE SEQUENTIAL.

SELECT WorkFile
    ASSIGN TO "Work.Tmp".

SELECT ForeignTouristReport
    ASSIGN TO "ForeignTourist.Rpt"
    ORGANIZATION IS LINE SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD IndiaTourismFile.
  01 TouristRec.
      88 EndOfFile VALUE HIGH-VALUES.
      02 TouristNameGF PIC X(20).
      02 CountryNameGF PIC X(20).
          88 CountryIsPakistan VALUE "PAKISTAN".
      02 TouristCommentGF PIC X(40).

SD WorkFile.
  01 WorkRec.
      88 EndOfFile VALUE HIGH-VALUES.
      02 CountryNameWF PIC X(20).

FD ForeignTouristReport.
  01 PrintLine PIC X(38).
WORKING-STOREAGE SECTION.
01 Heading1 PIC X(37)
   VALUE " Web site - Foreign Tourists Report".

01 Heading2.
  02 FILLER PIC X(25) VALUE " Country".
  02 FILLER PIC X(8) VALUE "Tourists".

01 CountryLine.
  02 FILLER PIC X(3) VALUE SPACES.
  02 PrnCountryName PIC X(20).
  02 PrnTouristCount PIC BBBZZ,ZZ9.

01 ReportFooting PIC X(36)
   VALUE "**** End of Foreign Tourists report ****".

01 TouristCount PIC 9(5).

PROCEDURE DIVISION.
PRINTFOREIGNTOURISTREPORT.
  SORT WorkFile ON ASCENDING CountryNameWF
  INPUT PROCEDURE IS SelectForeignTourists
  OUTPUT PROCEDURE IS PrintTouristReport.
  STOP RUN.

SELECTFOREIGNTOURISTS.

  OPEN INPUT IndiaTourismFile.
  READ IndiaTourismFile
  AT END SET EndOfFile TO TRUE
  END-READ
  PERFORM UNTIL EndOfFile
  IF NOT CountryIsPakistan
    MOVE CountryNameGF TO CountryNameWF
RELEASE WorkRec
END-IF
READ IndiaTourismFile
AT END SET EndOfFile TO TRUE
END-READ
END-PERFORM
CLOSE IndiaTourismFile.

PRINTTOURISTREPORT.
OPEN OUTPUT ForeignTouristReport
WRITE PrintLine FROM Heading1
   AFTER ADVANCING PAGE
WRITE PrintLine FROM Heading2
   AFTER ADVANCING 2 LINES
RETURN WorkFile
   AT END SET EndOfWorkFile TO TRUE
END-RETURN
PERFORM PrintReportBody UNTIL EndOfWorkFile
WRITE PrintLine FROM ReportFooting
   AFTER ADVANCING 3 LINES
CLOSE ForeignTouristReport.

PRINTREPORTBODY.
MOVE CountryNameWF TO PrnCountryName
MOVE ZEROS TO TouristCount
PERFORM UNTIL CountryNameWF NOT EQUAL TO PrnCountryName
   OR EndOfWorkFile
   ADD 1 TO TouristCount
RETURN WorkFile
   AT END SET EndofWorkFile TO TRUE
END-RETURN
END-PERFORM
MOVE TouristCount TO PrnTouristCount
1.2.2 MERGE VERB

It is often useful to combine two or more files into a single large file. If the files are unordered, this is easy to accomplish because you can simply append the records in one file to the end of the other. But if the files are unordered, the task is somewhat more complicated, especially if there are more than two files, because you must preserve the ordering in the combined file.

In COBOL, instead of having to write special code every time you want to merge files, you can use the MERGE verb. The MERGE verb takes two or more identically sequenced files and combines them, according to the key values specified. The combined file is then sent to an output file or an OUTPUT PROCEDURE. The syntax of MERGE verb is given below:

```
MERGE SDWorkFileName
    {ON {ASCENDING | DESCENDING} KEY {MergeKeyIdentifier} ...} ...

    [COLLATING SEQUENCE IS AlphabetName]
    USING inputFileName {inputFileName} ... 

    {OUTPUT PROCEDURE IS ProcName \{THRU ProcName \} }
    {GIVING {OutputFileName} ...}
```

- The results of the MERGE verb are predictable only when the records in the input files are ordered as described in the KEY clause associated with the MERGE statement. For instance, if the MERGE statement has
an ON DESCENDING KEY then all the USING files must be ordered on descending.

- As with the SORT, the SDWorkFileName is the name of a temporary file, with an SD entry in the FILE SECTION, a SELECT and ASSIGN entry in the INPUT-OUTPUT SECTION, and an organization of RECORD SEQUENTIAL.

- Each MergeKeyIdentifier identifies a field in the record of the work file. The sorted file will be in sequence on this key field(s).

- When more than one MergeKeyIdentifier is specified, the keys decrease in significance from left to right (leftmost key is most significant, rightmost is least significant).

- InFileName and OutFileName, are the names of the input and output files respectively. These files are automatically opened by the MERGE. When the MERGE executes they must not be already open.

- AlphabetName is an alphabet-name defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION. This clause is used to select the character set the SORT verb uses for collating the records in the file. The character set may be ASCII (8 or 7 bit), EBCDIC, or user-defined.
• The MERGE can use an OUTPUT PROCEDURE and the RETURN verb to get merged records from the SDWorkFileName.

• The OUTPUT PROCEDURE only executes after the files have been merged and must contain at least one RETURN statement to get the records from the SortFile.

For example:

MERGE MergeWorkFile
  ON ASCENDING KEY TransDate, TransCode, StudentId
  USING InsertTransFile, DeleteTransFile, UpdateTransFile
  GIVING CombinedTransFile.

Here is an outline of a COBOL MERGE program:
  DATA DIVISION.
  FILE SECTION.
    FD SEMESTER-FIRST LABEL RECORDS STANDARD
        DATA RECORDS MARKS-OBTAINED.
      01 MARKS-DETAIL.
        02 DEPT-CODE PIC 99.
        02 REG-NO PIC999999
        .
        .
    FD SEMESTER-SECOND...
        .
        .
    FD SEMESTER-THIRD...
        .
        .
FD SEMESTER-FORTH...
.
.
.
FD RESULT LABEL RECORDS STANDARD
DATA RECORD FINAL-RESULT.
01 FINAL-RESULT.
  02 DEPT-CODE PIC 99.
  02 REG-NO PIC 999999.
.
.
.
SD MERGE-FILE DATA-RECORD MERGE-RECORD.
01 MERGE-RECORD.
  02 DEPARTMENT PIC 99.
  02 REGISTRATION PIC 999999.
.
.
.
PROCEDURE DIVISION.
PARA-1.
.
.
MERGE MERGE-FILE ON ASCENDING KEY DEPARTMENT
ON ASCENDING KEY REGISTRATION
USING SEMESTER-FIRST, SEMESTER-SECOND,
SEMESTER-THIRD, SEMESTER-FORTH
GIVING RESULT.

A complete Program to illustrate MERGE verb: The program merges the file Students.Dat and Transins.Dat to create a new file Students.New

IDENTIFICATION DIVISION.
PROGRAM-ID. MergingFiles.
AUTHOR. ABC.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-COM Mand.
   SELECT StudentFile ASSIGN TO "STUDENTS.DAT"
       ORGANIZATION IS LINE SEQUENTIAL.

   SELECT InsertionsFile ASSIGN TO "TRANSINS.DAT"
       ORGANIZATION IS LINE SEQUENTIAL.

   SELECT NewStudentFile ASSIGN TO "STUDENTS.NEW"
       ORGANIZATION IS LINE SEQUENTIAL.

   SELECT WorkFile ASSIGN TO "WORK.TMP".

DATA DIVISION.
FILE SECTION.
FD StudentFile.
  01 StudentRec PIC X(30).

FD InsertionsFile.
  01 InsertionRec PIC X(30).

FD NewStudentFile.
  01 NewStudentRec PIC X(30).

SD WorkFile.
  01 WorkRec.
      02 StudentId PIC 9(7).
      02 FILLER PIC X(23).

PROCEDURE DIVISION.
MERGE-PARA.

MERGE WorkFile

ON ASCENDING KEY StudentId

USING InsertionsFile, StudentFile

GIVING NewStudentFile.

STOP RUN.

1.4 Summary

You can arrange records in a particular sequence by using the SORT or MERGE statement. You can mix SORT and MERGE statements in the same COBOL program.

SORT statement accepts input (from a file or an internal procedure) that is not in sequence, and produces output (to a file or an internal procedure) in a requested sequence. You can add, delete, or change records before or after they are merged.

MERGE statement compares records from two or more sequenced files and combines them in order. You can add, or change records before or after they are sorted. Describe the input file or files for sorting or merging by following the procedure below.

Write one or more SELECT clauses in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION to name the input files. For example:

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

SELECT Input-File ASSIGN TO InFile.
*Input-File* is the name of the file in your program. Use this name to refer to the file.

Describe the input file (or files when merging) in an **FD** entry in the **FILE SECTION** of the **DATA DIVISION**. For example:

```plaintext
DATA DIVISION.
FILE SECTION.
FD Input-File

   LABEL RECORDS ARE STANDARD
   BLOCK CONTAINS 0 CHARACTERS
   RECORDING MODE IS F
   RECORD CONTAINS 100 CHARACTERS.

   01 Input-Record PIC X(100).
```

Describe the sort file to be used for sorting or merging. You need **SELECT** clauses and **SD** entries even if you are sorting or merging data items only from **WORKING-STORAGE** or **LOCAL-STORAGE**.

Write one or more **SELECT** clauses in the **FILE-CONTROL** paragraph of the **ENVIRONMENT DIVISION** to name a sort file. For example:

```plaintext
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

   SELECT Sort-Work-1 ASSIGN TO SortFile.
```

*Sort-Work-1* is the name of the file in your program. Use this name to refer to the file.

Describe the sort file in an **SD** entry in the **FILE SECTION** of the **DATA DIVISION**. Every **SD** entry must contain a record description. For example:
DATA DIVISION.
FILE SECTION.
SD  Sort-Work-1
    RECORD CONTAINS 100 CHARACTERS.
    01  SORT-WORK-1-AREA.
        05  SORT-KEY-1   PIC  X(10).
        05  SORT-KEY-2   PIC  X(10).
        05  FILLER       PIC  X(80).

If the output from sorting or merging is a file, describe the file by following the procedure below.

Write a SELECT clause in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION to name the output file. For example:

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT Output-File ASSIGN TO OutFile.

*Output-File* is the name of the file in your program. Use this name to refer to the file.

Describe the output file (or files when merging) in an FD entry in the FILE SECTION of the DATA DIVISION. For example:

DATA DIVISION.
FILE SECTION.
FD  Output-File
    LABEL RECORDS ARE STANDARD
    BLOCK CONTAINS 0 CHARACTERS
    RECORDING MODE IS F
    RECORD CONTAINS 100 CHARACTERS.
    01  Output-Record  PIC X(100).
The file described in an *SD* entry is the working file used for a sort or merge operation. You cannot perform any input or output operations on this file and you do not need to provide a data definition for it.

A program can contain any number of sort and merge operations. They can be the same operation performed many times or different operations. However, one operation must finish before another begins.

### 1.5 Key Words

Sort, merge, using, file, key, input, output.

### 1.6 Self Assessment Questions (SAQ)

(i) What do you mean by the term sorting?
(ii) What is the concept of sort-key? Is it possible to sort a file on more than one key?
(iii) Give your comments on the following:

```
SORT ON ASCENDING KEY STU-ID-NUM
    USING STU-RECORDS-FILE
    GIVING SORT-OUT-FILE.
```

Determine here the major key, minor-key and the final sorted file.

(iv) Given two sorted files FILE-A and FILE-B. Write a program in COBOL to merge these files into FILE-C using MERGE verb?

(v) Determine name of the file with the FD entry, name of the merged file from the following statement of COBOL MERGE:

```
MERGE MERGE-FILE ON ASCENDING KEY DEPARTMENT
    ON ASCENDING KEY REGISTRATION
    USING SEMESTER-FIRST, SEMESTER-SECOND,
```
(vi) Differentiate between sorting and merging of files.

(vii) Write up a COBOL program to merge FILE-A, FILE-B and FILE-C to produce merged file MERGE-FILE as shown in the table below:

<table>
<thead>
<tr>
<th>FILE-A</th>
<th>FILE-B</th>
<th>FILE-C</th>
<th>MERGE-FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>40</td>
<td>25</td>
<td>15</td>
</tr>
<tr>
<td>50</td>
<td>60</td>
<td>35</td>
<td>20</td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>70</td>
</tr>
</tbody>
</table>

1.7 References/Suggested Readings

- COBOL Programming by M.K. Roy and D. Dastidar; TMH
- Schaum’s outline series Programming with Structured COBOL; MGH
- Comprehensive COBOL, vol-I, Fundamentals of COBOL programming, 4/e by A.S. Philippakis and Leonard J. Kazmier; TMH
- Comprehensive COBOL, vol-II, Advanced COBOL programming, 4/e by A.S. Philippakis and Leonard J. Kazmier; TMH
- Structured COBOL: Fundamentals and style, 4/e by Welburn; TMH
- Computer Programming in COBOL by V. Rajaraman; PHI
- Fundamentals of Structured COBOL Programming by Carl Feingold; Galgotia Booksource.
1.0 Objectives

- To identify the results obtained from the execution of data manipulation statements based upon the data specified.
- To describe the format and various aspects of the EXAMINE verb with its various options.
- To describe the format and various aspects of the INSPECT verb.
- To describe the format and various aspects of the STRING verb and UNSTRING verb to obtain the optimization use of the data file.
- Differentiate between the STRING and UNSTRING verbs.

1.1 Introduction

A group of characters is known as a string or we can say any field with DISPLAY usage can be considered as a string. There are a number of string manipulation operations like comparison, concatenation, segmentation, scanning and replacement. The string manipulation verbs supported by COBOL are:

- The EXAMINE is used to inspect data with or without the movement of the data.
- The INSPECT is an improvement of the EXAMINE verb with more power.
• The STRING and UNSTRING are used for the concatenation or segmentation of given strings.

1.2 Presentation of Contents

1.2.1 EXAMINE VERB

In the early years data manipulation in COBOL was limited by MOVE and EXAMINE verbs only. But now COBOL has introduced a very powerful verb EXAMINE. This verb can be used to search the frequency of a desired character in a given string. It can also be used to replace the said character by another character. This verb has three different forms. Syntax for each of these forms are given below:

SYNTAX-1

```
EXAMINE   identifier   TALLYING   { ALL
                       LEADING
                       UNTIL FIRST } literal-1
```

SYNTAX-2

```
EXAMINE   identifier   REPLACE   { ALL
                       LEADING
                       [UNTIL] FIRST } literal-2 BY Literal3
```

SYNTAX-3

```
EXAMINE   identifier   TALLYING   { ALL
                       LEADING
                       UNTIL FIRST } literal-4

                        REPLACING   BY literal-5
```

1.2.1.1 Descriptions of Syntax for EXAMINE verb:
• In SYNTAX -1, **ALL** phrase is used to scan the given string and match its characters for the literal-1. If the match is successful then the TALLY register is incremented by one.

• In SYNTAX -1, **LEADING** phrase is used, the contiguous repetition of the character (refer by literal-1) starting from the leading position of the identifier are examined, if match is successful then the TALLY register is incremented by one and the search terminates as soon as no match occurs.

• In SYNTAX -1, **UNTIL FIRST** phrase is used, TALLY register is incremented in response of every search of the character from the leftmost position of the given string and as a match of character (literal-1) is there search terminates.

• The identifier must have DISPLAY usage.

• Every literal class must be similar and it must be single character.

• Scanning process must starts from the left of the string.

• In case of TALLYING option a counter TALLY is used to store the results of it, and it must be initialized with zero at the time of its execution.

• In case of SYNTAX-2, all three phrases have similar significance except that instead of incremented the TALLY, the matched character is replaced by the specified character (literal-3) also.

• In case of SYNTAX-2, UNTIL is optional i.e. if used only FIRST then only the first appearance of the specified character (literal-2) is replaced by the character (literal-3). If UNTIL is used with FIRST then all the characters up to the first appearance of the character (literal-2).

• In case of SYNTAX-3, the result is the combined effects of SYNTAX-1 and SYNTAX-2 i.e. TALLY is incremented as well as match character is replaced by the specified character (literal-4).

Some examples based upon the above Syntax:-

Consider the entry in DATA DIVISION

```
77   A   PIC   X(5)   VALUE   IS   “BBADB”.
```
Statement in PROCEDURE DIVISION:

SYNTAX-1
EXAMINE A TALLYING ALL “B”
On execution of this statement, the value of the register TALLY will be 3 as there are total three B in the string “BBADB”.

EXAMINE A TALLYING LEADING “B”
In result of this statement TALLY=2, since there are only two leading B in “BBADB”.

EXAMINE A TALLYING UNTIL FIRST “D”
In result of this statement TALLY=3, since there are only three characters before the character D in “BBADB”.

SYNTAX-3
EXAMINE A TALLYING ALL “B” REPLACEING BY “K”
In result of this statement TALLY=3 and will change the contents of “A” to “KKADK”.

Note: - In case of ALL or LEADING phrase, if the desired character is not found, TALLY= 0.

1.2.2 INSPECT VERB

COBOL provides the INSPECT verb for character manipulations, this verb replaces EXAMINE verb which served the similar but limited purpose in old versions of COBOL. The INSPECT verb is more powerful but little bit complicated syntax with different ways, these are:-

SYNTAX-1
1.2.2.1 Description of SYNTAX-1 of INSPECT Verb

- All the identifier-n are elementary items except identifier-1 is a group, which must usage DISPLAY verb.
- All the scanning must be from left to right.
- Identifier-2 act as a register to store all the information which was stored in TALLY in case of EXAMINE verb.
- If the CHARACTERS phrase is used, identifier-2 is incremented (by one) for each character in identifier-1.
- The BEFORE and AFTER phrase is used as the length controller of the identifier-1, which is to be searched.
- All the other rules are same as in case of EXAMINE verb.

SYNTAX-2

INSPECT identifier-1 REPLACING
1.2.2.1 Description of SYNTAX-2 of INSPECT Verb

(1) In the result of this SYNTAX, matched characters are replaced by the specified characters refer by the identifier or literal after BY.

(2) The impact of ALL and LEADING phrase is similar as in case of SYNTAX-1, except in this case there is no count increment. But the matched characters are replaced w.r.t. specified character.

(3) In case of CHARACTERS phrase is used, identifier-5 or literal-3 is of single character.

(4) If the FIRST phrase is used, the leftmost appearance of identifier-7/literal-5 matched within the contents of identifier-1 is replaced by identifier-8 or literal-6.
1.2.3 Differences between EXAMINE and INSPECT

- The INSPECT statement permits the matching and replacement to be named in sequence, so a number of them can be used within a single statement.

- In case of INSPECT verb, the options BEFORE and AFTER play very important role, which are not present in case of EXAMINE verb.

- In case of EXAMINE we can compare only a single character whereas in case of INSPECT we can compare a group of characters. These characters can be counted and replaced.

- It is important that all the literals with INSPECT must be alphanumeric (enclosed within quotation marks).

- The INSPECT doest not use a field TALLY to count the characters in a field.

1.2.4 STRING AND UNSTRING VERBS

The STRING and UNSTRING verbs are used to transfer data from several sources into single destination or vice versa. A STRING verb is used to concatenate two or more characters to form a long string, on the other side UNSTRING verb, as names implies, acts in the reverse direction of the STRING verb, it is used for the segmentation of a long string in to many substrings of desired formats.

1.2.4.1 Syntax of the STRING Verb
1.2.4.1.1 Description of syntax of STRING verb

- STRING is used to concatenate two or more string side by side, the source strings come from identifiers/literals -1, 2, 4, 5 and the destination field is identifier-7.

- The source field may be alphanumeric literals, figurative constants (treated as single character) or identifier with usage DISPLAY. The destination field must also be with DISPLAY.
When DELIMITED BY SIZE phrase is used then the entire contents transferred from left to right into destination (identifier-7) until the right most character is shifted or destination is full. On the other side if DELIMITED is without SIZE then the process of transfer is stopped when: - (i) end of source strings are reached OR (ii) the specified character (refer in DELIMITED) is matched. OR (iii) identifier-7 is full.

The identifier/literal-3/6 in the DELIMITED phrase can denote one or more characters.

There can be a number of delimiters in a STRING statement, when a delimiter is encountered the transmission of characters stop.

The POINTER phrase is used to determine the left most location of the destination field (identifier-7). If the identifier-7< identifier-8 <1 then no transfer take place.

The STRING process is terminate as the end of the data item referred by the identifier-7 is reached or all the desired data has been transferred.

1.2.4.2 Syntax of UNSTRING verb

```
[DELIMITED BY [ALL] {identifier-2 literal-1} [ , OR [ALL] {identifier-3 literal-2} ] ... ]

UNSTRING identifier-1
```
1.2.4.2.1 Description of syntax of UNSTRING verb

- The data from the source (identifier-1 is an alphanumeric field with DISPLAY usage) is segmented and placed in various destinations (like identifier-4/7 are alphanumeric, alphabetic or numeric field with DISPLAY usage etc.).

- All the literals must be described as numbers and if it is figurative constant, it must be considered as single character. The identifiers-6/9/10/11 must be elementary integers.

- When DELIMITED BY phrase is used, the sending field is examined for the occurrence of the character(s) in DELIMITED BY phrase. If two adjacent delimiters are occurred without ALL then the first delimiter terminates the transfer of data to the current receiving field and the second delimiter will be the reason of filling of the next receiving field either by spaces or with zeros as per the description of the field.
• On the other side if the DELIMITED BY phrase is not used then the characters of the source field are transferred from left to right in destination field.

• When TALLYING phrase is used

\[ \text{identifier-11} = \text{initial value} + \text{number of receiving fields}. \]

1.3 Summary

• Data manipulation verbs are used to move data from one memory area to another with in the system. The EXAMINE verb is used to replace a given character/ or count the number of times a character appears in a data field.

• The TALLYING option of the EXAMINE is used to scan a data item, counting the number of occurrence of a given character.

• The REPLACING option is used to modify the value of an item by replacing certain characters in the original value with new characters.

• The INSPECT statement increases the power of the EXAMINE statement. The INSPECT is used in conjunction with character strings and examines the contents of a data item from left to right.

• The STRING statement causes characters from one or more data items to be transferred in to a single data item; here characters are transferred from the sending fields to the receiving field in a left-to-right order.
• The UNSTRING is basically the opposite of the STRING i.e. UNSTRING segmented a string into a number of fields according to the predefined condition.

1.4 Key words

examine, inspect, string, unstring, delimiter, tally

1.5 Self Assessment Questions (SAQ)

1. What are the main functions of the data manipulation statements?
2. What are the different data manipulation statements with their main functions?
3. What is the main purpose of the EXAMINE and its various options?
4. What is the main purpose of the INSPECT and its various options?
5. Briefly explain the TALLYING phrase of the EXAMINE?
6. Briefly explain the TALLYING phrase of the INSPECT?
7. What is the main function of the STRING statement?
8. Briefly explain the function of the UNSTRING statement?

1.6 References/Suggested Readings

1. COBOL Programming by M.K.Roy and D..Dastidar ; TMH
2. Schaum’s outline series Programming with Structured COBOL ; MGH
4. Comprehensive COBOL, vol-II, Advanced COBOL programming, 4/e by A.S.Philippakis and Leonard J. Kazmier ; TMH
5. Structured COBOL: Fundamentals and style, 4/e by Welburn ; TMH
6. Computer Programming in COBOL by V.Rajaraman; PHI
7. Fundamentals of Structured COBOL Programming by Carl Feingold; Galgotia Booksourse.